**End-to-End Project on**

**Complete Secret Management**

**Vault Integration with EKS and MySQL**

This project demonstrates how to set up a complete infrastructure with AWS EKS, HashiCorp Vault for secrets management, MySQL database, and a two-tier application that securely accesses database credentials through Vault.

**Project Roadmap:**

**Step 1: Set Up Bastion Host**

**Step 2: Create EKS Cluster**

**Step 3: Set Up Storage Class**

**Step 4: Install MySQL**

**Step 5: Install and Configure Vault**

**Step 6: Configure Vault Secrets and Authentication**

**Step 7: Test Vault Integration with Sample Application Step**

**Step 8: Configure Dynamic Database Credentials**

**Step 9: Deploy Two-Tier Application**

**Step 10: Clean Up (Optional)**

**In Details**

**Step 1: Set Up Bastion Host**

*# Create a new EC2 instance (bastion host) with Ubuntu AMI*

*# SSH into the instance and run:*

*# Update system and install prerequisites*

sudo apt update && sudo apt -y full-upgrade

sudo apt install unzip vim -y

*# Install AWS CLI*

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

rm -fr awscliv2.zip aws

*# Configure AWS CLI*

aws configure

*# Enter your AWS Access Key ID, Secret Access Key, default region (us-east-1), and output format (json)*

*# Install kubectl*

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

rm -f kubectl

*# Install eksctl*

ARCH=amd64

PLATFORM=$(uname -s)\_$ARCH

curl -sLO "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$PLATFORM.tar.gz"

curl -sL "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_checksums.txt" | grep $PLATFORM | sha256sum --check

tar -xzf eksctl\_$PLATFORM.tar.gz -C /tmp && rm eksctl\_$PLATFORM.tar.gz

sudo mv /tmp/eksctl /usr/local/bin

*# Install Helm*

curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null

sudo apt-get install apt-transport-https --yes

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list

sudo apt-get update

sudo apt-get install helm

*# Set up aliases*

echo "alias k='kubectl'" >> ~/.bashrc

echo "alias kgp='kubectl get pods'" >> ~/.bashrc

echo "alias kgn='kubectl get nodes'" >> ~/.bashrc

echo "alias kgs='kubectl get service'" >> ~/.bashrc

source ~/.bashrc

**Step 2: Create EKS Cluster**

*# Create cluster configuration file*

cat > cluster.yaml <<EOF

apiVersion: eksctl.io/v1alpha5

kind: ClusterConfig

metadata:

name: cluster-vault

region: us-east-1

nodeGroups:

- name: ng-1

instanceType: t2.medium

desiredCapacity: 2

volumeSize: 30

iam:

withOIDC: true

addons:

- name: vpc-cni

attachPolicyARNs:

- arn:aws:iam::aws:policy/AmazonEKS\_CNI\_Policy

- name: coredns

version: latest

- name: kube-proxy

version: latest

- name: aws-ebs-csi-driver

wellKnownPolicies:

ebsCSIController: true

EOF

*# Create the cluster*

eksctl create cluster -f cluster.yaml

*# Verify cluster is ready*

k get nodes

**Step 3: Set Up Storage Class**

*# Create storage class configuration*

cat > storage.yaml <<EOF

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ebs-sc

annotations:

storageclass.kubernetes.io/is-default-class: "true"

provisioner: ebs.csi.aws.com

parameters:

type: gp2

reclaimPolicy: Retain

volumeBindingMode: WaitForFirstConsumer

EOF

*# Apply storage class*

kubectl apply -f storage.yaml

**Step 4: Install MySQL**

**kubectl apply -f storage.yaml**

*(install Vault from* [***https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-amazon-eks***](https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-amazon-eks) ***)***

**helm repo add bitnami** [**https://charts.bitnami.com/bitnami**](https://charts.bitnami.com/bitnami)

**helm repo update**

**vim mysql-data.yaml**

primary:

persistence:

enabled: true

storageClass: "ebs-sc"

accessModes:

- ReadWriteOnce

size: 20Gi

**helm install mysql bitnami/mysql -f mysql-data.yaml**

single node mysql created

**kgp**

Mysql\_ROOT\_Password=$(**inside only**) **copy** then paste at cli

**Kgp**

**K get pv**

**K get pvc** … see bound

**kubectl exec -it mysql-0 -- bash**

**env (** copy password**)**

**mysql -u root -p (**now paste that password)

**show databases;**

**create database two\_tier;**

**use two\_tier;**

**CREATE TABLE users (id int NOT NULL AUTO\_INCREMENT, email varchar(100), password varchar(100), primary key(id));**

**CREATE TABLE history (id int NOT NULL AUTO\_INCREMENT, number varchar(100), response varchar(100), type varchar(30), time timestamp default current\_timestamp, primary key(id));**

**Show tables;**

**INSERT INTO users (email, password) values ("hello@example.com", "hello123");**

**SELECT \* FROM users;**

**exit**

**exit**

**kgs**

**Step 5: Install the Vault Helm chart**

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

**helm repo update**

**helm search repo vault --versions**

go to <https://artifacthub.io/packages/helm/hashicorp/vault> > default vault

copy then paste as: cat > helm-vault-raft-values.yml <<EOF

or,

**vim vault-values.yaml**

server:

affinity: ""

ha:

enabled: true

raft:

enabled: true

setNodeId: true

config: |

cluster\_name = "vault-integrated-storage"

storage "raft" {

path = "/vault/data/"

}

listener "tcp" {

address = "[::]:8200"

cluster\_address = "[::]:8201"

tls\_disable = "true"

}

service\_registration "kubernetes" {}

**---**

**helm install vault hashicorp/vault -f vault-values.yaml**

**kgp** … see <https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-amazon-eks>

**kubectl exec -it vault-0 -- sh**

**vault status** … see false

**vault operator init** …copy all the unseal keys + root token, save at notepad

**vault status** … see it is initialized but locked… need to unseal it

**vault operator unseal** .. pass keys… see sealed **false**

**vault login** … paste saved token

**vault operator raft list-peers** … see voter true

**exit**

**k exec -it vault-1 -- sh**

**vault operator raft join** [**http://vault-0.vault-internal:8200**](http://vault-0.vault-internal:8200)…see joined **true**

**exit**

**k exec -it vault-2 -- sh**

**vault operator raft join** [**http://vault-0.vault-internal:8200**](http://vault-0.vault-internal:8200)

**exit**

**kgp …**see vault-1 not ready

**k exec -it vault-1 -- sh**

**vault operator raft list-peers** …see vault-0 is leader **true**

**vault operator unseal** ….run all 3 tokens

**vault status …**see sealed **false**

**exit**

**k exec -it vault-2 -- sh**

**vault status …**see sealed true

**vault operator unseal** ….run all 3 tokens

**vault status …**see sealed **false**

**exit**

**kgp ..** see all are ready

**k exec -it vault-0 -- sh**

**vault operator raft list-peers** …see vault-0 is leader **true** others are followers

**Step 6: Injecting secrets into Kubernetes pods via Vault Agent containers**

<https://developer.hashicorp.com/vault/tutorials/kubernetes/kubernetes-sidecar>

**vault secrets enable -path=internal kv-v2** … see Success, enabled kv-v2

**vault kv put internal/database/config username="root" password="<**use saved mysql passwrd**>"**

**vault secrets list** …see manually injected secrets in vault named kv

Verify that the secret is defined at the path internal/database/config.

**vault kv get internal/database/config** … see password and user name

**Step 7:** Configure Kubernetes authentication

**vault auth enable kubernetes**

**vault write auth/kubernetes/config \  
kubernetes\_host="https://$KUBERNETES\_PORT\_443\_TCP\_ADDR:443"**

*Write out the policy named internal-app that enables the read capability for secrets at path internal/data/database/config.*

**vault policy write internal-app - <<EOF**

**path "internal/data/database/config" {**

**capabilities = ["read"]**

**}**

**EOF**

*Create a Kubernetes authentication role named internal-app.*

**vault write auth/kubernetes/role/internal-app \**

**bound\_service\_account\_names=internal-app \**

**bound\_service\_account\_namespaces=default \**

**policies=internal-app \**

**ttl=24h**

**exit**

**Step 8:** Define a Kubernetes **service** account

**k get sa**

**kubectl create sa internal-app**

Launch an application

**vim deploy.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: orgchart**

**labels:**

**app: orgchart**

**spec:**

**selector:**

**matchLabels:**

**app: orgchart**

**replicas: 1**

**template:**

**metadata:**

**annotations: {}**

**labels:**

**app: orgchart**

**spec:**

**serviceAccountName: internal-app**

**containers:**

**- name: orgchart**

**image: jweissig/app:0.0.1**

**k apply -f deploy.yaml** …orgchart is running

kgp

**k exec -it** <orgchart pod**> -- sh**

**cd /vault** … see **no** such file or directory… we need to configure it with adding annotations

**exit**

**kgp …**see only **one** pod running instead of 2 at orgchrt

**k delete deploy orgchart**

**vim deploy.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: orgchart**

**labels:**

**app: orgchart**

**spec:**

**selector:**

**matchLabels:**

**app: orgchart**

**replicas: 1**

**template:**

**metadata:**

**annotations:**

**vault.hashicorp.com/agent-inject: 'true'**

**vault.hashicorp.com/agent-inject-status: 'update'**

**vault.hashicorp.com/role: 'internal-app'**

**vault.hashicorp.com/agent-inject-secret-database-config.txt: 'internal/data/database/config'**

**labels:**

**app: orgchart**

**spec:**

**serviceAccountName: internal-app**

**containers:**

**- name: orgchart**

**image: jweissig/app:0.0.1**

**k apply -f deploy.yaml**

**kgp** …see 2 orgchart is running.. one access credentials, another has credentials…

**k exec -it orgchart pod -- sh**

**cd /vault**

**ls …**see secrets

cd **secrets**

**ls**

**cat database-config.txt** …see password, username

**exit**

**k delete deploy orgchart**

**Step 9:** Apply a template to the injected secrets

**vim deploy.yaml …** vault injector template [**https://developer.hashicorp.com/vault/docs/platform/k8s/injector**](https://developer.hashicorp.com/vault/docs/platform/k8s/injector)

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: orgchart**

**labels:**

**app: orgchart**

**spec:**

**selector:**

**matchLabels:**

**app: orgchart**

**replicas: 1**

**template:**

**metadata:**

**annotations:**

**vault.hashicorp.com/agent-inject: 'true'**

**vault.hashicorp.com/agent-inject-status: 'update'**

**vault.hashicorp.com/role: 'internal-app'**

**vault.hashicorp.com/agent-inject-secret-database-config.txt: 'internal/data/database/config'**

**vault.hashicorp.com/agent-inject-template-database-config.txt: |**

**{{- with secret "internal/data/database/config" -}}**

**{{ range $k, $v := .Data }}**

**{{ $k }}: {{ $v }}**

**{{ end }}**

**{{- end -}}**

**labels:**

**app: orgchart**

**spec:**

**serviceAccountName: internal-app**

**containers:**

**- name: orgchart**

**image: jweissig/app:0.0.1**

**k apply -f deploy.yaml**

**kgp**

**k exec -it <orgchart pod> --sh**

**cd /vault/secrets**

**cat database-config.txt …**see password and user name in **good shape**

(its **completed**: how to **inject data into Vault orgchart)**

**Exit**

**Step 10:** Create a Vault database role

**kgp**

**k exec -it vault-0 -- sh**

**vault secrets enable database**

**vault secrets list** …. See database in list

**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="<use the saved password>"**

Create a database secrets engine role named **readonly**.

**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"**

**vault read database/creds/readonly** … copy then **save** username + password

**exit**

**kgp**

**k exec -it mysql-0 -- bash**

**mysql -u** <write username saved earlier> **-p**

**use two\_tier;**

**INSERT INTO users (email, password) VALUES (‘Azizul@gmail.com’, ‘Hello123’);**

it throws **errors**, bcz **read-only** creds given

**exit**

**exit**

**k exec -it vault-0 -- sh**

**vault write database/roles/readonly \**

**db\_name=mysql \**

**creation\_statements="CREATE USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';GRANT SELECT, ALTER, INSERT, UPDATE ON \*.\* TO '{{name}}'@'%';" \**

**default\_ttl="1h" \**

**max\_ttl="24h"**

**vault read database/creds/readonly …**see username + password are changed everytime happens! Copy+save it

**exit**

**k exec -it mysql-0 -- bash**

**mysql -u <new user name> -p**

**use two\_tier;**

**INSERT INTO users (email, password) VALUES ('Azizul@gmail.com', 'Hello123');**

**select\* from users;**

**exit**

**exit**

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

**k exec -it vault-0 -- sh**

**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**

**Step 11: Launch a web application**

**cp deploy.yaml dynamic.yaml**

**vim dynamic.yaml**

**---**

apiVersion: apps/v1

kind: Deployment

metadata:

name: dynamic

labels:

app: dynamic

spec:

selector:

matchLabels:

app: dynamic

replicas: 1

template:

metadata:

annotations:

vault.hashicorp.com/agent-inject: 'true'

vault.hashicorp.com/agent-cache-enable: "true"

vault.hashicorp.com/agent-inject-status: 'update'

vault.hashicorp.com/role: "devweb-app"

vault.hashicorp.com/agent-inject-secret-database-connect.txt: "database/creds/readonly"

vault.hashicorp.com/agent-inject-template-database-connect.txt: |

{{- with secret "database/creds/readonly" -}}

{{ range $k, $v := .Data }}

{{ $k }}: {{ $v }}

{{ end }}

{{- end -}}

labels:

app: dynamic

spec:

serviceAccountName: internal-app

containers:

- name: dynamic

image: jweissig/app:0.0.1

**k apply -f dynamic.yaml**

**kgp**

**k exec -it <dynamic pod name> -- sh**

**cd /vault/secrets**

**ls**

**cat database-connect.txt** … see paswrd, username

\*\* now, duplicate the screen

**kubectl exec -it mysql-0 -- bash**

**>>mysql -u** <use copied from another screen> **-p**

\*\* delete this screen then go back earlier one..

**Exit**

**Step 12: Continuous Integrate Full Stack / Two-tier Application with HashiCorp Vault**

**git clone** <https://github.com/azizulmaqsud/two-tier-app.git>

**cd two-tier**

**ls**

**cd backend**

**git branch -a**

**git checkout two-tier-v2**

**vim .env**

**MYSQL\_HOST="mysql.default.svc.cluster.local"**

**MYSQL\_DB="two\_tier"**

MYSQL\_HOST=‘mysql’

MYSQL\_DB=‘two\_tier’

**vim Dockerfile**

FROM python:3.8

WORKDIR /app

RUN apt update -y

RUN apt install pkg-config python3-pip pyhton3-dev -y

COPY ./requirements.txt /app/requirements.txt

RUN pip install -r requirements.txt

ENV FLASK\_APP=app.py

EXPOSE 5000

RUN flask run --host=0.0.0.0

**vim entry.sh**

#!/bin/bash

export MYSQL\_USER=$(cat /vault/secrets/backend-user)

export MYSQL\_PASSWORD=$(cat /vault/secrets/backend-pass)

flask run --host=0.0.0.0

**vim Dockerfile**

**FROM python:3.8**

**WORKDIR /app**

**# Update package lists**

**RUN apt update -y**

**# Install necessary packages**

**RUN apt install -y pkg-config python3-pip python3-dev**

**# Copy requirements file and install Python dependencies**

**COPY ./requirements.txt /app/requirements.txt**

**RUN pip install -r requirements.txt**

**# Set environment variables**

**ENV FLASK\_APP=app.py**

**# Add the application files**

**ADD . /app**

**# Expose the port the app will run on**

**EXPOSE 5000**

**# Set the entry point for the container**

**ENTRYPOINT ["./entry.sh"]**

**chmod +x entry.sh**

**sudo apt install docker.io**

**sudo docker build -t azizul2go/backend-hashicorp:latest .**

**sudo docker login**

**sudo docker push azizul2go/backend-hashicorp:latest**

**cd ../../**

**cp dynamic.yaml backend.yaml**

**vim backend.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: backend

labels:

app: backend

spec:

selector:

matchLabels:

app: backend

replicas: 1

template:

metadata:

annotations:

vault.hashicorp.com/agent-inject: 'true'

vault.hashicorp.com/agent-cache-enable: "true"

vault.hashicorp.com/agent-inject-status: 'update'

vault.hashicorp.com/role: "devweb-app"

vault.hashicorp.com/agent-inject-secret-backend-user: "database/creds/readonly"

vault.hashicorp.com/agent-inject-template-backend-user: |

{{- with secret "database/creds/readonly" -}}

{{ .Data.username }}

{{- end -}}

vault.hashicorp.com/agent-inject-secret-backend-pass: "database/creds/readonly"

vault.hashicorp.com/agent-inject-template-backend-pass: |

{{- with secret "database/creds/readonly" -}}

{{ .Data.password }}

{{- end -}}

labels:

app: backend

spec:

serviceAccountName: internal-app

containers:

- name: backend

image: azizul2go/backend-hashicorp

\*\*\* duplicate tab to see cat **entry.sh**

**k apply -f backend.yaml**

**kgp**

**k exec -it <backend pod> -- bash**

**cd /vault/secrets**

**ls** …see backend-pass & user

**cat backend-user**

**cat backend-pass**

**exit**

**kgp**

**k expose pod <backend pod> --type NodePort --port 5000 --name backend-server**

**k get svc …**get backend server NodePort ip

go back ec2 … change both SG inbound all traffic… then at browser ip address:nodeport number

see Hello world from backend

**Step 13: Testing Backend APIs with Postman: Integrating HashiCorp Vault**

**open postman…** post: url from backend browser/summation **body .. raw… json**

{

"num": "3526"

}

**Send > see body .. pretty …**

backend browser/reverser try it

**kgp**

**k exec -it <backend pod> -- bash**

**cd /vault/secrets**

**cat backend-user**

**cat backend-pass …**save both at notepad

**exit**

**k exec -it mysql-0 -- bash**

**mysql -u <username> -p**

**use two\_tier;**

**show tables;**

**select \* from history;** …see history data got from api postman… success! Our vault accessing data from application; at the same time we can access secret credentials from vault. That’s the whole story!

**Thank you**