Hashicorp Vault + K8s

Что с нами будет?

- Ветка для работы: `kubernetes-vault`
- В ходе работы мы:
  - установим кластер vault в kubernetes
  - научимся создавать секреты и политики
  - настроим авторизацию в vault через kubernetes sa
  - сделаем под с контейнером nginx, в который прокинем секреты из vault через consul-template

Подготовка

- должен быть запущенный kubernetes кластер
- Все созданные в процесс ДЗ файлы должны быть в репозитории
- вспомогательные ссылки
  - vault
  - vault-guides
- лейбл homework-11
- юзернейм для assignee erlong15

Инсталляция hashicorp vault HA в k8s

- склонируем репозиторий consul (необходимо минимум 3 ноды)

```
git clone https://github.com/hashicorp/consul-helm.git
helm install --name=consul consul-helm
```

  - склонируем репозиторий vault

```
git clone https://github.com/hashicorp/vault-helm.git
```

Отредактируем параметры установки в values.yaml

```
 standalone:

enabled: false
....
ha:

enabled: true
...


ui:
enabled: true
serviceType: "ClusterIP"
```

Установим vault

```
helm install --name=vault .
helm status vault
kubectl logs vault-0
```

- обратите внимание на статус подов vault
- вывод helm status vault - добавьте в README.md

Инициализируем vault

- проведите инициализацию черерз любой под vault'a `kubectl exec -it vault-0 -- vault operator init --key-shares=1 --key-threshold=1`
- сохраните ключи, полученные при инициализации

```
Unseal Key 1: qpt7e1w2D2tQqPdknR8A5VFrzFZ0Yz6W/BPoFMX5x2A=
```

Initial Root Token: s.KZmV3wbjEJabNtcwJWIlOfSj

```
* вывод добавьте в README.md

* 🐍 поэкспериментируйте с разными значениями --key-shares --key-threshold
```

Проверим состояние vault'a

```
kubectl logs vault-0
```

- Обратите внимание на параметры Initialized, Sealed

```
kubectl exec -it vault-0 -- vault status
Key                Value
---                -----
Seal Type          shamir
Initialized        true
Sealed             true
Total Shares       1
Threshold          1
Unseal Progress    0/1
Unseal Nonce       n/a
Version            1.2.2
HA Enabled         true
```

Распечатаем vault

- Обратите внимание на переменные окружения в подах

```
kubectl exec -it vault-0 env | grep VAULT
VAULT_ADDR=http://127.0.0.1:8200
```

- Распечатать нужно каждый под

```
kubectl exec -it vault-0 -- vault operator unseal
'qpt7e1w2D2tQqPdknR8A5VFrzFZ0Yz6W/BPoFMX5x2A='
kubectl exec -it vault-1 -- vault operator unseal
'qpt7e1w2D2tQqPdknR8A5VFrzFZ0Yz6W/BPoFMX5x2A='
kubectl exec -it vault-2 -- vault operator unseal
'qpt7e1w2D2tQqPdknR8A5VFrzFZ0Yz6W/BPoFMX5x2A='
```

- добавьте выдачу `vault status` в README.md

Посмотрим список доступных авторизаций

- выполните `kubectl exec -it vault-0 -- vault auth list`
- получите ошибку

```
Error listing enabled authentications: Error making API request.

URL: GET http://127.0.0.1:8200/v1/sys/auth
Code: 400. Errors:

* missing client token
```

Залогинимся в vault (у нас есть root token)

```
kubectl exec -it vault-0 --  vault login

Token (will be hidden):
```

- Вывод после логина добавьте в README.md
- повторно запросим список авторизаций

```
kubectl exec -it vault-0 --  vault auth list
```

- Вывод сохранить в README.md

Заведем секреты

```
kubectl exec -it vault-0 -- vault secrets enable --path=otus kv
kubectl exec -it vault-0 -- vault secrets list --detailed
kubectl exec -it vault-0 -- vault kv put otus/otus-ro/config username='otus'
password='asajkjkahs'
kubectl exec -it vault-0 -- vault kv put otus/otus-rw/config username='otus'
password='asajkjkahs'
kubectl exec -it vault-0 -- vault read otus/otus-ro/config
kubectl exec -it vault-0 -- vault kv get otus/otus-rw/config
```

- выыод команды чтения секрета добавить в README.md

Включим авторизацию черерз k8s

```
kubectl exec -it vault-0 -- vault auth enable kubernetes
kubectl exec -it vault-0 --  vault auth list
```

- Обновленный список авторизаций - добавить в README.md

Создадим yaml для ClusterRoleBinding

```
$ tee vault-auth-service-account.yml <<EOF
    ---
    apiVersion: rbac.authorization.k8s.io/v1beta1
    kind: ClusterRoleBinding
    metadata:
      name: role-tokenreview-binding
      namespace: default
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: ClusterRole
      name: system:auth-delegator
    subjects:
    - kind: ServiceAccount
      name: vault-auth
      namespace: default
    EOF
```

- файл должен быть приложен в ДЗ

Создадим Service Account vault-auth и применим ClusterRoleBinding

```
# Create a service account, 'vault-auth'
$ kubectl create serviceaccount vault-auth

# Update the 'vault-auth' service account
$ kubectl apply --filename vault-auth-service-account.yml
```

Подготовим переменные для записи в конфиг кубер авторизации

```
export VAULT_SA_NAME=$(kubectl get sa vault-auth -o jsonpath="{.secrets[*]['name']}")
export SA_JWT_TOKEN=$(kubectl get secret $VAULT_SA_NAME -o jsonpath="{.data.token}" |
base64 --decode; echo)
export SA_CA_CRT=$(kubectl get secret $VAULT_SA_NAME -o jsonpath="{.data['ca\.crt']}" |
base64 --decode; echo)
export K8S_HOST=$(more ~/.kube/config | grep server |awk '/http/ {print $NF}')

### alternative way
export K8S_HOST=$(kubectl cluster-info | grep ‘Kubernetes master’ | awk ‘/https/ {print
$NF}’ | sed ’s/\x1b\[[0-9;]*m//g’ )
```

- Обратите внимание на конструкцию `sed ’s/\x1b\[[0-9;]*m//g`, что по вашему она делает?

Запишем конфиг в vault

```
kubectl exec -it vault-0 -- vault write auth/kubernetes/config \
token_reviewer_jwt="$SA_JWT_TOKEN" \
kubernetes_host="$K8S_HOST" \
kubernetes_ca_cert="$SA_CA_CRT"
```

Создадим файл политики

```
tee otus-policy.hcl <<EOF
path "otus/otus-ro/*" {
    capabilities = ["read", "list"]
}

path "otus/otus-rw/*" {
    capabilities = ["read", "create", "list"]
}
EOF
```

создадим политку и роль в vault

```
kubectl cp otus-policy.hcl vault-0:./

kubectl exec -it vault-0 -- vault policy write otus-policy /otus-policy.hcl

kubectl exec -it vault-0 -- vault write auth/kubernetes/role/otus  \
bound_service_account_names=vault-auth          \
bound_service_account_namespaces=default policies=otus-policy  ttl=24h
```

Проверим как работает авторизация

- Создадим под с привязанным сервис аккоунтом и установим туда curl и jq

```
kubectl run --generator=run-pod/v1 tmp --rm -i --tty --serviceaccount=vault-auth --image
alpine:3.7
apk add curl jq
```

- Залогинимся и получим клиентский токен

```
VAULT_ADDR=http://vault:8200
KUBE_TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
curl --request POST  --data '{"jwt": "'$KUBE_TOKEN'", "role": "otus"}'
$VAULT_ADDR/v1/auth/kubernetes/login | jq
TOKEN=$(curl -k -s --request POST  --data '{"jwt": "'$KUBE_TOKEN'", "role": "test"}'
$VAULT_ADDR/v1/auth/kubernetes/login | jq '.auth.client_token' | awk -F\" '{p
rint $2}')
```

Прочитаем записанные ранее секреты и попробуем их обновить

- используйте свой клиентский токен
- проверим чтение

```
curl --header "X-Vault-Token:s.pPjvLHcbKsNoWo7zAAuhMoVK"
$VAULT_ADDR/v1/otus/otus-ro/config
curl --header "X-Vault-Token:s.pPjvLHcbKsNoWo7zAAuhMoVK"
$VAULT_ADDR/v1/otus/otus-rw/config
```

- проверим запись

```
curl --request POST --data '{"bar": "baz"}'   --header "X-Vault-
Token:s.pPjvLHcbKsNoWo7zAAuhMoVK" $VAULT_ADDR/v1/otus/otus-ro/config
curl --request POST --data '{"bar": "baz"}'   --header "X-Vault-
Token:s.pPjvLHcbKsNoWo7zAAuhMoVK" $VAULT_ADDR/v1/otus/otus-rw/config
curl --request POST --data '{"bar": "baz"}'   --header "X-Vault-
Token:s.pPjvLHcbKsNoWo7zAAuhMoVK" $VAULT_ADDR/v1/otus/otus-rw/config1
```

Разберемся с ошибками при записи

- Почему мы смогли записать otus-rw/config1 но не смогли otus-rw/config
- Измените политику так, чтобы можно было менять otus-rw/config
- Ответы на вопросы добавить в README.md

Use case использования авторизации через кубер

- Авторизуемся через vault-agent и получим клиентский токен
- Через consul-template достанем секрет и положим его в nginx
- Итог - nginx получил секрет из волта, не зная ничего про волт

Заберем репозиторий с примерами

```
git clone https://github.com/hashicorp/vault-guides.git
cd vault-guides/identity/vault-agent-k8s-demo
```

- В каталоге configs-k8s скорректируйте конфиги с учетом ранее созданых ролей и секретов
- Проверьте и скорректируйте конфиг example-k8s-spec.yml
- Скорректированные конфиги приложить к ДЗ

Запускаем пример

```
# Create a ConfigMap, example-vault-agent-config
$ kubectl create configmap example-vault-agent-config --from-file=./configs-k8s/

# View the created ConfigMap
$ kubectl get configmap example-vault-agent-config -o yaml

# Finally, create vault-agent-example Pod
$ kubectl apply -f example-k8s-spec.yml --record
```

Проверка

- законнектится к поду nginx и вытащить оттуда index.html
- index.html приложить к ДЗ

создадим CA на базе vault

- Включим pki секретс

```
 kubectl exec -it vault-0 -- vault secrets enable pki
 kubectl exec -it vault-0 -- vault secrets tune -max-lease-ttl=87600h pki
 kubectl exec -it vault-0 -- vault write -field=certificate pki/root/generate/internal
\
 common_name="exmaple.ru"  ttl=87600h > CA_cert.crt
```

пропишем урлы для ca и отозванных сертификатов

```
kubectl exec -it vault-0 -- vault write pki/config/urls  \
issuing_certificates="http://vault:8200/v1/pki/ca"       \
crl_distribution_points="http://vault:8200/v1/pki/crl"
```

создадим промежуточный сертификат

```
kubectl exec -it vault-0 -- vault secrets enable --path=pki_int pki
kubectl exec -it vault-0 -- vault secrets tune -max-lease-ttl=87600h pki_int
kubectl exec -it vault-0 -- vault write -format=json
pki_int/intermediate/generate/internal  \
common_name="example.ru Intermediate Authority"         | jq -r '.data.csr' >
pki_intermediate.csr
```

пропишем промежуточный сертификат в vault

```
kubectl cp pki_intermediate.csr vault-0:./
kubectl exec -it vault-0 -- vault write -format=json pki/root/sign-intermediate \
csr=@pki_intermediate.csr   \
format=pem_bundle ttl="43800h" |  jq -r '.data.certificate' > intermediate.cert.pem
kubectl cp intermediate.cert.pem vault-0:./
kubectl exec -it vault-0 -- vault write pki_int/intermediate/set-signed
certificate=@intermediate.cert.pem
```

Создадим и отзовем новые сертификаты

- Создадим роль для выдачи с ертификатов

```
kubectl exec -it vault-0 -- vault write pki_int/roles/example-dot-ru \
allowed_domains="example.ru" allow_subdomains=true   max_ttl="720h"
```

- Создадим и отзовем сертификат

```
kubectl exec -it vault-0 -- vault write pki_int/issue/devlab-dot-ru
common_name="gitlab.devlab.ru" ttl="24h"
kubectl exec -it vault-0 -- vault write pki_int/revoke
serial_number="71:a8:4f:4c:bd:74:c6:d8:ea:27:64:cb:53:ef:80:1a:6b:c8:be:e3"
```

- выдачу при создании сертификата добавить в README.md

🐍 Задание со 🌟 (1)

- реализовать доступ к vault через https
- в README.md описать последовательность действий
- предоставить примеры работы курлом

🐍 Задание со 🌟 (2)

- Настроить autounseal
- провайдер для autoanseal на ваш выбор
- описать все в README.md

🐍 Задание со 🌟 (3)

- Настроить lease временных секретов для доступа к БД
- Описать процесс в README.md