Vault installation to Amazon Elastic Kubernetes Service via Helm

1. Prerequisites
2. Start cluster
3. Install the MySQL Helm chart
4. Install the Vault Helm chart
5. Initialize and unseal one Vault pod
6. Join the other Vaults to the Vault cluster
7. Create a Vault database role
8. Configure Kubernetes authentication
9. Launch a web application
10. Clean up

1. Prerequisites

   choco install awscli
   choco install kubernetes-cli
   choco install kubernetes-helm
   aws configure
   This command prompts you to enter an AWS access key ID, AWS secret access key, and default region name.
   aws ec2 create-key-pair --key-name learn-vault

Create EKS Cluster with EKSCTL

eksctl create cluster --name learn-vault --nodes 3 --with-oidc --ssh-access --ssh-public-key learn-vault –managed or use terraform scripts.

2. Start cluster

Use terraform scripts to create EKS cluster.

Enable volume support with the EBS CSI driver add-on.

eksctl create iamserviceaccount --name ebs-csi-controller-sa --namespace kube-system --cluster learn-vault --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy --approve --role-only ` --role-name AmazonEKS_EBS_CSI_DriverRole

eksctl create addon --name aws-ebs-csi-driver --cluster learn-vault ` --service-account-role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output text):role/AmazonEKS_EBS_CSI_DriverRole

3 Install the MySQL Helm chart

Add the Bitnami Helm repository.

helm repo add bitnami https://charts.bitnami.com/bitnami

Install the latest version of the MySQL Helm chart.

helm install mysql bitnami/mysql

4 Install the Vault Helm chart

Add the HashiCorp Helm repository.

helm repo add hashicorp https://helm.releases.hashicorp.com

Initialize and unseal (way to get root key required to read decryption key the data) one Vault pod

Initialize Vault with one key share and one key threshold.

kubectl exec vault-0 -- vault operator init -key-shares=1 -key-threshold=1 ` -format=json > cluster-keys.json

VAULT_UNSEAL_KEY=$(cat cluster-keys.json | jq -r ".unseal_keys_b64[]")

5. Unseal Vault running on the vault-0 pod.

kubectl exec vault-0 -- vault operator unseal $VAULT_UNSEAL_KEY

Retrieve the status of Vault on the vault-0 pod.

kubectl exec vault-0 -- vault status

6. Join the other Vaults to the Vault cluster

Display the root token found in cluster-keys.json.

cat cluster-keys.json | jq -r ".root_token"

CLUSTERROOTTOKEN=(cat cluster-keys.json | jq -r ".root_token")

Login with the root token on the vault-0 pod.

kubectl exec vault-0 -- vault login $CLUSTER_ROOT_TOKEN

List all the nodes within the Vault cluster for the vault-0 pod.

kubectl exec vault-0 -- vault operator raft list-peers

Join the Vault server on vault-1 to the Vault cluster.

kubectl exec vault-1 -- vault operator raft join http://vault-0.vault-internal:8200

7 Unseal the Vault server on vault-1 with the unseal key.

kubectl exec vault-1 -- vault operator unseal $VAULT_UNSEAL_KEY

Join the Vault server on vault-2 to the Vault cluster.

kubectl exec vault-2 -- vault operator raft join http://vault-0.vault-internal:8200

Unseal the Vault server on vault-2 with the unseal key.

kubectl exec vault-2 -- vault operator unseal $VAULT_UNSEAL_KEY

List all the nodes within the Vault cluster for the vault-0 pod

kubectl exec vault-0 -- vault operator raft list-peers

Get all the pods within the default namespace

kubectl get pods

##Create a Vault database role

Enable database secrets at the path database.

kubectl exec vault-0 -- vault secrets enable database

Configure the database secrets engine with the connection credentials for the MySQL database.

kubectl exec vault-0 -- vault write database/config/mysql plugin_name=mysql-database-plugin connection_url="{{username}}:{{password}}@tcp(mysql.default.svc.cluster.local:3306)/" allowed_roles="readonly" username="root" ` password="$ROOT_PASSWORD"

Create a database secrets engine role named readonly.

kubectl exec vault-0 -- vault write database/roles/readonly db_name=mysql creation_statements="CREATE USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';GRANT SELECT ON . TO '{{name}}'@'%';" default_ttl="1h" max_ttl="24h"

Read credentials from the readonly database role.

kubectl exec vault-0 -- vault read database/creds/readonly

8. Configure Kubernetes authentication

Vault provides a Kubernetes authentication method that enables clients to authenticate with a Kubernetes Service Account Token.

Start an interactive shell session on the vault-0 pod.

kubectl exec --stdin=true --tty=true vault-0 -- /bin/sh

Enable the Kubernetes authentication method.

vault auth enable kubernetes

Configure the Kubernetes authentication method to use the location of the Kubernetes API.

vault write auth/kubernetes/config kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"

Write out the policy named devwebapp that enables the read capability for secrets at path database/creds/readonly

vault policy write devwebapp - <<EOF path "database/creds/readonly" { capabilities = ["read"] } EOF

Create a Kubernetes authentication role named devweb-app.

vault write auth/kubernetes/role/devweb-app bound_service_account_names=internal-app bound_service_account_namespaces=default policies=devwebapp ttl=24h

exit out of vault pod

exit

9. Launch a web application

Create the internal-app service account.

kubectl apply --filename internal-app.yaml

Create the devwebapp pod.

kubectl apply --filename devwebapp.yaml

Display the secrets written to the file /vault/secrets/database-connect.sh on the devwebapp pod.

kubectl exec --stdin=true --tty=true devwebapp --container devwebapp ` -- cat /vault/secrets/database-connect.sh

official documents

https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-amazon-eks