

DevOps Project BankApp Deployment using ArgoCD and Secrets Stored in
Hashicorp Vault Multibranch Pipeline with Webhook Multicloud (AWS and Azure)



By Ritesh Kumar Singh

Email Address: - riteshkumarsingh9559@gmail.com

LinkedIn: - <https://www.linkedin.com/in/ritesh-kumar-singh-41113128b/>

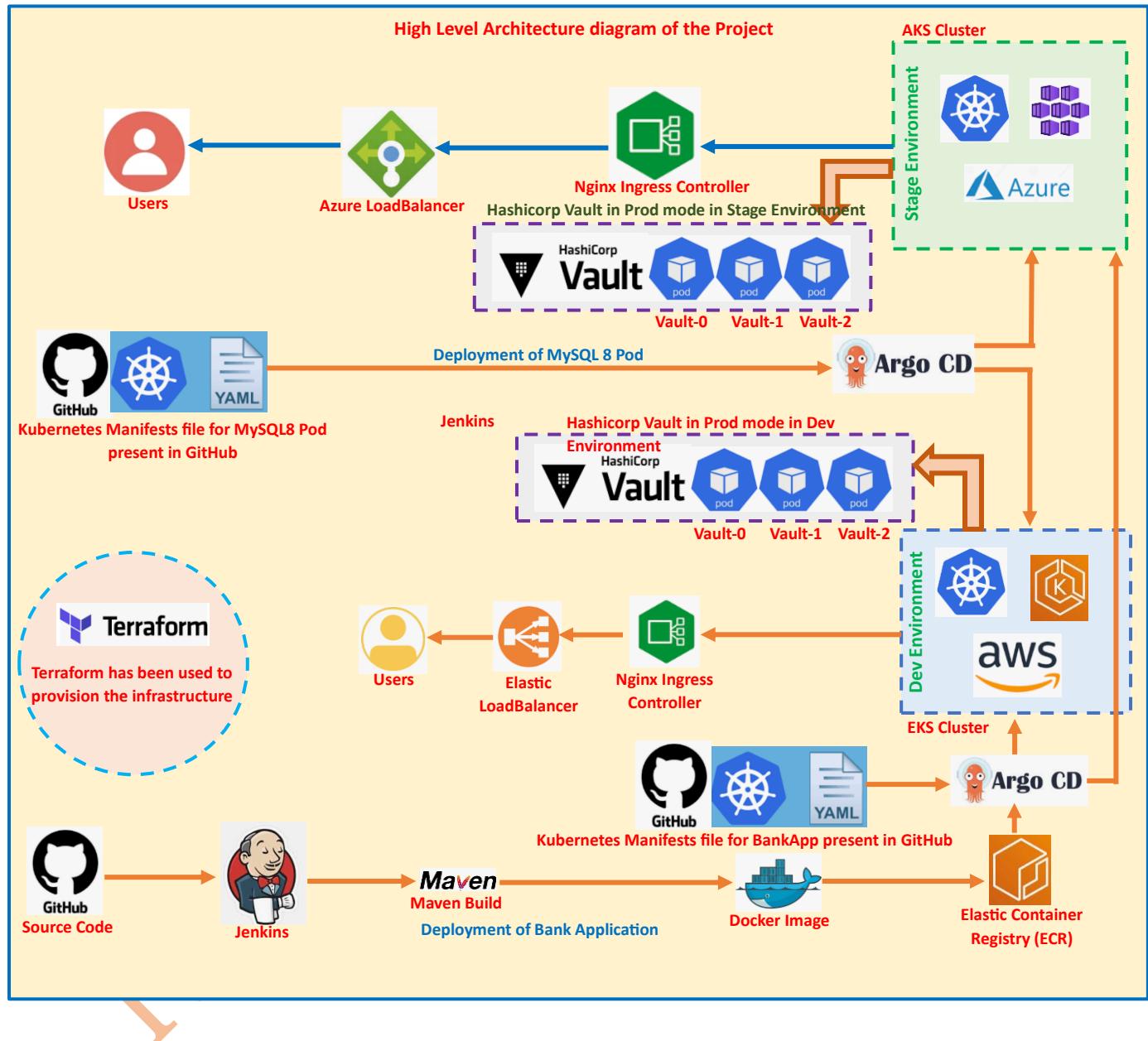
GitHub: - <https://github.com/singhritesh85>



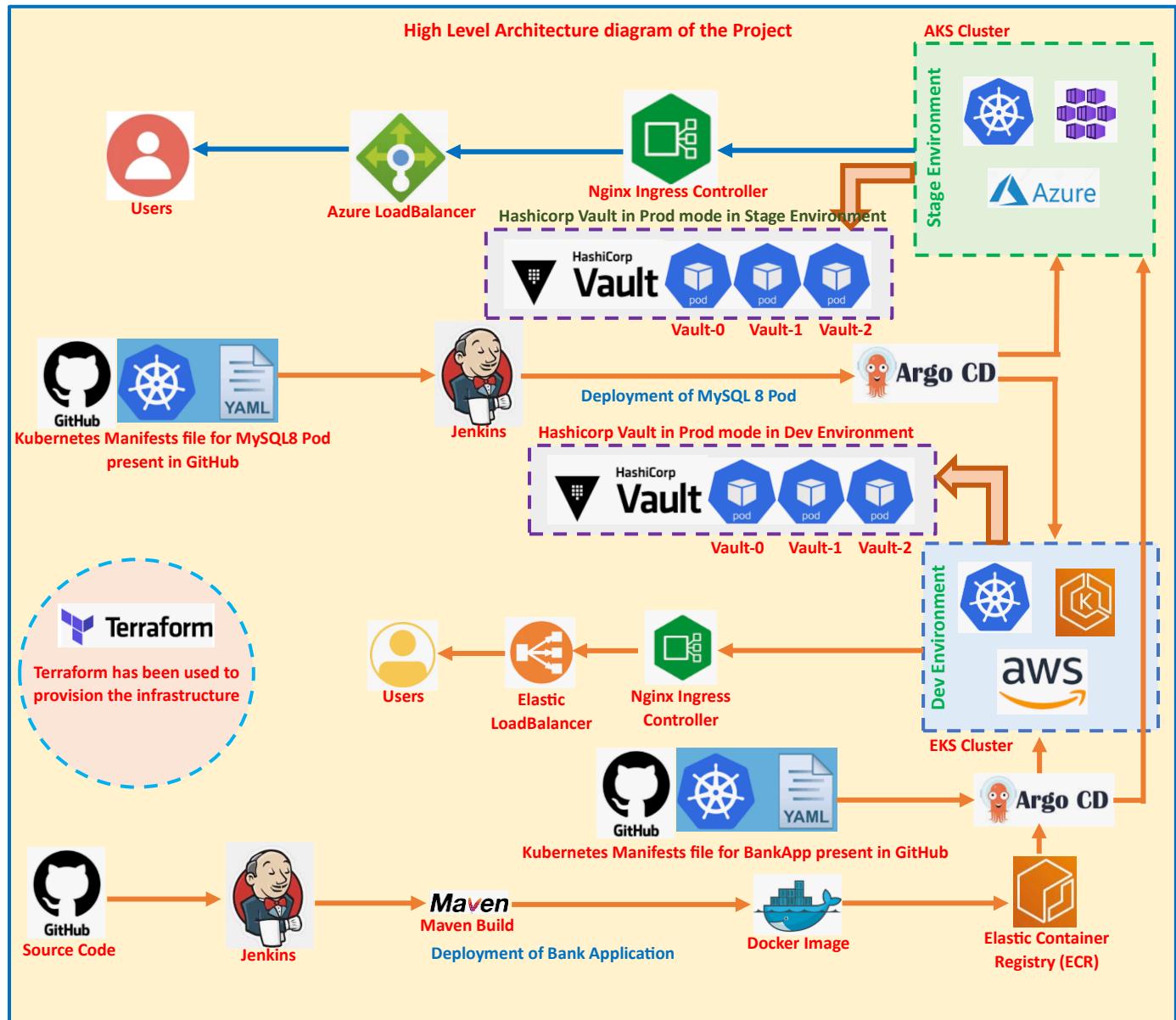
या कुन्देन्दुतुषारहारधवला या शुभ्रवस्त्रावृता
या वीणावरदण्डमण्डितकरा या श्वेतपद्मासना।
या ब्रह्माच्युत शंकरप्रभृतिभिर्देवैः सदा वन्दिता
सा मां पातु सरस्वती भगवती निःशेषजाङ्ग्यापहा ॥

DevOps Project BankApp Deployment using ArgoCD and Secrets Stored in Hashicorp Vault Multibranch Pipeline with Webhook Multicloud (AWS and Azure)

Module-1: Manual Approach



Module-2: Automation Approach



This DevOps project aims to create the Resources in AWS and Azure cloud using the Terraform Script present in the GitHub Repo <https://github.com/singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud.git> at the path **terraform-bankapp-multibranch-multicloud-nonprod** and setup of CICD pipeline as shown in the high-level architecture diagram drawn above. In this project I did not use the SonarQube, Nexus and Trivy Image scan if you are interested in that then you can go through the GitHub Repo <https://github.com/singhritesh85/DevOps-Project-BankApplication-Multibranch-MultiCloud.git>. In this project I used **Hashicorp Vault** in Prod mode (High Availability) to store the MySQL8 Pod credentials. I had used two ECR (Elastic Container Registries) **bankapp-1** which keeps the Docker Image for dev environment (EKS Cluster) and **bankapp-2** which keeps the Docker Image for stage environment (AKS Cluster).

To validate the SSL Certificate generated from AWS certificate manager I used DNS validation and did the entry for Azure DNS Zone record set of type CNAME as shown in the screenshot attached below.

The screenshot shows the AWS Certificate Manager interface. At the top, it displays a certificate's identifier, ARN, and type (Amazon Issued). Below this, under 'Domains (1)', it lists a domain mapping for *.singhritesh85.com, which is successful and uses a CNAME record. A red arrow points from this section to the Azure DNS Zone configuration.

The screenshot shows the Azure DNS Zone 'singhritesh85.com' configuration. It displays a CNAME record for the domain *.singhritesh85.com, pointing to a value that starts with 'f'. A red arrow points from this record to the AWS certificate status.

Then I wait for around 20-25 seconds and after that SSL Certificate was issued as shown in the first screenshot attached above.

I installed terraform on Alma Linux2 Azure VM and State file was kept in the S3 Bucket and state lock had been achieved using the AWS DynamoDB. Before running the terraform script I ran the shell script present in the directory **terraform-bankapp-multibranch-multicloud-nonprod** as shown in the screenshot attached below. This shell script will install the aws cli, kubectl and helm.

```
[root@... terraform-bankapp-multibranch-multicloud-nonprod]# ./initial-setup.sh
```

Then I logged-out and login again ran the command aws configure as shown in the screenshot attached below. As it is an important step to Authenticate and Authorize the user before creating resources using terraform in the AWS Account.

```
[root@Terraform-Server terraform-bankapp-multibranch-multicloud-nonprod]# cd main/
[root@[REDACTED] main]# aws configure
AWS Access Key ID [*****]: [REDACTED]
AWS Secret Access Key [*****]: [REDACTED]
Default region name [REDACTED]: [REDACTED]
Default output format [REDACTED]: [REDACTED]
[root@[REDACTED] main]#
```

Then I installed Azure CLI and authenticated and authorize the user as shown in the screenshot attached below.

```
[root@[REDACTED] main]# yum install -y https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm
[root@[REDACTED] main]# yum install azure-cli -y
[root@[REDACTED] main]# az login
To sign in, use a web browser to open the page [REDACTED] and enter the code [REDACTED] to authenticate.
```

Then you can run the below commands

terraform init -----> initializes a working directory containing configuration files and installs plugins for required providers.

terraform validate -----> verify that terraform configuration file is correct or not

terraform plan -----> Check which resources are going to be created.

Then you can run the command **terraform apply -auto-approve** -----> Finally, Create the resources.

```
module.eks_cluster.aws_eks_addon.vpc_cni: Still creating... [06m20s elapsed]
module.eks_cluster.aws_eks_addon.vpc_cni: Still creating... [06m30s elapsed]
module.eks_cluster.aws_eks_addon.vpc_cni: Creation complete after 6m39s [id=eks-demo-cluster-dev:vpc-cni]

Apply complete! Resources: 83 added, 0 changed, 0 destroyed.

Outputs:

acr_ec2_private_ip_alb_dns = {
  "EC2_Instance_Jenkins_Master_Server_Private_IP_Address" = "10.[REDACTED]"
  "EC2_Instance_Jenkins_Slave_Server_Private_IP_Address" = "10.[REDACTED]"
  "Jenkins_ALB_DNS_Name" = "jenkins-ms-[REDACTED].us-east-2.elb.amazonaws.com"
  "registry_id" = [
    "02[REDACTED]6",
    "02[REDACTED]6",
  ]
  "repository_url" = [
    "02[REDACTED]6.dkr.ecr.us-east-2.amazonaws.com/bankapp-1",
    "02[REDACTED]6.dkr.ecr.us-east-2.amazonaws.com/bankapp-2",
  ]
}
```

After creation of the resources a kubeconfig file was generated, I did below change in the kubeconfig file as shown in the screenshot attached below.

```

- cluster:
  certificate-authority-data: L
  server: https://F 6.sk1.us-east-2.eks.amazonaws.com
  name: eks-demo-cluster-dev
contexts:
- context:
  cluster: aks-cluster
  user: clusterUser_aks_rg_aks-cluster
  name: aks-cluster
- context:
  cluster: eks-demo-cluster-dev
  user: arn:aws:eks:us-east-2:02 6:cluster/eks-demo-cluster-dev
  name: eks-demo-cluster-dev
current-context: aks-cluster
kind: Config
preferences: {}
users:
- name: arn:aws:eks:us-east-2:02 6:cluster/eks-demo-cluster-dev
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      args:
        - --region
        - us-east-2
        - eks
        - get-token
        - --cluster-name
        - eks-demo-cluster-dev
        - --output
        - json

```

It is my suggestion before any change in the kubeconfig file take a backup of the original kubeconfig file. After creation of EKS and AKS Cluster the node in the Kubernetes Cluster can be seen as shown in the screenshot attached below.

```

[root@  main]# kubectl get nodes --context=aks-cluster
NAME           STATUS   ROLES   AGE   VERSION
aks-agentpool-  Ready    <none>  121m  v1.30.0
aks-userpool-  Ready    <none>  116m  v1.30.0
aks-userpool-  Ready    <none>  47s   v1.30.0
aks-userpool-  Ready    <none>  30s   v1.30.0
[root@  main]# kubectl get nodes --context=eks-demo-cluster-dev
NAME           STATUS   ROLES   AGE   VERSION
ip-10- 167.us-east-2.compute.internal  Ready    <none>  115m  v1.30.4-eks-
ip-10- 65.us-east-2.compute.internal  Ready    <none>  115m  v1.30.4-eks-
ip-10- 11.us-east-2.compute.internal  Ready    <none>  3m44s v1.30.4-eks-
ip-10- 122.us-east-2.compute.internal  Ready    <none>  3m43s v1.30.4-eks-

```

I had created the URL for Jenkins by creating the record set of Type CNAME in Azure DNS Zone using the Jenkins LoadBalancer DNS Name as shown in the screenshot attached below.

The screenshot shows the AWS Route 53 console under the 'singhritesh85.com' DNS zone. A modal window titled 'Add record set' is open, prompting for a 'Name' (set to 'jenkins-ms') and a 'Type' (set to 'CNAME – Link your subdomain to another record'). The 'Alias' field contains 'jenkins-ms-[REDACTED].us-east-2.elb.amazonaws.com'. The 'Add' button at the bottom of the modal is highlighted with a yellow box.

Finally, I logged-in to the Jenkins and created two credentials for Jenkins Slave and GitHub Credentials.

The screenshot shows the Jenkins 'Manage Jenkins' configuration page. The 'Manage Jenkins' tab is selected. In the 'System Configuration' section, the 'Credentials' link under the 'Tools' category is highlighted with a yellow box. The URL in the browser bar is 'jenkins-ms.singhritesh85.com/manage/'.

jenkins-ms.singhrithesh85.com/manage/credentials/store/system/domain/_/

Jenkins

Ritesh Kumar Singh log out

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
This credential domain is empty. How about adding some credentials?			

Icon: S M L

+ Add Credentials

jenkins-ms.singhrithesh85.com/manage/credentials/store/system/domain/_/newCredentials

REST API Jenkins 2.504.1

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: jenkins

Treat username as secret

Password:

ID: jenkins-cred

Description: jenkins-cred

Create

The screenshot shows the Jenkins Global credentials (unrestricted) page. A single credential named "jenkins-cred" is listed. The "Kind" is "Username with password". The "Description" is "jenkins-cred". A blue button labeled "+ Add Credentials" is visible at the top right.

ID	Name	Kind	Description
jenkins-cred	jenkins/******** (jenkins-cred)	Username with password	jenkins-cred

Icon: S M L

The screenshot shows the Jenkins "newCredentials" form for creating a new credential. The "Kind" is set to "Username with password". The "Scope" is "Global (Jenkins, nodes, items, all child items, etc)". The "Username" field is filled with a yellow placeholder. The "Password" field is filled with a yellow placeholder. The "ID" is "github-cred" and the "Description" is "github-cred". A blue "Create" button is at the bottom.

Then I had added Jenkins Slave Node as shown in the screenshot attached below. Go to Jenkins and then Manage Jenkins > Nodes > New Node.

jenkins-ms.singhrithesh85.com/manage/computer/new

Jenkins

Dashboard > Manage Jenkins > Nodes > New node

New node

Node name

Type Permanent Agent
Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

Create

jenkins-ms.singhrithesh85.com/manage/computer/creatitem

Jenkins

Dashboard > Manage Jenkins > Nodes >

Name ?

Description ?

Plain text [Preview](#)

Number of executors ?

Remote root directory ?

Save

The image shows two screenshots of the Jenkins web interface. The top screenshot is titled 'Manage Jenkins > Nodes > Create New' and shows the configuration for a new node named 'Slave-1'. It includes fields for Labels ('Slave-1'), Usage ('Use this node as much as possible'), Launch method ('Launch agents via SSH'), Host ('10.10.10.10'), Credentials ('jenkins/******** (jenkins-cred)'), and Host Key Verification Strategy ('Non verifying Verification Strategy'). The bottom screenshot is titled 'Manage Jenkins > Nodes > Slave-1' and shows the details for the 'Agent Slave-1' node. It lists 'Status' (Status), 'Delete Agent', 'Configure', 'Build History', 'Load Statistics', and 'Log'. The 'Projects tied to Slave-1' section shows 'None'. There are buttons for 'Edit description', 'Relaunch agent', 'Mark this node temporarily offline', and a help icon.

Installation of Nginx Ingress Controller and Hashicorp Vault Prod in EKS and AKS Cluster

Installation of Nginx Ingress Controller and Hashicorp Vault Prod in EKS

Installation of Nginx Ingress Controller in EKS Cluster

To install Nginx Ingress Controller in EKS Cluster I used helm and below are the commands used.

```
kubectl create ns ingress-nginx
```

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

```
helm repo update
```

```
helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx --set
controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-ssl-
cert"=arn:aws:acm:us-east-2:02XXXXXXXXXX6:certificate/XXXXXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXXX --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-
balancer-connection-idle-timeout"="60" --set
controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-cross-zone-load-
balancing-enabled"="true" --set
controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="elb" --
set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-backend-
protocol"="http" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-
balancer-ssl-ports"="https" --set controller.service.targetPorts.https=http --set-string
controller.config.use-forwarded-headers="true"
```

```
[root@XXXXXXXXXX ~]# kubectl config get-contexts
CURRENT  NAME          CLUSTER      AUTHINFO           NAMESPACE
*       aks-cluster    aks-cluster   clusterUser_aks-rg_aks-cluster
        eks-demo-cluster-dev  eks-demo-cluster-dev  arn:aws:eks:us-east-2:0XXXXXXXXXX6:cluster/eks-demo-cluster-dev
[root@XXXXXXXXXX ~]# kubectl config use-context eks-demo-cluster-dev
Switched to context "eks-demo-cluster-dev".
[root@XXXXXXXXXX ~]# kubectl create ns ingress-nginx
namespace/ingress-nginx created
[root@XXXXXXXXXX ~]# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
"ingress-nginx" has been added to your repositories
[root@XXXXXXXXXX ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
Update Complete.  Happy Helm-ing!
```

```
[root@XXXXXXXXXX ~]# helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-ssl-cert"=arn:aws:acm:us-east-2:0XXXXXXXXXX6:certificate/XXXXXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXXX --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-cross-zone-load-balancing-enabled"="true" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-type"="elb" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-backend-protocol"="http" --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-ssl-ports"="https" --set controller.service.targetPorts.https=http --set-string controller.config.use-forwarded-headers="true"
```

```
[root@XXXXXXXXXX ~]# kubectl get all -n ingress-nginx
NAME                                READY   STATUS    RESTARTS   AGE
pod/ingress-nginx-controller-XXXXXX  1/1     Running   0          2m6s

NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
service/ingress-nginx-controller     LoadBalancer  10.XX.XX.92   aXXXXXXXXXX.us-east-2.elb.amazonaws.com  80:32585/
TCP,443:31666/TCP  2m6s
service/ingress-nginx-controller-admission ClusterIP   10.XX.XX.187  <none>           443/TCP
2m6s

NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller  1/1     1           1          2m6s

NAME          DESIRED  CURRENT  READY   AGE
replicaset.apps/ingress-nginx-controller-XXXXXX  1        1        1        2m6s
```

Installation of Hashicorp Vault Prod in EKS Cluster

I had installed Hashicorp Vault Prod in EKS Cluster using the commands as shown in the screenshot attached below.

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

```
helm repo update
```

```
helm upgrade --install vault hashicorp/vault --namespace vault --create-namespace --set
server.image.tag=1.19.0,server.ha.enabled=true,server.ha.raft.enabled=true,server.ha.replicas=3,server.dev.enabled=false,server.ui.enabled=true,server.ui.serviceType=ClusterIP,server.dataStorage.storageClass=gp2,server.dataStorage.size=2Gi
```

```
[root@XXXXXXXXXX ~]# helm repo add hashicorp https://helm.releases.hashicorp.com
"hashicorp" has been added to your repositories
[root@XXXXXXXXXX ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "hashicorp" chart repository
Update Complete. Happy Helm-ing!
[root@XXXXXXXXXX ~]# helm upgrade --install vault hashicorp/vault --namespace vault --create-namespace --set server.image.tag=1.19.0,server.ha.enabled=true,server.ha.raft.enabled=true,server.ha.replicas=3,server.dev.enabled=false,server.ui.enabled=true,server.ui.serviceType=ClusterIP,server.dataStorage.storageClass=gp2,server.dataStorage.size=2Gi
Release "vault" does not exist. Installing it now.
NAME: vault
LAST DEPLOYED: XXXXXXXXXX 2025
NAMESPACE: vault
STATUS: deployed
REVISION: 1
NOTES:
Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using
Vault with Kubernetes available here:

https://developer.hashicorp.com/vault/docs

Your release is named vault. To learn more about the release, try:

$ helm status vault
$ helm get manifest vault
```

```
[root@XXXXXXXXXX main]# kubectl get pods -n vault
NAME                      READY   STATUS    RESTARTS   AGE
vault-0                  0/1     Running   0          26s
vault-1                  0/1     Running   0          26s
vault-2                  0/1     Running   0          26s
vault-agent-injector-  1/1     Running   0          26s
```

Now logged-in to the vault pods and executed the below commands.

```
=====  
on pod vault-0  
=====
```

```
kubectl exec -it vault-0 -n vault sh
```

```
$ vault operator init
```

```
----- Key1
```

```
----- Key2
```

```
----- Key3
```

```
----- Key4
```

```
----- Key5
```

```
----- Root Token
```

```
$ vault operator unseal -----> First Time
```

```
Unseal Key:
```

```
$ vault operator unseal -----> Second Time
```

```
Unseal Key:
```

```
$ vault operator unseal -----> Third Time
```

```
Unseal Key:
```

```
$ vault login -----> Should run only on vault-0
```

```
Token:
```

=====

Run the below commands as written below

=====

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

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

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

=====

on pod vault-1

=====

kubectl exec -it vault-1 -n vault sh

\$ vault operator unseal -----> First Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Second Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Third Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

=====

on pod vault-2

=====

kubectl exec -it vault-2 -n vault sh

\$ vault operator unseal -----> First Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Second Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Third Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

=====

Finally, Run the below command to check the leader and follower of Hashicorp Vault

=====

kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers -----> Here you will see that vault-0 will be leader and vault-1 and vault-2 pod will be as forwarder

=====

Now check the status of three vault pods

=====

kubectl get pods -n vault

NAME	READY	STATUS	RESTARTS	AGE
vault-0	1/1	Running	0	3h46m
vault-1	1/1	Running	0	3h46m
vault-2	1/1	Running	0	3h46m
vault-agent-injector-7XXXXXXXXXb-mXXXv	1/1	Running	0	3h46m

The Pods will be in the running status and READY 1/1

```
[root@yellow ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault operator init
Unseal Key 1: 1yellow8
Unseal Key 2: Nyellowd
Unseal Key 3: Fyellowx
Unseal Key 4: 2yellowG
Unseal Key 5: Xyellowi

Initial Root Token: hyellow4

Vault initialized with 5 key shares and a key threshold of 3. Please securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 3 of these keys to unseal it
before it can start servicing requests.

Vault does not store the generated root key. Without at least 3 keys to
reconstruct the root key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
existing unseal keys shares. See "vault operator rekey" for more information.
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          ---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
Unseal Nonce  8yellowf
Version      1.19.0
Build Date   2025
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

Ritesh Kumar

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 2/3
Unseal Nonce  8 [REDACTED] f
Version      1.19.0
Build Date   2025-[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       false
Total Shares 5
Threshold    3
Version      1.19.0
Build Date   2025-[REDACTED]
Storage Type raft
Cluster Name vault-cluster-[REDACTED]
Cluster ID   6 [REDACTED]
Removed From Cluster false
HA Enabled   true
HA Cluster   https://vault-0.vault-internal:8201
HA Mode      active
Active Since 2025-[REDACTED]
Raft Committed Index 37
Raft Applied Index  37
```

```

/ $ vault login
Token (will be hidden):
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key          Value
---          -----
token        h[REDACTED]4
token_accessor [REDACTED]
token_duration   ∞
token_renewable  false
token_policies   ["root"]
identity_policies []
policies       ["root"]

[root@[REDACTED] ~]# kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers
Node           Address      State  Voter
---           -----
[REDACTED]     vault-0.vault-internal:8201  leader  true
[root@[REDACTED] ~]# kubectl exec -ti vault-1 -n vault -- vault operator raft join http://vault-0.vault-internal:8200
Key  Value
---  ---
Joined  true
[root@Terraform-Server ~]# kubectl exec -ti vault-2 -n vault -- vault operator raft join http://vault-0.vault-internal:8200
Key  Value
---  ---
Joined  true

[root@[REDACTED] main]# kubectl exec -it vault-1 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $

/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized  true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
UnsealNonce [REDACTED]
Version      1.19.0
Build Date   2025-[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $

/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized  true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 2/3
UnsealNonce [REDACTED]
Version      1.19.0
Build Date   2025-[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true

```

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 0/3
UnsealNonce  n/a
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $
```

```
[root@[REDACTED] main]# kubectl exec -it vault-2 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
UnsealNonce  [REDACTED]
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 2/3
UnsealNonce  [REDACTED]
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

K
Ritesh Kumar Singh

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 0/3
UnsealNonce  n/a
Version      1.19.0
Build Date   2025-03-14T14:44:00Z
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $
```

```
[root@REDACTED main]# kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers
Node          Address          State  Voter
---          -----
REDACTED        vault-0.vault-internal:8201  leader  true
REDACTED        vault-1.vault-internal:8201  follower  true
REDACTED        vault-2.vault-internal:8201  follower  true
```

Then I created the Ingress Rule and finally a Record Set of Type CNAME in Azure DNS Zone to access the vault in prod mode in dev cluster.

```
[root@REDACTED ~]# kubectl apply -f vault-dev-ingress-rule.yaml --context=eks-demo-cluster-dev
ingress.networking.k8s.io/vault-ingress created
[root@REDACTED ~]# kubectl get ing -A --watch --context=eks-demo-cluster-dev
NAMESPACE  NAME          CLASS   HOSTS          ADDRESS          PORTS   AGE
vault      vault-ingress  nginx   vault-dev.singhritesh85.com  aREDACTED.us-east-2.elb.amazonaws.com  80      15s
^C[root@REDACTED ~]#
[root@REDACTED ~]#
[root@REDACTED ~]# cat vault-dev-ingress-rule.yaml
...
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vault-ingress
  namespace: vault
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: vault-dev.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: vault-active
            port:
              number: 8200
```

```
cat vault-dev-ingress-rule.yaml
```

```
---
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vault-ingress
  namespace: vault
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: vault-dev.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
      backend:
        service:
          name: vault-active
          port:
            number: 8200
```

The screenshot shows the AWS Route 53 console under the 'singhritesh85.com' DNS zone. In the left sidebar, 'Recordsets' is selected. On the right, a modal window titled 'Add record set' is open, showing the configuration for a new CNAME record. The 'Name' field contains 'vault-dev', the 'Type' field is set to 'CNAME – Link your subdomain to another record', and the 'Alias record set' dropdown is set to 'No'. Below these, 'TTL' is set to 1 and 'TTL unit' is set to 'Hours'. The 'Alias' field contains 'a' with a value of '21'. At the bottom of the modal, the 'Add' button is highlighted with a yellow box.

Finally, I was able to access the Hashicorp vault in prod mode in dev cluster (EKS Cluster) as shown in the screenshot attached below.

The screenshot shows a browser window with the URL 'vault-dev.singhritesh85.com/ui/vault/auth?with=token'. The page title is 'Sign in to Vault'. It features a logo of a stylized hexagon with a vertical bar. Below the logo, there's a form with a 'Method' dropdown set to 'Token' and a 'Token' input field. A 'Sign in' button is at the bottom of the form. A note at the bottom says 'Contact your administrator for login credentials.' At the very bottom of the page, there's a footer with links to 'Vault', 'Upgrade to Vault Enterprise', 'Documentation', 'Support', 'Terms', 'Privacy', 'Security', 'Accessibility', and a copyright notice for HashiCorp.

Then I installed Nginx Ingress Controller and Hashicorp Vault in Prod mode in Stage Cluster (AKS Cluster).

Installation of Nginx Ingress Controller and Hashicorp Vault in Prod mode in AKS Cluster

Installation of Nginx Ingress Controller in AKS Cluster

To install Nginx Ingress Controller in AKS Cluster I used the command as written below.

kubectl create ns ingress-nginx

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx

helm upgrade ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx --set controller.service.annotations."service\\.beta\\.kubernetes\\.io/azure-load-balancer-health-probe-request-path"/=healthz --set controller.service.externalTrafficPolicy=Local

```
[root@yellow ~]# kubectl config get-contexts
CURRENT  NAME          CLUSTER           AUTHINFO
*        aks-cluster    aks-cluster        clusterUser_aks-rg_aks-cluster
*        eks-demo-cluster-dev  eks-demo-cluster-dev  arn:aws:eks:us-east-2:0:yellow:cluster/eks-demo-cluster-dev
[root@yellow ~]# kubectl config use-context aks-cluster
Switched to context "aks-cluster".
[root@yellow ~]# kubectl create ns ingress-nginx

[root@yellow ~]# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
"ingress-nginx" already exists with the same configuration, skipping
[root@yellow ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "hashicorp" chart repository
Update Complete. Happy Helm-ing!
[root@yellow ~]# helm install ingress-nginx ingress-nginx/ingress-nginx -n ingress-nginx

[root@yellow ~]# helm upgrade ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx --set controller.service.annotations."service\\.beta\\.kubernetes\\.io/azure-load-balancer-health-probe-request-path"/=healthz --set controller.service.externalTrafficPolicy=Local

[root@yellow ~]# kubectl get all -n ingress-nginx
NAME                           READY   STATUS    RESTARTS   AGE
pod/ingress-nginx-controller-  1/1     Running   0          5m43s
                                         TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/ingress-nginx-controller   LoadBalancer   10.0.0.227   4.0.0.216   80:32191/TCP,443:31275/TCP   5m43s
service/ingress-nginx-controller-admission   ClusterIP   10.0.0.213   <none>     443/TCP   5m43s
                                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1          1          5m43s
                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-  1         1         1         5m43s
```

Installation of Hashicorp Vault in Prod mode in AKS Cluster (Stage Environment)

To install Hashicorp Vault in AKS Cluster I used the commands as written below.

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

helm repo update

helm install vault hashicorp/vault --namespace vault --create-namespace --set server.image.tag=1.19.0,server.ha.enabled=true,server.ha.raft.enabled=true,server.ha.replicas=3,server.dev.enabled=false,server.ui.enabled=true,server.ui.serviceType=ClusterIP

kubectl get pods -n vault

```
[root@] ~]# helm repo add hashicorp https://helm.releases.hashicorp.com
"hashicorp" already exists with the same configuration, skipping
[root@] ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
...Successfully got an update from the "hashicorp" chart repository
Update Complete. Happy Helming!
[root@] ~]# helm install vault hashicorp/vault --namespace vault --create-namespace --set server.image.tag=1.19.0,server.ha.enabled=true,server.ha.raft.enabled=true,server.ha.replicas=3,server.dev.enabled=false,server.ui.enabled=true,server.ui.serviceType=ClusterIP
NAME: vault
LAST DEPLOYED: [REDACTED] 2025
NAMESPACE: vault
STATUS: deployed
REVISION: 1
NOTES:
Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using
Vault with Kubernetes available here:

https://developer.hashicorp.com/vault/docs

Your release is named vault. To learn more about the release, try:

$ helm status vault
$ helm get manifest vault
```

NAME	READY	STATUS	RESTARTS	AGE
vault-0	0/1	Running	0	110s
vault-1	0/1	Running	0	110s
vault-2	0/1	Running	0	110s
vault-agent-injector-[REDACTED]	1/1	Running	0	110s

Now logged-in to the vault pods in AKS Cluster and executed the below commands.

```
=====  
on pod vault-0  
=====
```

```
kubectl exec -it vault-0 -n vault sh
```

```
$ vault operator init
```

```
----- Key1
```

```
----- Key2
```

```
----- Key3
```

```
----- Key4
```

```
----- Key5
```

```
----- Root Token
```

```
$ vault operator unseal -----> First Time
```

```
Unseal Key:
```

```
$ vault operator unseal -----> Second Time
```

```
Unseal Key:
```

```
$ vault operator unseal -----> Third Time
```

```
Unseal Key:
```

```
$ vault login -----> Should run only on vault-0
```

```
Token:
```

=====

Then run the below commands as written below

=====

```
kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers
```

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

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

=====

on pod vault-1

=====

```
kubectl exec -it vault-1 -n vault sh
```

```
$ vault operator unseal -----> First Time
```

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

```
$ vault operator unseal -----> Second Time
```

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

```
$ vault operator unseal -----> Third Time
```

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

=====

on pod vault-2

=====

```
kubectl exec -it vault-2 -n vault sh
```

Ritesh Kumar Singh || Email Address: - riteshkumarsingh9559@gmail.com || LinkedIn: - <https://www.linkedin.com/in/ritesh-kumar-singh-41113128b/> || GitHub: - <https://github.com/singhritesh85>

\$ vault operator unseal -----> First Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Second Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Third Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

=====

Finally, Run the below command to check the leader and follower in vault

=====

kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers -----> Here you will see that vault-0 will be leader and vault-1 and vault-2 pod will be as forwarder

=====

Now check the status of three vault pods

=====

kubectl get pods -n vault

NAME	READY	STATUS	RESTARTS	AGE
vault-0	1/1	Running	0	3h46m
vault-1	1/1	Running	0	3h46m
vault-2	1/1	Running	0	3h46m
vault-agent-injector-7594d5f8bb-m486v	1/1	Running	0	3h46m

The Pods will be in the running status and READY 1/1

```
[root@yellow ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault operator init
Unseal Key 1: XyellowF
Unseal Key 2: OyellowJ
Unseal Key 3: 8yellowZ
Unseal Key 4: 8yellowI
Unseal Key 5: oyellowC

Initial Root Token: Hyellowh

Vault initialized with 5 key shares and a key threshold of 3. Please securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 3 of these keys to unseal it
before it can start servicing requests.

Vault does not store the generated root key. Without at least 3 keys to
reconstruct the root key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
existing unseal keys shares. See "vault operator rekey" for more information.
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
Unseal Nonce yellow
Version      1.19.0
Build Date   2025yellow
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

Ritesh Kumar

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 2/3
Unseal Nonce
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       false
Total Shares 5
Threshold    3
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Cluster Name vault-cluster-[REDACTED]
Cluster ID   [REDACTED]
Removed From Cluster false
HA Enabled   true
HA Cluster   https://vault-0.vault-internal:8201
HA Mode      active
Active Since 2025[REDACTED]
Raft Committed Index 37
Raft Applied Index 37
```

```

/ $ vault login
Token (will be hidden):
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key          Value
---          ---
token        h[REDACTED]h
token_accessor z[REDACTED]g
token_duration   infinity
token_renewable  false
token_policies   ["root"]
identity_policies []
policies       ["root"]

[root@[REDACTED] ~]# kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers
Node          Address      State  Voter
---          -----
[REDACTED]    vault-0.vault-internal:8201  leader  true
[root@[REDACTED] ~]# kubectl exec -ti vault-1 -n vault -- vault operator raft join http://vault-0.vault-internal:8200
Key          Value
---          ---
Joined      true
[root@Terraform-Server ~]# kubectl exec -ti vault-2 -n vault -- vault operator raft join http://vault-0.vault-internal:8200
Key          Value
---          ---
Joined      true

[root@[REDACTED] ~]# kubectl exec -it vault-1 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.

```

Ritesh Kumar

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold     3
Unseal Progress 1/3
Unseal Nonce
Version      1.19.0
Build Date   2025
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold     3
Unseal Progress 2/3
Unseal Nonce
Version      1.19.0
Build Date   2025
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

Ritesh

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 0/3
UnsealNonce  n/a
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

```
[root@[REDACTED] ~]# kubectl exec -it vault-2 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
UnsealNonce  [REDACTED]
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 2/3
UnsealNonce  [REDACTED]
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

Ritesh Kumar Singh

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 0/3
UnsealNonce  n/a
Version      1.19.0
Build Date   2025-04-18T14:45:00Z
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $
[root@[REDACTED] ~]#
```

```
[root@[REDACTED] ~]# kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers
Node          Address          State    Voter
-----
[REDACTED]           vault-0.vault-internal:8201  leader   true
[REDACTED]           vault-1.vault-internal:8201  follower true
[REDACTED]           vault-2.vault-internal:8201  follower true
```

To create the Ingress Rule, I created the Kubernetes Secrets first as shown in the screenshot attached below.

```
kubectl create secret tls vault-tls-secret --cert=STAR_singhritesh85_com.crt --key=mykey.key -n vault
```

```
[root@[REDACTED] ~]# kubectl create secret tls vault-tls-secret --cert=STAR_singhritesh85_com.crt --key=mykey.key -n vault
secret/vault-tls-secret created
```

Then I created the Ingress Rule and finally a Record Set of A Type in Azure DNS Zone to access the vault in prod mode in dev cluster.

```
[root@XXXXXXXXXX ~]# kubectl apply -f vault-stage-ingress-rule.yaml --context=aks-cluster
ingress.networking.k8s.io/vault-ingress created
[root@XXXXXXXXXX ~]# kubectl get ing -A --watch --context=aks-cluster
NAMESPACE   NAME      CLASS    HOSTS          ADDRESS        PORTS      AGE
vault       vault-ingress  nginx  vault-stage.singhritesh85.com  80, 443   4s
vault       vault-ingress  nginx  vault-stage.singhritesh85.com  4.1XXXXXX.216  80, 443  10s
^C[root@XXXXXXXXXX ~]#
[root@XXXXXXXXXX ~]# cat vault-stage-ingress-rule.yaml
# kubectl create secret tls vault-tls-secret --cert=STAR_singhritesh85_com.crt --key=mykey.key -n vault
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vault-ingress
  namespace: vault
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - vault-stage.singhritesh85.com
    secretName: vault-tls-secret
  rules:
  - host: vault-stage.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: vault-active
            port:
              number: 8200
```

Ritesh Kumar

```
cat vault-stage-ingress-rule.yaml

# kubectl create secret tls vault-tls-secret --cert=STAR_singhrites85_com.crt --key=mykey.key -n
vault

---

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vault-ingress
  namespace: vault
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - vault-stage.singhrites85.com
    secretName: vault-tls-secret
  rules:
  - host: vault-stage.singhrites85.com
    http:
      paths:
      - path: /
        pathType: Prefix
      backend:
        service:
          name: vault-active
        port:
          number: 8200
```

singhritesh85.com | Recordsets

A record set is a collection of records in a zone that have been loaded on this page. If you don't see what you're looking for, try [Learn more](#).

TTL	Value
172800	[REDACTED]
3600	[REDACTED]
3600	[REDACTED]

Add or remove favorites by pressing **Ctrl+Shift+F**

Add record set

Name: **vault-stage**

Type: **A – IPv4 Address records**

Alias record set: **No**

TTL *: **1**

TTL unit: **Hours**

IP address: **4.192.216.0.0.0**

Add **Cancel** **Give feedback**

Sign in to Vault

Method: Token

Token: [REDACTED]

Sign in

Contact your administrator for login credentials.

For the first time login method into the vault is Token. The Token which I got from the vault-0 pod in dev cluster and stage cluster was used to login into the Hashicorp vault prod mode in dev cluster and stage cluster respectively. Then I enabled the vault auth userpass and created a username with password with Administrator privileges which I will use to login in to the Hashicorp Vault prod mode in dev cluster (EKS Cluster) and stage cluster (AKS Cluster) as shown in the screenshot attached below.

```
$ vault auth enable userpass
```

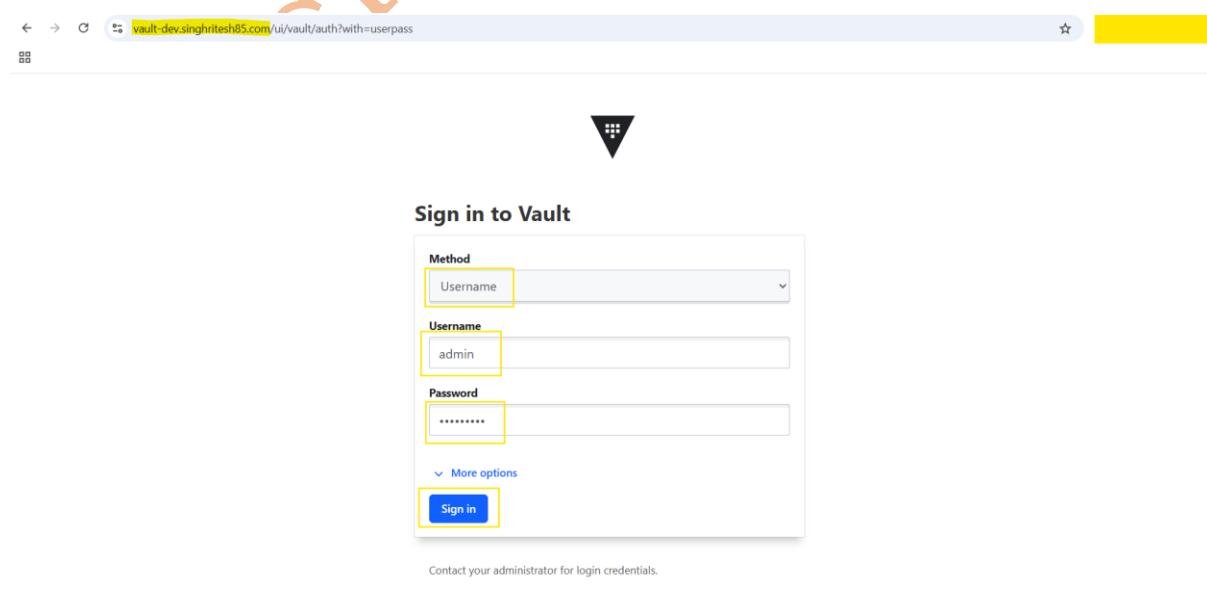
```
$ vault policy write admin -<<EOF
```

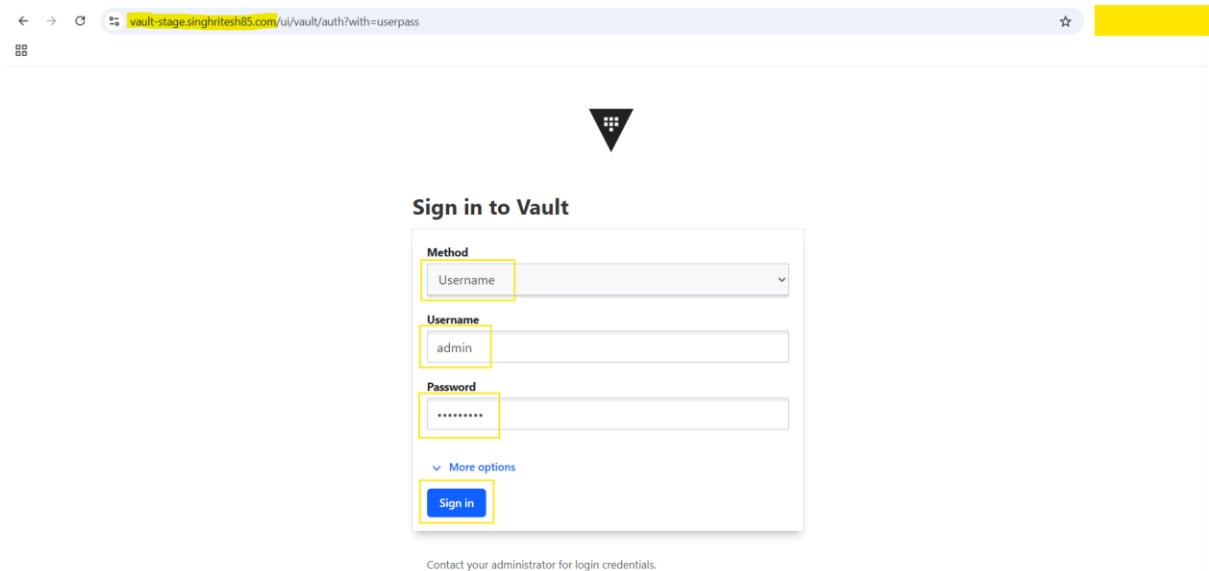
```
path "*" {
  capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}
EOF
```

```
$ vault write auth/userpass/users/admin password=Admin@123 policies=admin
```

```
[root@██████████ ~]# kubectl exec -it vault-0 -n vault sh --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault auth enable userpass
Success! Enabled userpass auth method at: userpass/
/ $ vault policy write admin -<<EOF
> path "*" {
>   capabilities = ["create", "read", "update", "delete", "list", "sudo"]
> }
> EOF
Success! Uploaded policy: admin
/ $ vault write auth/userpass/users/admin password=Admin@123 policies=admin
Success! Data written to: auth/userpass/users/admin
/ $
[root@██████████ ~]# kubectl exec -it vault-0 -n vault sh --context=eks-demo-cluster-dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault auth enable userpass
Success! Enabled userpass auth method at: userpass/
/ $ vault policy write admin -<<EOF
> path "*" {
>   capabilities = ["create", "read", "update", "delete", "list", "sudo"]
> }
> EOF
Success! Uploaded policy: admin
/ $ vault write auth/userpass/users/admin password=Admin@123 policies=admin
Success! Data written to: auth/userpass/users/admin
```

Then I logged-in with the created username and password as shown in the screenshot attached below.





Then created the Hashicorp vault secrets in dev cluster (EKS Cluster) and stage cluster (AKS Cluster) as shown in the screenshot attached below.

```
[root@yellow ~]# kubectl exec -it vault-0 -n vault sh --context=eks-demo-cluster-dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
```

```
/ $ vault secrets enable -path=bankapp kv-v2
Success! Enabled the kv-v2 secrets engine at: bankapp/
/ $
/ $ vault kv put bankapp/database/config password=Dexter@123
===== Secret Path =====
bankapp/data/database/config

===== Metadata =====
Key          Value
---          -----
created_time 2025-05-18T06:37:57.871570522Z
custom_metadata <nil>
deletion_time n/a
destroyed      false
version        1
/ $
/ $ vault kv get bankapp/database/config
===== Secret Path =====
bankapp/data/database/config

===== Metadata =====
Key          Value
---          -----
created_time 2025-05-18T06:37:57.871570522Z
custom_metadata <nil>
deletion_time n/a
destroyed      false
version        1

===== Data =====
Key          Value
---          -----
password     Dexter@123
```

Ritesh

```
[root@ ~]# kubectl exec -it vault-0 -n vault sh --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault secrets list
Path          Type      Accessor      Description
----          ----      -----      -----
cubbyhole/    cubbyhole  cubbyhole_947c03b7  per-token private secret storage
identity/    identity   identity_47365a9f  identity store
sys/         system    system_ea37fc08  system endpoints used for control, policy and debugging
/ $ vault secrets enable -path=bankapp kv-v2
Success! Enabled the kv-v2 secrets engine at: bankapp/
/ $ vault kv put bankapp/database/config password=Dexter@123
===== Secret Path =====
bankapp/data/database/config

===== Metadata =====
Key          Value
---          ---
created_time 2025
custom_metadata <nil>
deletion_time n/a
destroyed     false
version       1
/ $ vault kv get bankapp/database/config
===== Secret Path =====
bankapp/data/database/config

===== Metadata =====
Key          Value
---          ---
created_time 2025
custom_metadata <nil>
deletion_time n/a
destroyed     false
version       1

===== Data =====
Key          Value
---          ---
password    Dexter@123
```

You can verify the same secrets in your dev cluster (EKS Cluster) and stage cluster (AKS Cluster) Hashicorp Vault UI as shown in the screenshot attached below.

The screenshot shows the Hashicorp Vault UI interface. The left sidebar has a dark theme with orange icons for Vault, Dashboard, Secrets Engines, Access, Policies, Tools, Monitoring, Raft Storage, Client Count, and Seal Vault. The 'Secrets Engines' section is currently selected. The main area shows a breadcrumb path: Secrets / bankapp / database / config. Below this, there is a table with a single row:

Key	Value
password	Dexter@123

At the bottom of the UI, there are links for Vault 1.19.0, Upgrade to Vault Enterprise, Documentation, Support, Terms, Privacy, Security, Accessibility, and a copyright notice: © 2025 HashiCorp.

The screenshot shows the HashiCorp Vault UI interface. On the left is a sidebar with 'Vault' and 'Dashboard' sections, and 'Secrets Engines' highlighted. The main area shows a breadcrumb path: Secrets / bankapp / database / config. Below this is a table with one row, where 'password' is the key and 'Dexter@123' is the value. A yellow box highlights the 'password' key and its value. At the bottom right of the table, it says 'Version 1 created May 18, 2025 01:23 PM'. The footer includes links for Vault 1.19.0, Upgrade to Vault Enterprise, Documentation, Support, Terms, Privacy, Security, Accessibility, and © 2025 HashiCorp.

Create and apply the policy in vault to read secrets from Hashicorp Vault in EKS and AKS Cluster

Next, I had created a policy which allowed reading secrets. This policy was attached to a role, which could be used to grant access to specific Kubernetes service accounts.

```
vault policy write bankapp-policy <<EOF
path "bankapp/data/database/config" {
  capabilities = ["read"]
}
EOF
```

```
[root@... ~]# kubectl exec -it vault-0 -n vault sh --context=eks-demo-cluster-dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
```

```
/ $ vault policy write bankapp-policy <<EOF
> path "bankapp/data/database/config" {
>   capabilities = ["read"]
> }
> EOF
Success! Uploaded policy: bankapp-policy
```

```
/ $ vault policy list
admin
bankapp-policy
default
root
/ $ vault policy read bankapp-policy
path "bankapp/data/database/config" {
    capabilities = ["read"]
}
/ $
```

vault policy list

vault policy read <policy-name>

[root@**XXXXXXXXXX** ~]# kubectl exec -it vault-0 -n vault sh --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.

```
/ $ vault policy write bankapp-policy -<<EOF
> path "bankapp/data/database/config" {
>     capabilities = ["read"]
> }
> EOF
```

```
/ $ vault policy list
admin
bankapp-policy
default
root
/ $ vault policy read bankapp-policy
path "bankapp/data/database/config" {
    capabilities = ["read"]
}
/ $
```

Enable the Kubernetes authentication method in Vault in EKS and AKS Cluster as shown in the screenshot attached below.

Vault auth enable kubernetes

[root@**XXXXXXXXXX** ~]# kubectl exec -it vault-0 -n vault sh --context=eks-demo-cluster-dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ \$ vault auth enable kubernetes
Success! Enabled kubernetes auth method at: kubernetes/
/ \$
[root@**XXXXXXXXXX** ~]# kubectl exec -it vault-0 -n vault sh --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ \$ vault auth enable kubernetes
Success! Enabled kubernetes auth method at: kubernetes/
/ \$

Configure Vault to communicate with the Kubernetes API server on EKS and AKS Cluster as shown in the screenshot attached below.

```
vault write auth/kubernetes/config \
token_reviewer_jwt=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \
kubernetes_host=https:// ${KUBERNETES_PORT_443_TCP_ADDR}:443 \
kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
```

```
[root@yellow ~]# kubectl exec -it vault-0 -n vault sh --context=eks-demo-cluster-dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault write auth/kubernetes/config \
> token_reviewer_jwt=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \
> kubernetes_host=https:// ${KUBERNETES_PORT_443_TCP_ADDR}:443 \
> kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
Success! Data written to: auth/kubernetes/config
/ $
[root@yellow ~]# kubectl exec -it vault-0 -n vault sh --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault write auth/kubernetes/config \
> token_reviewer_jwt=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \
> kubernetes_host=https:// ${KUBERNETES_PORT_443_TCP_ADDR}:443 \
> kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
Success! Data written to: auth/kubernetes/config
/ $
```

Then I created a role named as bankapp that binds the policy bankapp-policy to a Kubernetes service account(bankapp-sa) in a specific namespace mysql. This allows the service account to access secrets stored in Hashicorp Vault

```
vault write auth/kubernetes/role/bankapp bound_service_account_names=bankapp-sa
bound_service_account_namespaces=mysql policies=bankapp-policy ttl=1h
```

```
[root@yellow ~]# kubectl exec -it vault-0 -n vault sh --context=eks-demo-cluster-dev
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault write auth/kubernetes/role/bankapp bound_service_account_names=bankapp-sa bound_service_account_namespaces=mysql policies=bankapp-policy ttl=1h
Success! Data written to: auth/kubernetes/role/bankapp
/ $
[root@yellow ~]# kubectl exec -it vault-0 -n vault sh --context=aks-cluster
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault write auth/kubernetes/role/bankapp bound_service_account_names=bankapp-sa bound_service_account_namespaces=mysql policies=bankapp-policy ttl=1h
Success! Data written to: auth/kubernetes/role/bankapp
/ $
```

The vault secrets will be accessed from mysql database which will exist in mysql namespace so here I had created a namespace mysql then service account named as vault-sa in namespace mysql on EKS and AKS Cluster.

```
kubectl create ns mysql
```

```
kubectl create sa bankapp-sa-n mysql
```

```
[root@yellow ~]# kubectl config get-contexts
CURRENT NAME CLUSTER AUTHINFO NAMESPACE
* eks-demo-cluster-dev eks-demo-cluster-dev clusterUser_aks-rg_aks-cluster
[root@yellow ~]# kubectl create ns mysql
namespace/mysql created
[root@yellow ~]# kubectl create sa bankapp-sa -n mysql
serviceaccount/bankapp-sa created
[root@yellow ~]# kubectl config use-context aks-cluster
Switched to context "aks-cluster".
[root@yellow ~]# kubectl create ns mysql
namespace/mysql created
[root@yellow ~]# kubectl create sa bankapp-sa -n mysql
serviceaccount/bankapp-sa created
```

Now I install ArgoCD in EKS Cluster (dev Cluster) and then created the ingress rule for ArgoCD to access it by adding the DNS Name of the LoadBalancer in Azure DNS Zone by creating a Record Set of CNAME Type then added AKS Cluster into ArgoCD as shown in the screenshot attached below.

Installation of ArgoCD

```
kubectl create namespace argocd
```

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

You can get the default password of admin user in ArgoCD using the command as written below

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath='{.data.password}' | base64 -d
```

```
[root@XXXXXXXXXX ~]# kubectl config get-contexts
CURRENT   NAME          CLUSTER           AUTHINFO
*         aks-cluster    aks-cluster        clusterUser_aks_aks-cluster
          eks-demo-cluster-dev  eks-demo-cluster-dev  arn:aws:eks:us-east-2:02XXXXXXXXXX:cluster/eks-demo-cluster-dev
[root@XXXXXXXXXX ~]# kubectl config use-context eks-demo-cluster-dev
Switched to context "eks-demo-cluster-dev".
[root@XXXXXXXXXX ~]# kubectl create namespace argocd
namespace/argocd created
[root@XXXXXXXXXX ~]# kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

[root@XXXXXXXXXX ~]# kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath='{.data.password}' | base64 -d
k[b][root@XXXXXXXXXX ~]#
```

Ritesh Kumar Singh

```
cat argocd-ingress-rule.yaml

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  namespace: argocd
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"    ### You can use this option for this particular case for ArgoCD but not for all
    # nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  ingressClassName: nginx
  rules:
  - host: argocd.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
      backend:
        service:
          name: argocd-server  ### Provide your service Name
        port:
          number: 80    ##### Provide your service port for this particular example you can also choose 443
```

```
[root@yellow ~]# kubectl apply -f argocd-ingress-rule.yaml
ingress.networking.k8s.io/minimal-ingress created
[root@yellow ~]# cat argocd-ingress-rule.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  namespace: argocd
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"    ### You can use this option for this particular case for ArgoCD but not for all
    # nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  ingressClassName: nginx
  rules:
  - host: argocd.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
      backend:
        service:
          name: argocd-server  ### Provide your service Name
        port:
          number: 80    ##### Provide your service port for this particular example you can also choose 443

[root@yellow ~]# kubectl get ing -A --watch
NAMESPACE   NAME        CLASS      HOSTS           ADDRESS          PORTS      AGE
argocd      minimal-ingress  nginx     argocd.singhritesh85.com  a[REDACTED] 6.us-east-2.elb.amazonaws.com  80      85s
vault       vault-ingress  nginx     vault-dev.singhritesh85.com a[REDACTED] 6.us-east-2.elb.amazonaws.com  80      140m
```

The screenshot shows the AWS Route 53 console for the domain singhritesh85.com. On the left, there's a navigation pane with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Settings, DNS Management, and Recordsets. The Recordsets section is selected. On the right, a modal window titled 'Add record set' is open, showing the configuration for a new CNAME record. The 'Name' field contains 'argocd', the 'Type' is 'CNAME - Link your subdomain to another record', and the 'Alias' field contains 'us-east-2.elb.amazonaws.com'. Other fields include 'TTL *' set to 1, 'TTL unit' set to Hours, and 'Alias record set' set to 'No'. At the bottom of the modal are 'Add', 'Cancel', and 'Give feedback' buttons.

Finally, I was able to login into the ArgoCD and successfully changed its password from default admin.

The screenshot shows the ArgoCD web interface at the URL argocd.singhritesh85.com/applications. The left sidebar has a dark theme with icons for Applications, Settings, User Info, and Documentation. The main area has a light background with a large circular icon containing three stacked rectangles. Below the icon, the text 'No applications available to you just yet' is displayed, followed by 'Create new application to start managing resources in your cluster' and a 'CREATE APPLICATION' button.

Then I had installed ArgoCD CLI on Jenkins Slave Node using the commands as shown in the screenshot attached below.

```
curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64
sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd
rm argocd-linux-amd64
```

Then I had added the AKS Cluster using the command **argocd cluster add aks-cluster** as shown in the screenshot attached below.

```
argocd login argocd.singhritesh85.com --username admin --password Admin@123 --skip-test-tls --grpc-web
```

```
argocd cluster add aks-cluster
```

```
[jenkins@[REDACTED] ~]$ argocd login argocd.singhritesh85.com --username admin --password Admin@123 --skip-test-tls --grpc-web
'admin/login' logged in successfully
Context 'argocd.singhritesh85.com' updated
[jenkins@[REDACTED] ~]$ argocd cluster add aks-cluster
WARNING: This will create a service account `argocd-manager` on the cluster referenced by context `aks-cluster` with full cluster level privileges. Do you want to continue [Y/N]? y
{"level": "info", "msg": "ServiceAccount \"argocd-manager\" created in namespace \"kube-system\"", "time": "2025-01-12T10:45:23.000Z"}
{"level": "info", "msg": "ClusterRole \"argocd-manager-role\" created", "time": "2025-01-12T10:45:23.000Z"}
{"level": "info", "msg": "ClusterRoleBinding \"argocd-manager-role-binding\" created", "time": "2025-01-12T10:45:23.000Z"}
{"level": "info", "msg": "Created bearer token secret for ServiceAccount \"argocd-manager\"", "time": "2025-01-12T10:45:23.000Z"}
Cluster 'https://aks-cluster-dns-[REDACTED].eastus.azurek8s.io:443' added
```

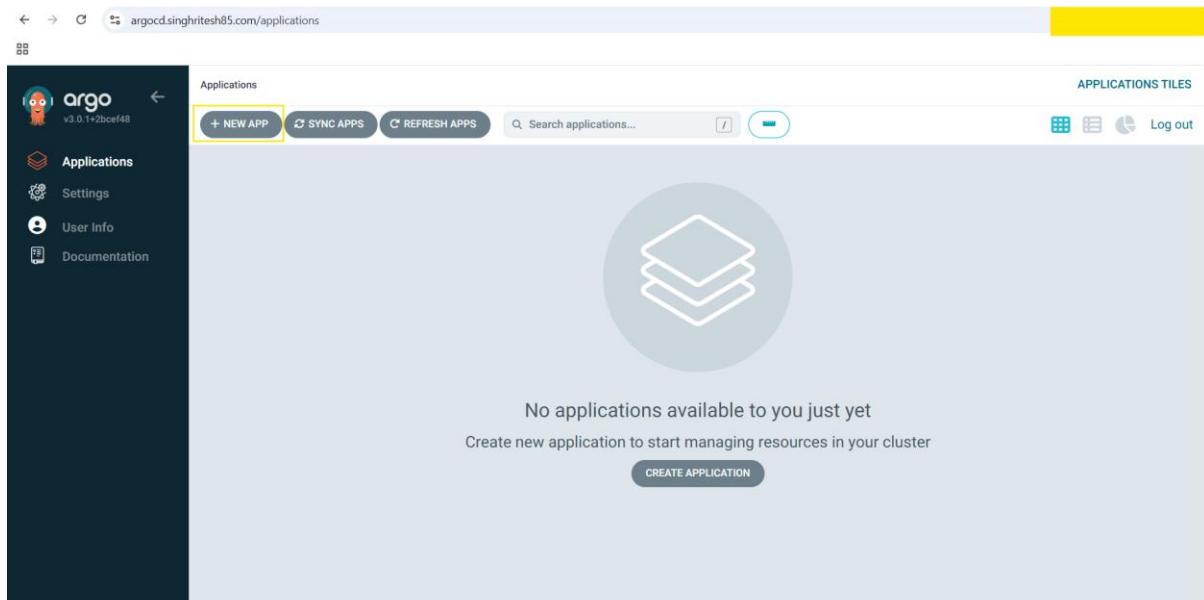
NAME	URL	VERSION	CONNECTION STATUS
aks-cluster	https://aks-cluster-dns-[REDACTED].eastus.azurek8s.io:443		Unknown
in-cluster	https://kubernetes.default.svc		Unknown

On Jenkins Slave Node I had provided Authentication and Authorization using Access Key and Secret Key as shown in the screenshot attached below.

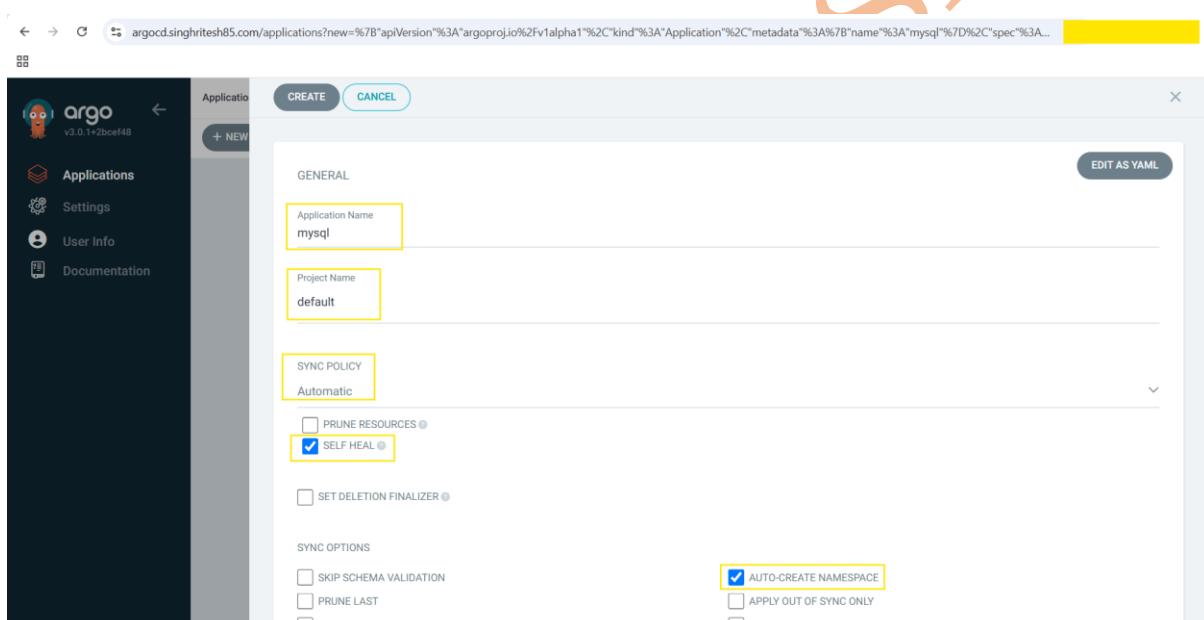
```
[jenkins@[REDACTED] ~]$ aws configure
AWS Access Key ID [None]: [REDACTED]
AWS Secret Access Key [None]: [REDACTED]
Default region name [None]: [REDACTED]
Default output format [None]: [REDACTED]
```

Module-1: Manual Approach

This project is basically a multibranch pipeline project in which only dev and stage environment exists. I had created mysql pod in dev and stage environment using the ArgoCD as shown in the screenshot attached below.



No applications available to you just yet
Create new application to start managing resources in your cluster
CREATE APPLICATION



CREATE **CANCEL**

GENERAL

Application Name: mysql

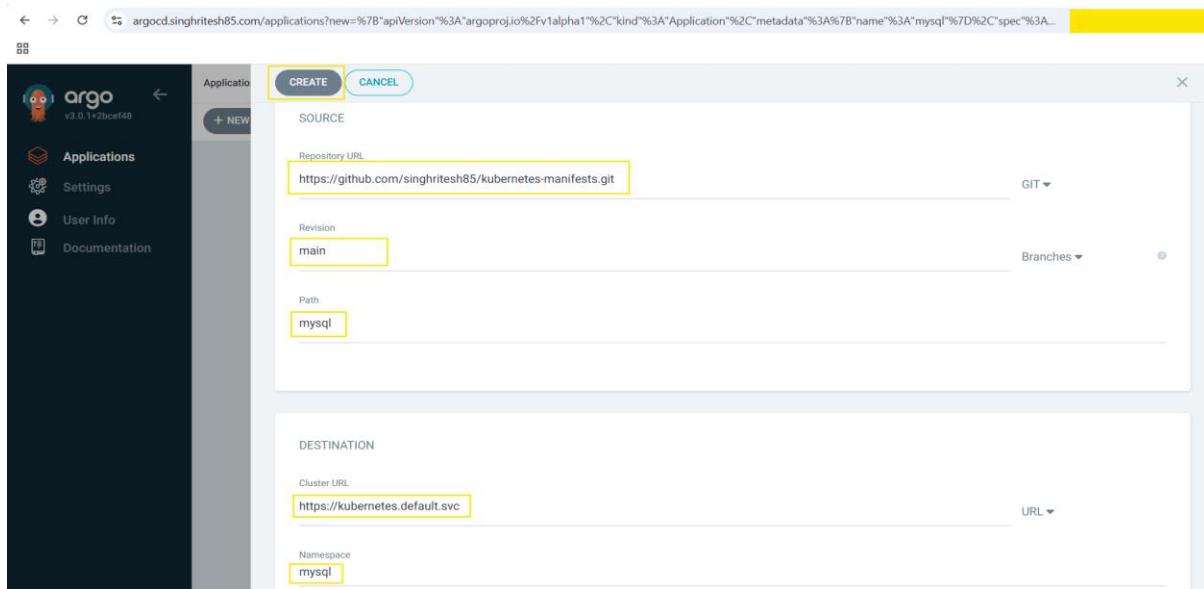
Project Name: default

SYNC POLICY: Automatic

PRUNE RESOURCES SELF HEAL SET DELETION FINALIZER

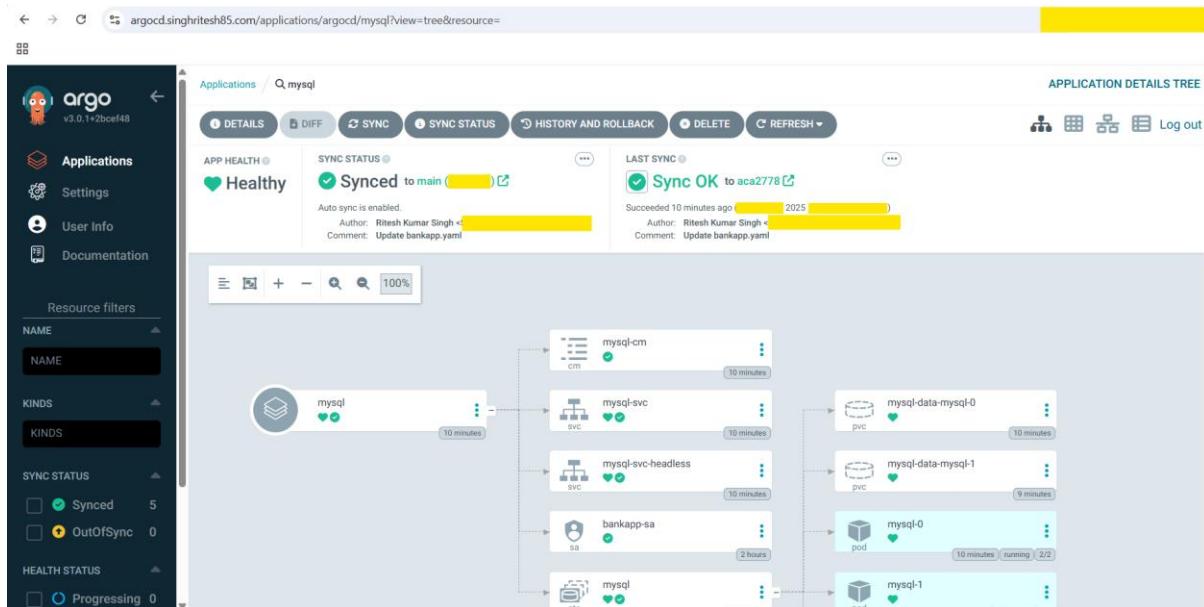
SYNC OPTIONS

SKIP SCHEMA VALIDATION AUTO-CREATE NAMESPACE
 PRUNE LAST APPLY OUT OF SYNC ONLY
 RESPECT IGNORE DIFFERENCES SERVER-SIDE APPLY

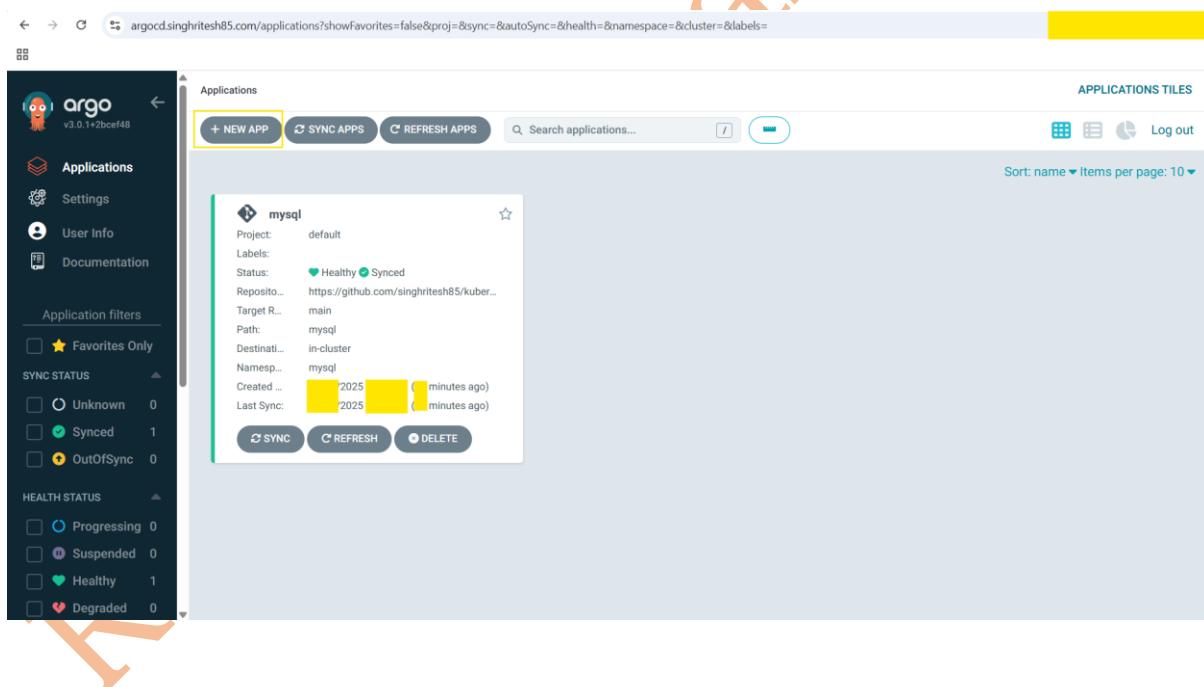


After creation of the Application mysql as shown in screenshot attached below the mysql pods had been created on dev cluster (EKS Cluster) as shown in the screenshot attached below.

```
[jenkins@... ~]$ kubectl get pods -n mysql
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          9m
mysql-1   2/2     Running   0          9m
[jenkins@... ~]$ kubectl get svc -n mysql
NAME        TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
mysql-svc   ClusterIP  10.0.4.11   <none>       3306/TCP  9m
mysql-svc-headless   ClusterIP  None        <none>       3306/TCP  9m
[jenkins@... ~]$ kubectl get pvc -n mysql
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mysql-data-mysql-0   Bound    pvc-...   1Gi        RWO          gp2           <unset>        <unset>        9m
mysql-data-mysql-1   Bound    pvc-...   1Gi        RWO          gp2           <unset>        <unset>        9m
```



Then I had created **New App** in ArgoCD to deploy pods on AKS Cluster (Stage Environment) as shown in the screenshot attached below.



The screenshots show the Argo UI interface for creating a new application. The top screenshot displays the 'GENERAL' configuration, including the application name 'mysql-stage', project name 'default', sync policy set to 'Automatic', and the 'SELF HEAL' option selected under 'SYNC POLICY'. The bottom screenshot shows the 'SOURCE' configuration, where the repository URL is set to 'https://github.com/singhritesh85/kubernetes-manifests.git', the revision is 'main', and the path is 'mysql-stage-aks'. The 'DESTINATION' configuration shows the cluster URL as 'https://aks-cluster-dns-.eastus.azurek8s.io:443' and the namespace as 'mysql'.

After creation of the Application **mysql-stage** as shown in screenshot attached below the mysql pods had been created on stage cluster (AKS Cluster) as shown in the screenshot attached below.

```
[jenkins@  1Gi        Rwo        managed-csi  <unset>           2m21s
mysql-data-mysql-1   Bound    pvc-  1Gi        Rwo        managed-csi  <unset>           86s
```

The screenshot shows the Argo CD interface with two application configurations listed:

- mysql**: Project: default. Status: Healthy Synced. Repository: https://github.com/singhritesh85/kuber... Target R.: main. Path: mysql. Destination: in-cluster. Namespace: mysql. Created: 2025-02-20. Last Sync: 2025-02-20.
- mysql-stage**: Project: default. Status: Healthy Synced. Repository: https://github.com/singhritesh85/kuber... Target R.: main. Path: mysql-stage-aks. Destination: aks-cluster. Namespace: mysql. Created: 2025-02-20. Last Sync: 2025-02-20.

On the left sidebar, there are filters for Application filters (Favorites Only), SYNC STATUS (Unknown: 0, Synced: 2, OutOfSync: 0), and HEALTH STATUS (Progressing: 0, Suspended: 0, Healthy: 2, Degraded: 0).

To build the Bank Application code presented in the GitHub Repo <https://github.com/singhritesh85/Bank-App-using-Hashicorp-Vault.git> and created Docker Image then pushed the Docker Image to ECR (Elastic Container Registry). This Docker Image will be used to deploy pods in EKS and AKS Cluster.

To access Docker Image from ECR (Elastic Container Registry) I created kubernetes secret in AKS Cluster as shown in the screenshot attached below.

```
kubectl create secret docker-registry regcred --docker-server=02XXXXXXXXX6.dkr.ecr.us-east-2.amazonaws.com --docker-username=AWS --docker-password=$(aws ecr get-login-password) --namespace=bankapp --context=aks-cluster
```

```
[root@ip-172-31-10-10 ~]# kubectl create secret docker-registry regcred --docker-server=02XXXXXXXXX6.dkr.ecr.us-east-2.amazonaws.com --docker-username=AWS --docker-password=$(aws ecr get-login-password) --namespace=bankapp --context=aks-cluster
secret/regcred created
```

However, I had Attached IAM Role to EKS Node Group and hence the EKS Nodes so that they can easily access the ECR Private Repository whenever needed.

I had built the Bank Application code and created the Docker Image then pushed the Docker Image to ECR (Elastic Container Registry) using the Multibranch Jenkins Job as shown in the screenshot attached below.

The screenshot shows the Jenkins job configuration interface for a job named "bankapp".

General Tab:

- Display Name: bankapp
- Description: Plain text (empty)
- Enabled: Yes (blue toggle switch)

Branch Sources Tab:

- Git Project Repository: https://github.com/singhritesh85/Bank-App-using-Hashicorp-Vault.git
- Credentials: singhritesh85/******** (github-cred)
- Behaviors:
 - Discover branches: ?
 - Filter by name (with wildcards):
 - Include: dev stage

Configuration Sidebar:

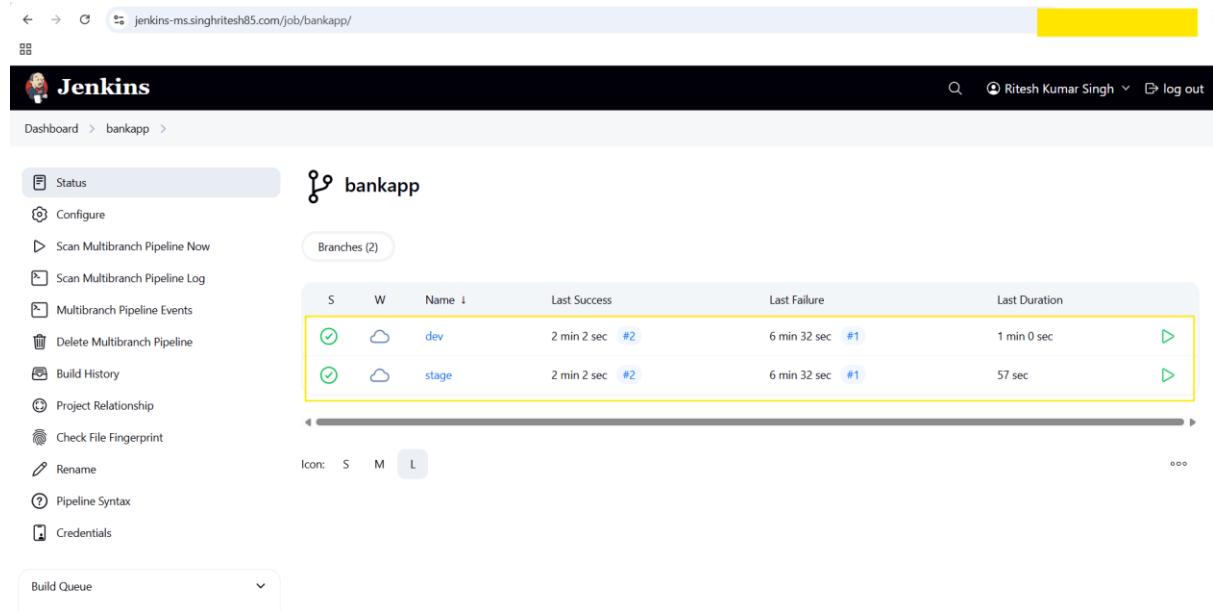
- General
- Branch Sources** (selected)
- Build Configuration
- Scan Multibranch Pipeline Triggers
- Orphaned Item Strategy
- Appearance
- Health metrics
- Properties

The screenshot shows the Jenkins job configuration page for 'bankapp'. The 'Build Configuration' section is selected. The 'Mode' dropdown is set to 'by Jenkinsfile'. The 'Script Path' input field contains 'Jenkinsfile-manual-approach'. Below this, the 'Scan Multibranch Pipeline Triggers' section has a checkbox for 'Periodically if not otherwise run' which is unchecked. The 'Orphaned Item Strategy' section contains two checkboxes: 'Abort builds' and 'Discard old items', both of which are unchecked. At the bottom are 'Save' and 'Apply' buttons.

After doing this I had seen two Jenkins Job created with the name as that of the branch name in the multi branch Jenkins Job as shown in the screenshot attached below.

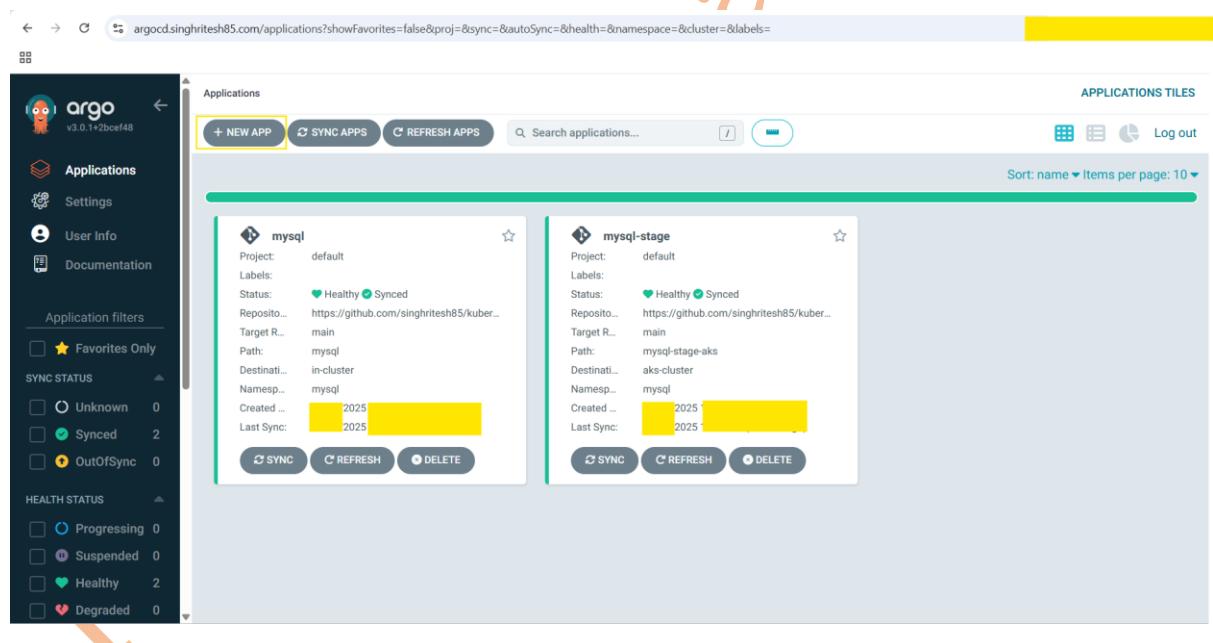
The screenshot shows the Jenkins dashboard. In the center, there is a table with one row. The columns are labeled 'S', 'W', 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The row contains icons for a person (S), a sun (W), the job name 'bankapp' (Name), '1 min 27 sec' (Last Success), 'N/A' (Last Failure), and '0.78 sec' (Last Duration). At the bottom of the dashboard, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (Built-in Node: 0/2, Slave-1: 0/2).

REST API Jenkins 2.504.1



The screenshot shows the Jenkins dashboard for a project named 'bankapp'. On the left, there's a sidebar with various Jenkins management options like 'Scan Multibranch Pipeline Now', 'Build History', and 'Pipeline Syntax'. The main area displays two branches: 'dev' and 'stage'. Each branch has a green checkmark icon, a cloud icon, and a timestamp indicating the last success ('2 min 2 sec #2'). The 'stage' branch also shows a timestamp for the last failure ('6 min 32 sec #1') and a duration of '57 sec'. A yellow box highlights the 'dev' and 'stage' rows.

Now Docker Image is ready in ECR (Elastic Container Registry). I had created two ECRs bankapp-1 (for dev cluster) and bankapp-2 (for stage-cluster). Then I created the ArgoCD Application to deploy the pods on Kubernetes Clusters (in dev and stage environment) using these Docker Images as shown in the screenshot attached below.



The screenshot shows the ArgoCD application dashboard. On the left, there's a sidebar with 'Applications', 'Settings', 'User Info', and 'Documentation'. Below that are sections for 'SYNC STATUS' and 'HEALTH STATUS'. The main area shows two applications: 'mysql' and 'mysql-stage'. Both applications are listed under the 'default' project. They both have a status of 'Healthy' and 'Synced'. The 'mysql' application has a target path of 'main' and a destination of 'in-cluster'. The 'mysql-stage' application has a target path of 'main' and a destination of 'aks-cluster'. Each application has 'SYNC', 'REFRESH', and 'DELETE' buttons at the bottom. A yellow box highlights the 'mysql' and 'mysql-stage' application cards.

The screenshot shows the Argo CD interface for creating a new application. The left sidebar displays the Argo CD dashboard with sections for Applications, Settings, User Info, Documentation, Application filters, SYNC STATUS, and HEALTH STATUS. The main area is a modal window titled 'CREATE' for an application named 'bankapp'. The 'GENERAL' tab is active, showing the application name 'bankapp' and project name 'default'. Under 'SYNC POLICY', 'Automatic' is selected. In the 'SYNC OPTIONS' section, 'SELF HEAL' is checked, and 'AUTO-CREATE NAMESPACE' is also checked. The 'SOURCE' tab is partially visible below, showing a repository URL of 'https://github.com/singhritesh85/kubernetes-manifests.git', revision 'main', path 'bankapp', and a destination cluster URL of 'https://kubernetes.default.svc'.

GENERAL

Application Name: bankapp-stage

Project Name: default

Sync Policy: Automatic

PRUNE RESOURCES

SELF HEAL

SET DELETION FINALIZER

SYNC OPTIONS

SKIP SCHEMA VALIDATION

PRUNE LAST

RESPECT IGNORE DIFFERENCES

AUTO-CREATE NAMESPACE

APPLY OUT OF SYNC ONLY

SERVER-SIDE APPLY

SOURCE

Repository URL: https://github.com/singhritesh85/kubernetes-manifests.git

Revision: main

Path: bankapp-stage

DESTINATION

Cluster URL: https://aks-cluster-dns-eastus.azurek8s.io:443

Namespace: bankapp

Finally, bankapp pods had been created in the EKS cluster (dev cluster) and AKS Cluster (Stage Cluster) as shown in the screenshot attached below.

```
[jenkins@... ~]$ kubectl get pods -n bankapp --watch --context=eks-demo-cluster-dev
NAME           READY   STATUS    RESTARTS   AGE
bankapp-...     1/1     Running   0          1m
bankapp-...     1/1     Running   0          1m
^C[jenkins@... ~]$
[jenkins@... ~]$ kubectl get pods -n bankapp --watch --context=aks-cluster
NAME           READY   STATUS    RESTARTS   AGE
bankapp-...     1/1     Running   0          1m
bankapp-...     1/1     Running   0          1m

[jenkins@... ~]$ kubectl get svc -n bankapp --watch --context=eks-demo-cluster-dev
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
bankapp-service   ClusterIP  10.10.182   <none>        80/TCP   1m
[jenkins@... ~]$
[jenkins@... ~]$ kubectl get svc -n bankapp --watch --context=aks-cluster
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
bankapp-service   ClusterIP  10.10.229   <none>        80/TCP   1m
```

Then I created Ingress Rule in dev cluster and stage cluster to access the Bank Application and did the entry for DNS Name and External IP Address of LoadBalancer in the Azure DNS Zone Record Set of Type CNAME and A Type respectively as shown in the screenshot attached below.

I had created the kubernetes secret in AKS Cluster for the Ingress Rule as shown in the screenshot attached below.

```
kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --
namespace bankapp --context=aks-cluster
```

```
[root@... ~]# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace bankapp --context=aks-cluster
secret/ingress-tls created

[root@... ~]# kubectl apply -f ingress-rule-stage.yaml --context=aks-cluster
ingress.networking.k8s.io/bankapp-ingress-stage created
[root@... ~]# kubectl apply -f ingress-rule-dev.yaml --context=eks-demo-cluster-dev
ingress.networking.k8s.io/bankapp-ingress-dev created
```

```
[root@192.168.1.101 ~]# cat ingress-rule-dev.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress-dev
  namespace: bankapp
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: bankapp-dev.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: bankapp-service
            port:
              number: 80
```

singhritesh85

```
[root@192.168.1.101 ~]# cat ingress-rule-stage.yaml
# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace bankapp --context=aks-cluster
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress-stage
  namespace: bankapp
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - bankapp-stage.singhritesh85.com
    secretName: ingress-tls
  rules:
  - host: bankapp-stage.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: bankapp-service
            port:
              number: 80
```

```
[root@192.168.1.101 ~]# kubectl get ing -A --watch --context=aks-cluster
NAMESPACE   NAME           CLASS      HOSTS           ADDRESS        PORTS      AGE
bankapp     bankapp-ingress-stage   nginx    bankapp-stage.singhritesh85.com  172.10.1.112  80, 443   1d
vault       vault-ingress       nginx    vault-stage.singhritesh85.com   172.10.1.112  80, 443   1d
[root@192.168.1.101 ~]
[root@192.168.1.101 ~]# kubectl get ing -A --watch --context=eks-demo-cluster-dev
NAMESPACE   NAME           CLASS      HOSTS           ADDRESS        PORTS      AGE
E
argocd      minimal-ingress   nginx    argocd.singhritesh85.com   a[REDACTED]  .us-east-2.elb.amazonaws.com  80   14m
bankapp     bankapp-ingress-dev  nginx   bankapp-dev.singhritesh85.com  a[REDACTED]  .us-east-2.elb.amazonaws.com  80   1s
s
vault       vault-ingress       nginx   vault-dev.singhritesh85.com   a[REDACTED]  .us-east-2.elb.amazonaws.com  80   1s
```

The screenshot shows two separate instances of the Azure DNS Management interface for the domain `singhritesh85.com`.

Left Window (Top):

- Title:** singhritesh85.com | Recordsets
- Section:** DNS zone
- Left sidebar:** Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Settings, DNS Management, Recordsets (selected), DNSSEC, Monitoring, Automation, Help.
- Recordsets Table:**

Name	Type
@	NS
@	SOA
- Bottom:** Add or remove favorites by pressing `Ctrl+Shift+F`

Right Window (Bottom):

- Title:** Add record set
- Domain:** singhritesh85.com
- Form Fields:**
 - Name:** bankapp-stage
 - Type:** A – IPv4 Address records
 - Alias record set:** No
 - TTL:** 1
 - TTL unit:** Hours
 - IP address:** 172.16.112.0.0.0
 - Add:** (button)

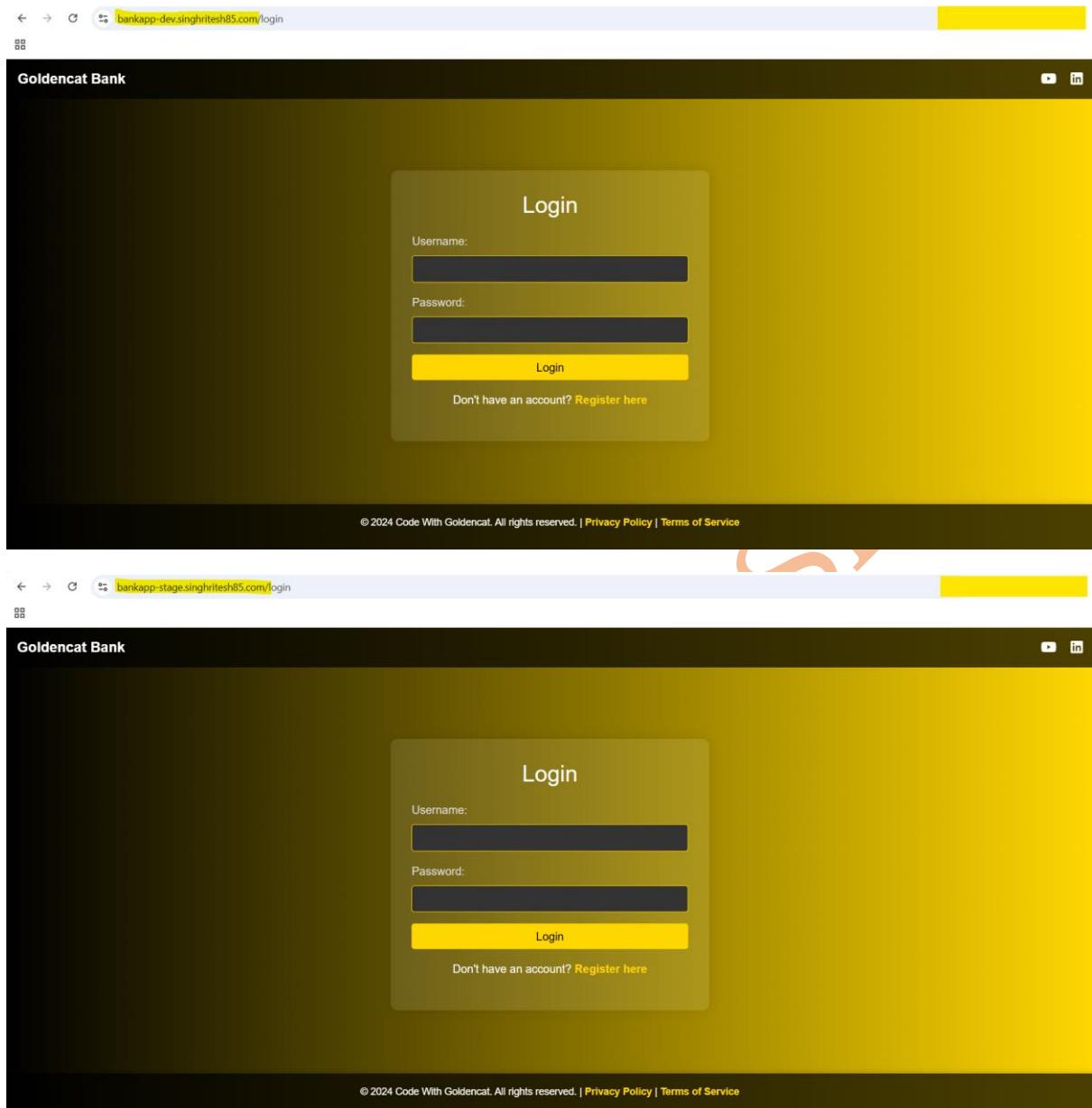
Left Window (Bottom):

- Title:** singhritesh85.com | Recordsets
- Section:** DNS zone
- Left sidebar:** Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Settings, DNS Management, Recordsets (selected), DNSSEC, Monitoring, Automation, Help.
- Recordsets Table:**

Name	Type
@	NS
@	SOA
- Bottom:** Add or remove favorites by pressing `Ctrl+Shift+F`

Right Window (Bottom):

- Title:** Add record set
- Domain:** singhritesh85.com
- Form Fields:**
 - Name:** bankapp-dev
 - Type:** CNAME – Link your subdomain to another record
 - Alias record set:** No
 - TTL:** 1
 - TTL unit:** Hours
 - Alias:** a
 - Add:** (button)

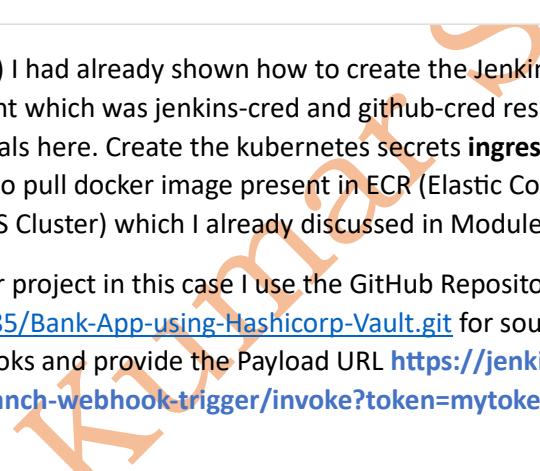


In Manual Approach the disadvantage is for a new Release, manually you need to Run the Jenkins Job then delete the corresponding Application in the ArgoCD then recreate it manually.

Module-2: Automation Approach

In Automation Approach I triggered the Jenkins Job using the webhook and for that I installed the Jenkins Plugin **multibranch scan webhook trigger** and then created a Jenkins credentials of kind secret text **argocd_password** as shown in the screenshot attached below.

The image contains two screenshots of the Jenkins web interface. The top screenshot shows the 'Available plugins' section of the 'Plugins' page. A search bar at the top has 'multibranch scan webhook trigger' typed into it. Below the search bar, there is a table with one row. The row contains a checkbox (which is checked), the name 'Multibranch Scan Webhook Trigger 1.0.11', a description 'Trigger that can receive any HTTP request and trigger a multibranch job scan when token matched.', and two status indicators: 'Released' and '1 yr 5 mo ago'. To the right of the table is a large yellow 'Install' button. The bottom screenshot shows the 'Manage Jenkins' dashboard. On the left, there is a sidebar with various management links like 'New Item', 'Build History', etc. The main area is titled 'Manage Jenkins' and contains several configuration sections: 'System Configuration' (with 'System', 'Tools', 'Clouds', 'Plugins', 'Appearance', and 'Credential Providers' sections), 'Build Queue' (showing 'No builds in the queue'), 'Build Executor Status' (showing 'Built-In Node' and 'Slave-1' with counts of 0/2 each), 'Security' (with a lock icon), and 'Users' (with a user icon). The 'Credentials' section under 'System Configuration' is highlighted with a yellow box.



Jenkins ms.singhritesh85.com/manage/credentials/store/system/domain/_/newCredentials

New credentials

Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret: argocd_password

ID: argocd_password

Description: argocd_password

Create

In Manual Approach (Module-1) I had already shown how to create the Jenkins credentials for jenkins slave and GitHub Account which was jenkins-cred and github-cred respectively. In the similar way create the Jenkins Credentials here. Create the kubernetes secrets **ingress-tls**, **vault-tls-secret** and kubernetes secret **regcred** to pull docker image present in ECR (Elastic Container Registry) **bankapp-2** for stage cluster (AKS Cluster) which I already discussed in Module-1 (Manual Approach).

Then go to Github Repo for your project in this case I use the GitHub Repository <https://github.com/singhritesh85/Bank-App-using-Hashicorp-Vault.git> for source code, and then go to Repository Settings > Webhooks and provide the Payload URL <https://jenkins-ms.singhritesh85.com/multibranch-webhook-trigger/invoke?token=mytoken> as shown in the screenshot attached below.



github.com/singhritesh85/Bank-App-using-Hashicorp-Vault/settings/hooks/new

singhritesh85 / Bank-App-using-Hashicorp-Vault

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Environments

Codepaces

Pages

Security

Advanced Security

Deploy keys

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *: <https://jenkins-ms.singhritesh85.com/multibranch-webhook-trigger/invoke?token=mytoken>

Content type *: application/x-www-form-urlencoded

Secret: (empty field)

SSL verification: Enable SSL verification Disable (not recommended)

Which events would you like to trigger this webhook?: Just the push event.

The screenshot shows the GitHub 'Webhooks' settings page for a repository. The left sidebar has 'Webhooks' selected. The main form is for creating a new webhook:

- Content type ***: application/x-www-form-urlencoded
- Secret**: (empty field)
- SSL verification**: Enable SSL verification Disable (not recommended)
- Which events would you like to trigger this webhook?**
 - Just the push event.
 - Send me everything.
 - Let me select individual events.
- Active**: (We will deliver event details when this hook is triggered.)
- Add webhook** button

Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

✓ <https://jenkins-ms.singhritesh85.co...> (push)

[Edit](#) [Delete](#)

Last delivery was successful.

Then create the multibranch pipeline for mysql and then for bankapp (with webhook) as shown in the screenshot attached below.

The screenshot shows the Jenkins 'New Item' creation dialog. The item name is 'mysql' and the item type is set to 'Multibranch Pipeline'.

Enter an item name: mysql

Select an item type:

- Freestyle project**: Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- Multibranch Pipeline**: Creates a set of Pipeline projects according to detected branches in one SCM repository. This option is highlighted with a yellow border.

OK button



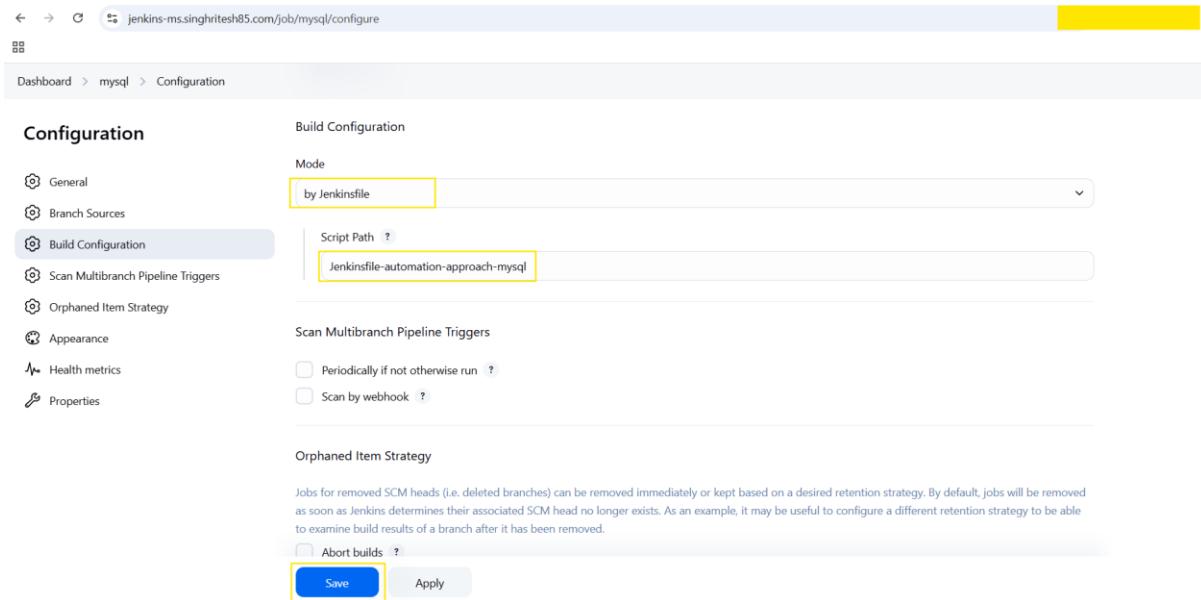
The screenshot shows two consecutive screenshots of the Jenkins job configuration interface for a MySQL pipeline.

Screenshot 1: General Configuration

- General:**
 - Display Name:** mysql
 - Description:** Jenkins multibranch pipeline to create MySQL Pods in dev and stage branch.
 - Enabled:**
- Branch Sources:**
 - Git:** Project Repository: https://github.com/singhritesh85/Bank-App-using-Hashicorp-Vault.git

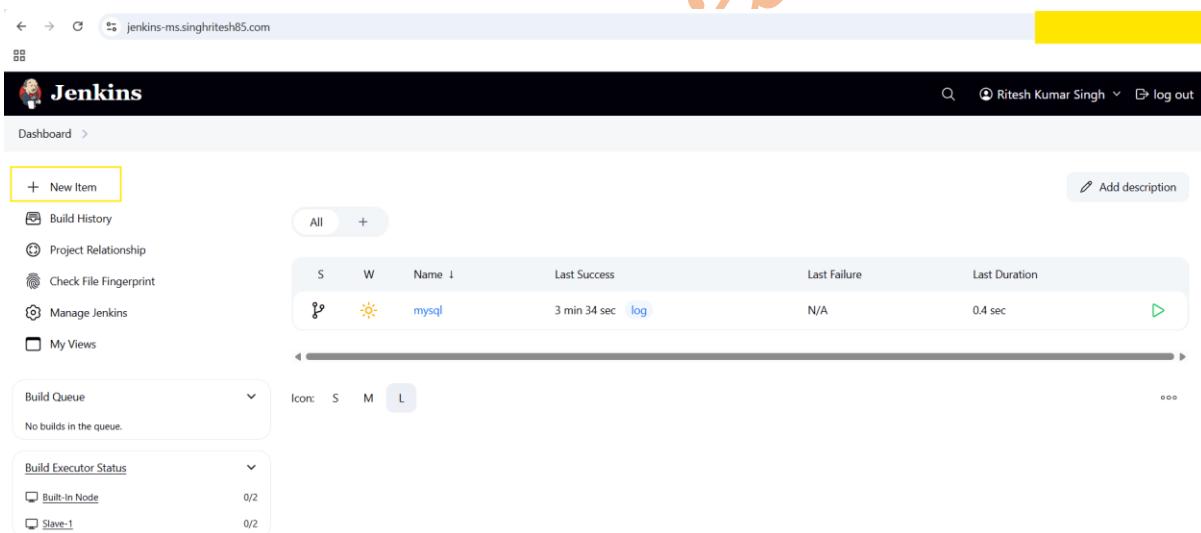
Screenshot 2: Branch Sources Configuration

- Git:** Project Repository: https://github.com/singhritesh85/Bank-App-using-Hashicorp-Vault.git
- Credentials:** [Redacted] /***** (github-cred)
- Behaviors:**
 - Discover branches:** [Redacted]
 - Filter by name (with wildcards):** Include: dev stage
 - Exclude:** [Redacted]



The screenshot shows the Jenkins MySQL Configuration page. On the left, a sidebar lists configuration options: General, Branch Sources, Build Configuration (selected), Scan Multibranch Pipeline Triggers, Orphaned Item Strategy, Appearance, Health metrics, and Properties. The main area is titled 'Build Configuration' with 'Mode' set to 'by Jenkinsfile'. The 'Script Path' is specified as 'Jenkinsfile-automation-approach-mysql'. Under 'Scan Multibranch Pipeline Triggers', there are two checkboxes: 'Periodically if not otherwise run' and 'Scan by webhook'. A note about 'Orphaned Item Strategy' explains that jobs for removed SCM heads can be removed immediately or kept based on a retention strategy. The 'Abort builds' checkbox is also present. At the bottom are 'Save' and 'Apply' buttons.

I did not apply the webhook to deploy the MySQL 8 Pods but I used the webhook to deploy bankapp application pods as shown in the screenshot attached below. Below screenshot shows how I had created the Multibranch Pipeline for Jenkins to create bankapp application pods in EKS and AKS Clusters.



The screenshot shows the Jenkins Dashboard. The left sidebar includes 'New Item', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'My Views'. The main area displays a table of jobs. One job, 'mysql', is listed with the following details: Status (S), Last Success (3 min 34 sec ago), Last Failure (N/A), and Last Duration (0.4 sec). Below the table, sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (Built-in Node 0/2, Slave-1 0/2) are shown. At the bottom right, links for 'REST API' and 'Jenkins 2.504.1' are visible.



The screenshot shows two Jenkins configuration pages. The top part is a 'New Item' creation dialog where 'bankapp' is selected as the item name and 'Multibranch Pipeline' is chosen as the item type. The bottom part is the 'Configuration' screen for the 'bankapp' job, showing the 'General' tab with 'Enabled' checked, 'Display Name' set to 'bankapp', and a description of 'Jenkins multibranch pipeline to create Bank Application Pods in dev and stage branch.' Under the 'Branch Sources' section, a 'Git' repository is listed. Navigation links at the top include 'Dashboard', 'All', and 'New Item'.

Configuration

- General
- Branch Sources**
- Build Configuration
- Scan Multibranch Pipeline Triggers
- Orphaned Item Strategy
- Appearance
- Health metrics
- Properties

Git
Project Repository ?
`https://github.com/singhritesh85/Bank-App-using-Hashicorp-Vault.git`

Credentials ?
`***** (github-cred)`

+ Add

Behaviors

Discover branches ?

Filter by name (with wildcards)
Include ?
`dev stage`

Exclude ?

Save Apply

Configuration

- General
- Branch Sources
- Build Configuration**
- Scan Multibranch Pipeline Triggers
- Orphaned Item Strategy
- Appearance
- Health metrics
- Properties

Build Configuration

Mode
`by Jenkinsfile`

Script Path
`Jenkinsfile-automation-approach-bankapp`

Scan Multibranch Pipeline Triggers

Periodically if not otherwise run ?

Scan by webhook ?
Trigger token ?
`mytoken`

Orphaned Item Strategy

Jobs for removed SCM heads (i.e. deleted branches) can be removed immediately or kept based on a desired retention strategy. By default, jobs will be removed

Save Apply

The screenshot for both the Multibranch Jenkins for MySQL and Bank Application Pods are as shown in the screenshot attached below.

The screenshot shows the Jenkins dashboard with the URL jenkins-ms.singhritesh85.com. The dashboard includes a sidebar with links like 'New Item', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'My Views'. A main table displays build information for 'bankapp' and 'mysql'. Below the table is a 'Build Queue' section indicating 'No builds in the queue'. On the right, there's a 'Build Executor Status' section showing 'Built-in Node' and 'Slave-1' with counts of 0/2 each. The bottom right corner shows 'REST API' and 'Jenkins 2.504.1'.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		bankapp	3 min 32 sec log	N/A	0.44 sec D
		mysql	13 min log	N/A	0.4 sec D

The screenshot shows the Jenkins job details for 'bankapp' with the URL jenkins-ms.singhritesh85.com/job/bankapp/. The left sidebar lists options like 'Status', 'Configure', 'Scan Multibranch Pipeline Now', 'Scan Multibranch Pipeline Log', 'Multibranch Pipeline Events', 'Delete Multibranch Pipeline', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Rename', 'Pipeline Syntax', and 'Credentials'. The main area shows the 'bankapp' job with a status of 'Running'. It has two branches: 'dev' and 'stage', both last updated 4 min 9 sec ago. The bottom section shows a 'Build Queue' with one entry: 'bankapp' with status 'Running'.

The screenshot shows the Jenkins interface for a MySQL pipeline. On the left, a sidebar lists various Jenkins management options like Status, Configure, Scan Multibranch Pipeline Now, and Pipeline Syntax. The main area is titled 'mysql' and displays a summary of the Jenkins multibranch pipeline. It indicates that the pipeline is used to create MySQL Pods in dev and stage branches. Below this, a table shows two branches: 'dev' and 'stage'. The 'dev' branch has a last success at 22 min ago (build #6) and a last failure at 41 min ago (build #3), with a duration of 6.8 sec. The 'stage' branch has a last success at 20 min ago (build #7) and a last failure at 44 min ago (build #2), with a duration of 8.5 sec. A large orange watermark 'Ritesh Kumar' is diagonally across the page.

S	W	Name	Last Success	Last Failure	Last Duration
		dev	22 min #6	41 min #3	6.8 sec
		stage	20 min #7	44 min #2	8.5 sec

Then create the ingress rule and do the entry for LoadBalancer DNS Name and LoadBalancer External IP in Azure DNS Zone to create the Record Set of type CNAME and A Type for EKS and AKS Cluster respectively as I discussed in Module-1.

```
cat ingress-rule-dev.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress-dev
  namespace: bankapp
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: bankapp-dev.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
      backend:
        service:
          name: bankapp-service
        port:
          number: 80
```

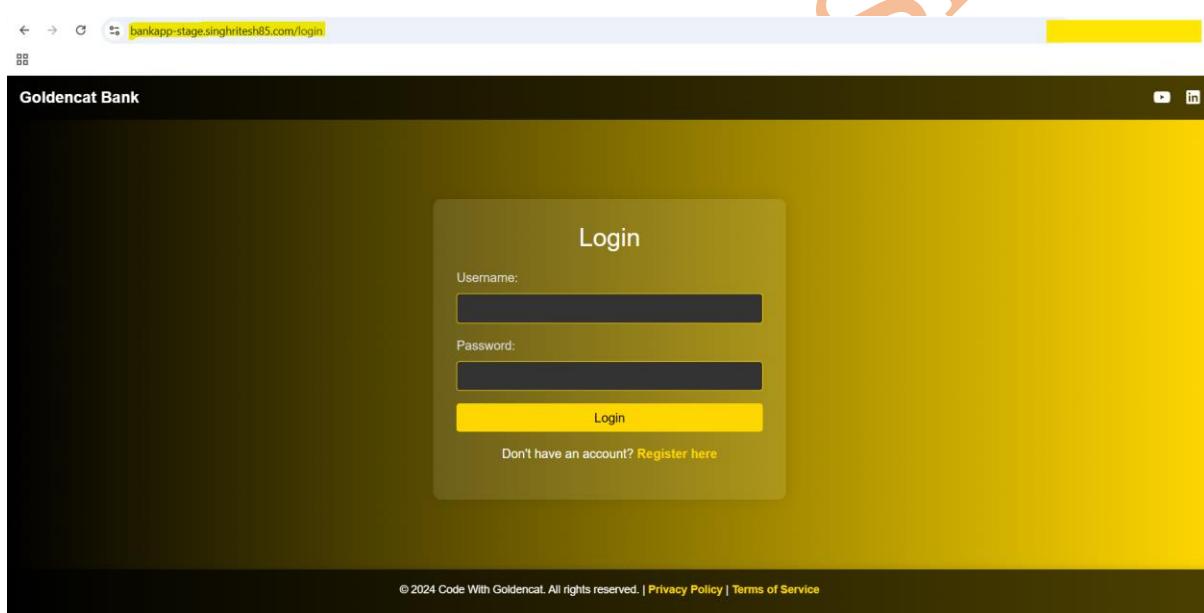
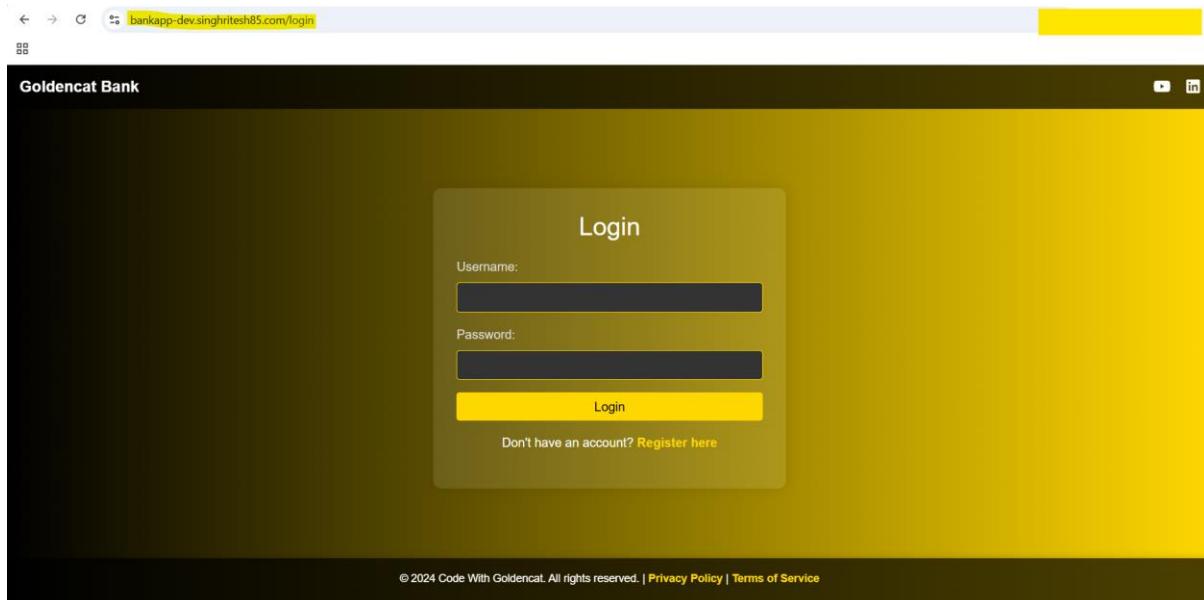
```
cat ingress-rule-stage.yaml
```

```
# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace bankapp --context=aks-cluster

---

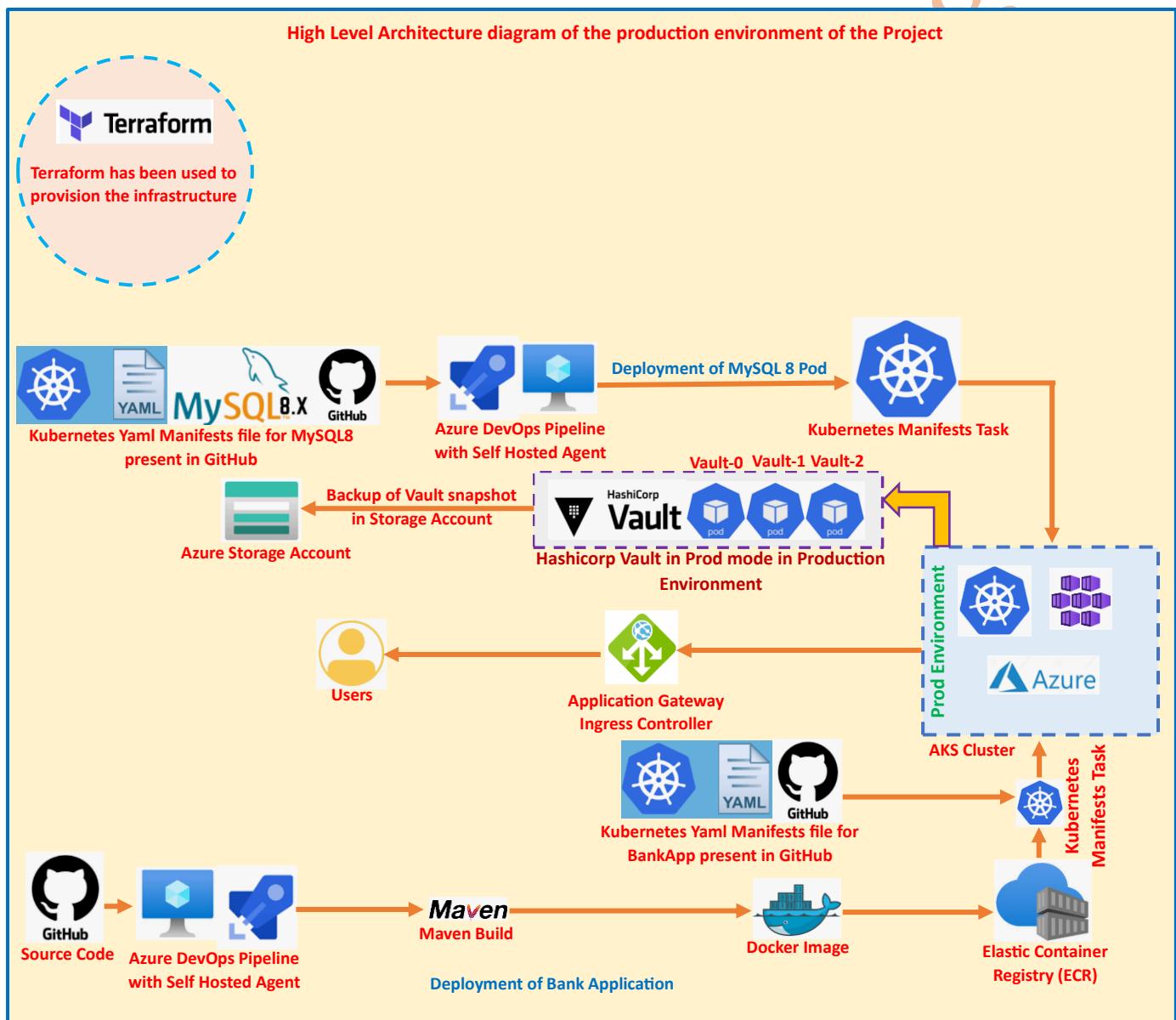
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress-stage
  namespace: bankapp
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - bankapp-stage.singhritesh85.com
    secretName: ingress-tls
  rules:
  - host: bankapp-stage.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
      backend:
        service:
          name: bankapp-service
        port:
          number: 80
```

Finally, I was able to access the Bank Application as shown in the screenshot attached below.



Module-3: [Production Environment]

The production environment completely existed in the Azure and I used Azure DevOps Pipeline as CI/CD Tool for this project. I had not discussed here about the Integration of SonarQube, Artifactory and Trivy Image Scan. If you are interested in the same the please refer the project present in GitHub Repository <https://github.com/singhritesh85/DevOps-Project-BankApplication-Multibranch-MultiCloud.git>, there I also discussed about Monitoring using Prometheus and Grafana, Log Aggregation using Loki and GitHub Branching Strategies. In **Module-3** of this project I discussed mainly about the Deployment in Production environment and storage of MySQL credentials in Hashicorp vault. The Hashicorp Vault Backup I took using the Cron Job which I will be discussed later in this module.



I had created the Azure Resources first using the Terraform script present in the GitHub Repo <https://github.com/singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud.git> at the path **terraform-bankapp-multibranch-multicloud-prod**. Before running the terraform script I ran the shell script present in the directory **terraform-bankapp-multibranch-multicloud-prod** as shown in the screenshot attached below. This shell script will install the aws cli, kubectl and helm.

```
[root@yellow terraform-bankapp-multibranch-multicloud-prod]# ./initial-setup.sh
```

Then I logged-out and login again and after that I installed Azure CLI and authenticated and authorize the user as shown in the screenshot attached below.

```
[root@yellow main]# yum install -y https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm
[root@yellow main]# yum install azure-cli -y
[root@yellow main]# az login
To sign in, use a web browser to open the page [REDACTED] and enter the code [REDACTED] to authenticate.
```

Then you can run the below commands

terraform init -----> initializes a working directory containing configuration files and installs plugins for required providers.

terraform validate -----> verify that terraform configuration file is correct or not

terraform plan -----> Check which resources are going to be created.

Then you can run the command **terraform apply -auto-approve** -----> Finally, Create the resources.

```
module.aks.null_resource.kubectl: Creation complete after 3s [id=yellow]
Apply complete! Resources: 33 added, 0 changed, 0 destroyed.

Outputs:

acr_azure_loki_vm_private_ip_container_sas_url = <sensitive>
[root@yellow main]#
[root@yellow main]# terraform output -json acr_azure_loki_vm_private_ip_container_sas_url
{"acr_login_server": "bankappprodcontainer24registry.azurecr.io", "azure_devopsagent_vm": "10.0.0.4", "sas_url_query_string": "https://vaultbackup24prod.blob.core.windows.net/dexter?sv="}
```

After execution of the Terraform script, you can get the Blob SAS URL using the command **terraform output -json acr_azure_loki_vm_private_ip_container_sas_url** as shown in the screenshot attached above. After creation of the resources a kubeconfig file was generated.

I had installed Hashicorp Vault Prod in AKS Cluster using the commands as shown in the screenshot attached below.

```
helm repo add hashicorp https://helm.releases.hashicorp.com
helm repo update
helm install vault hashicorp/vault --namespace vault --create-namespace --set
server.image.tag=1.19.0,server.ha.enabled=true,server.ha.raft.enabled=true,server.ha.replicas=3,server.dev.enabled=false,server.ui.enabled=true,server.ui.serviceType=ClusterIP
```

```
[root@[REDACTED] ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "hashicorp" chart repository
Update Complete. ⚡Happy Helming!⚡
[root@[REDACTED] ~]# helm install vault hashicorp/vault --namespace vault --create-namespace --set server.image.tag=1.19.0,server.ha.enabled=true,server.ha.raft.enabled=true,server.ha.replicas=3,server.dev.enabled=false,server.ui.enabled=true,server.ui.serviceType=ClusterIP
NAME: vault
LAST DEPLOYED: Wed May 21 08:49:54 2025
NAMESPACE: vault
STATUS: deployed
REVISION: 1
NOTES:
Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using
Vault with Kubernetes available here:

https://developer.hashicorp.com/vault/docs

Your release is named vault. To learn more about the release, try:

$ helm status vault
$ helm get manifest vault

[root@[REDACTED] ~]# kubectl get pods -n vault
NAME                  READY   STATUS    RESTARTS   AGE
vault-0               0/1     Running   0          63s
vault-1               0/1     Running   0          62s
vault-2               0/1     Running   0          62s
vault-agent-injector-[REDACTED] 1/1     Running   0          63s
```

Now logged-in to the vault pods and executed the below commands.

```
=====  
on pod vault-0  
=====
```

```
kubectl exec -it vault-0 -n vault sh
```

```
$ vault operator init
```

```
----- Key1
```

```
----- Key2
```

```
----- Key3
```

```
----- Key4
```

```
----- Key5
```

```
----- Root Token
```

```
$ vault operator unseal -----> First Time
```

```
Unseal Key:
```

```
$ vault operator unseal -----> Second Time
```

```
Unseal Key:
```

```
$ vault operator unseal -----> Third Time
```

```
Unseal Key:
```

```
$ vault login -----> Should run only on vault-0
```

```
Token:
```

=====

Run the below commands as written below

=====

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

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

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

=====

on pod vault-1

=====

kubectl exec -it vault-1 -n vault sh

\$ vault operator unseal -----> First Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Second Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Third Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

=====

on pod vault-2

=====

kubectl exec -it vault-2 -n vault sh

\$ vault operator unseal -----> First Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Second Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

\$ vault operator unseal -----> Third Time

Unseal Key: -----> Unseal Key should be one which you obtained from vault-0

=====

Finally, Run the below command to check the leader and follower of Hashicorp Vault

=====

kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers -----> Here you will see that vault-0 will be leader and vault-1 and vault-2 pod will be as forwarder

=====

Now check the status of three vault pods

=====

kubectl get pods -n vault

NAME	READY	STATUS	RESTARTS	AGE
vault-0	1/1	Running	0	3h46m
vault-1	1/1	Running	0	3h46m
vault-2	1/1	Running	0	3h46m
vault-agent-injector-7XXXXXXXXXb-mXXXv	1/1	Running	0	3h46m

The Pods will be in the running status and READY 1/1

```
[root@] ~]# kubectl exec -it vault-0 -n vault
error: you must specify at least one command for the container
[root@Terraform-Server ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault operator init
Unseal Key 1: [REDACTED]
Unseal Key 2: [REDACTED]
Unseal Key 3: [REDACTED]
Unseal Key 4: [REDACTED]
Unseal Key 5: [REDACTED]

Initial Root Token: h[REDACTED]A

Vault initialized with 5 key shares and a key threshold of 3. Please securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 3 of these keys to unseal it
before it can start servicing requests.

Vault does not store the generated root key. Without at least 3 keys to
reconstruct the root key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
existing unseal keys shares. See "vault operator rekey" for more information.
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
Unseal Nonce [REDACTED]
Version      1.19.0
Build Date   2025-01-10T14:45:00Z
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

Ritesh Kumar Singh

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key                      Value
---
Seal Type                shamir
Initialized              true
Sealed                  true
Total Shares             5
Threshold               3
Unseal Progress          2/3
Unseal Nonce             [REDACTED]
Version                 1.19.0
Build Date               2025 [REDACTED]
Storage Type             raft
Removed From Cluster    false
HA Enabled               true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key                      Value
---
Seal Type                shamir
Initialized              true
Sealed                  false
Total Shares             5
Threshold               3
Version                 1.19.0
Build Date               2025 [REDACTED]
Storage Type             raft
Cluster Name             vault-cluster-[REDACTED]
Cluster ID               [REDACTED]
Removed From Cluster    false
HA Enabled               true
HA Cluster               https://vault-0.vault-internal:8201
HA Mode                  active
Active Since             2025 [REDACTED]
Raft Committed Index    37
Raft Applied Index       37
/ $
```

```

/ $ vault login
Token (will be hidden):
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key          Value
---          -----
token        h[REDACTED]A
token_accessor [REDACTED]
token_duration   ∞
token_renewable  false
token_policies   ["root"]
identity_policies []
policies       ["root"]
/ $

[root@[REDACTED] ~]# kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers
Node           Address      State  Voter
---           -----
[REDACTED]     vault-0.vault-internal:8201  leader  true
[root@[REDACTED] ~]# kubectl exec -ti vault-1 -n vault -- vault operator raft join http://vault-0.vault-internal:8200
Key          Value
---          -----
Joined       true
[root@[REDACTED] ~]# kubectl exec -ti vault-2 -n vault -- vault operator raft join http://vault-0.vault-internal:8200
Key          Value
---          -----
Joined       true
[root@[REDACTED] ~]# 

[root@[REDACTED] ~]# kubectl exec -it vault-1 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized  true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
Unseal Nonce [REDACTED]
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized  true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 2/3
Unseal Nonce [REDACTED]
Version      1.19.0
Build Date   2025[REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true

```

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 0/3
UnsealNonce  n/a
Version      1.19.0
Build Date   2025 [REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $
```

```
[root@ [REDACTED] ~]# kubectl exec -it vault-2 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 1/3
UnsealNonce  [REDACTED]
Version      1.19.0
Build Date   2025 [REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 2/3
UnsealNonce  [REDACTED]
Version      1.19.0
Build Date   2025 [REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
```

K
Singh

```
/ $ vault operator unseal
Unseal Key (will be hidden):
Key          Value
---          -----
Seal Type    shamir
Initialized   true
Sealed       true
Total Shares 5
Threshold    3
Unseal Progress 0/3
UnsealNonce  n/a
Version      1.19.0
Build Date   2025 [REDACTED]
Storage Type raft
Removed From Cluster false
HA Enabled   true
/ $
```

```
[root@ [REDACTED] ~]# kubectl -n vault exec -ti vault-0 -- vault operator raft list-peers
Node          Address        State  Voter
---          -----        ----  ----
[REDACTED]    vault-0.vault-internal:8201  leader  true
[REDACTED]    vault-1.vault-internal:8201  follower  true
[REDACTED]    vault-2.vault-internal:8201  follower  true
```

Then I created the Ingress Rule and finally a Record Set of A Type in Azure DNS Zone to access the vault in prod mode in production cluster.

```
[root@ [REDACTED] ~]# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhrites85_com.crt --namespace vault
secret/ingress-tls created

[root@ [REDACTED] ~]# kubectl apply -f vault-ingress-rule-prod.yaml
ingress.networking.k8s.io/vault-ingress created
[root@ [REDACTED] ~]# cat vault-ingress-rule-prod.yaml
# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhrites85_com.crt --namespace vault
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vault-ingress
  namespace: vault
  annotations:
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: azure-application-gateway
  tls:
  - secretName: ingress-tls
  rules:
  - host: vault.singhrites85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: vault-active
            port:
              number: 8200

[root@ [REDACTED] ~]# kubectl get ing -A --watch
NAMESPACE  NAME          CLASS           HOSTS          ADDRESS        PORTS     AGE
vault      vault-ingress  azure-application-gateway  vault.singhrites85.com  4. [REDACTED].25  80, 443  13s
```

```
cat vault-ingress-rule-prod.yaml

# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace
vault

---

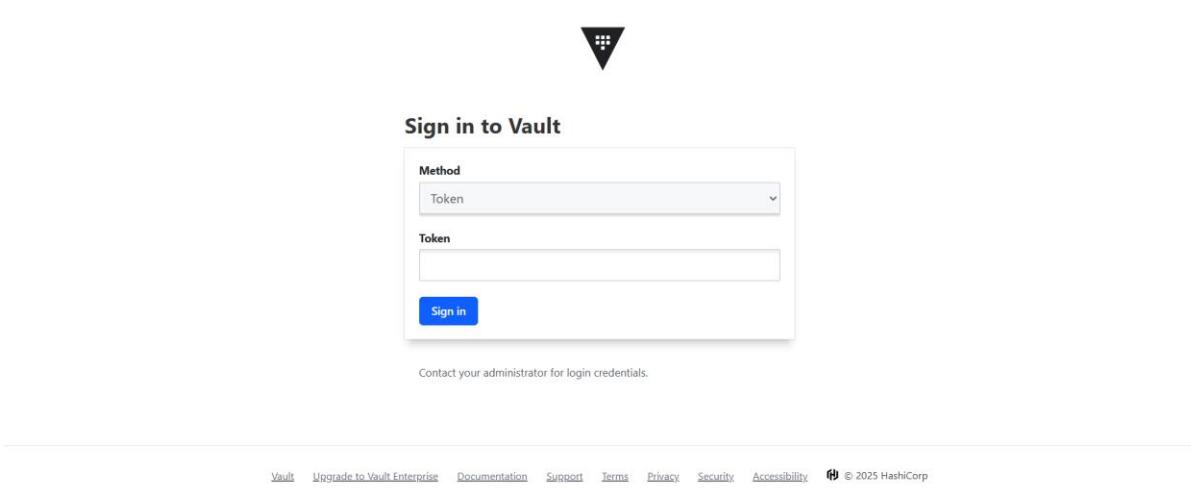
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: vault-ingress
  namespace: vault
  annotations:
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: azure-application-gateway
  tls:
    - secretName: ingress-tls
  rules:
    - host: vault.singhritesh85.com
      http:
        paths:
          - path: /
            pathType: Prefix
        backend:
          service:
            name: vault-active
            port:
              number: 8200
```

The screenshot shows the Azure DNS Management interface. On the left, a sidebar lists various services: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource visualizer, Settings, DNS Management, Recordsets (which is selected and highlighted in grey), DNSSEC, Monitoring, Automation, and Help. The main content area shows a search bar and a message stating "Fetched 2 record set(s)". Below this, there is a table with two rows:

Name	Type
@	NS
@	SOA

To the right, a modal window titled "Add record set" is open. It contains fields for Name (set to "vault"), Type (set to "A - IPv4 Address records"), Alias record set (set to "No"), TTL (set to "1"), TTL unit (set to "Hours"), IP address (containing "4.128.25"), and a placeholder for "0.0.0.0". At the bottom of the modal are "Add", "Cancel", and "Give feedback" buttons. A large orange hand-drawn annotation points from the text below to the "Add" button.

Finally, I was able to access the Hashicorp vault in prod mode in production cluster (Private AKS Cluster) as shown in the screenshot attached below.



For the first time login method into the vault is Token. The Token which I got from the vault-0 pod in production cluster was used to login into the Hashicorp vault prod mode in production cluster. Then I enabled the vault auth userpass and created a username with password with Administrator privileges which I will use to login in to the Hashicorp Vault prod mode in production cluster (AKS Cluster) as shown in the screenshot attached below.

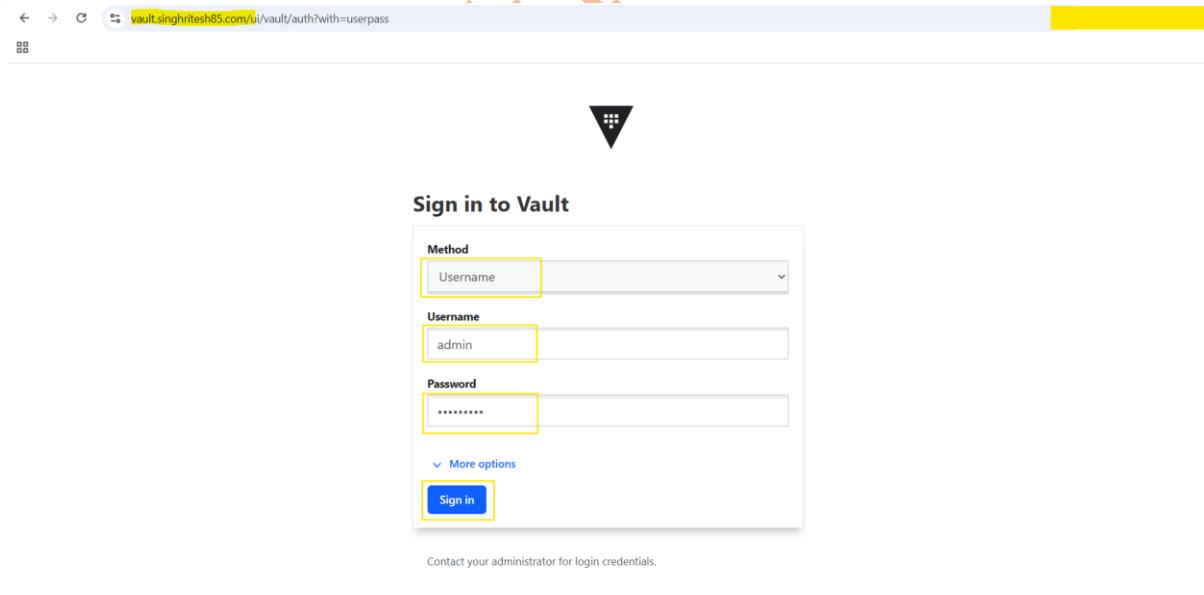
```
$ vault auth enable userpass
```

```
$ vault policy write admin -<<EOF
path "*" {
  capabilities = ["create", "read", "update", "delete", "list", "sudo"]
}
EOF
```

```
$ vault write auth/userpass/users/admin password=Admin@123 policies=admin
```

```
[root@ip-172-31-10-10 ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault auth enable userpass
Success! Enabled userpass auth method at: userpass/
/ $ vault policy write admin -<<EOF
> path "*" {
>   capabilities = ["create", "read", "update", "delete", "list", "sudo"]
> }
> EOF
Success! Uploaded policy: admin
/ $ vault write auth/userpass/users/admin password=Admin@123 policies=admin
Success! Data written to: auth/userpass/users/admin
/ $
```

Then I logged-in with the created username and password as shown in the screenshot attached below.



Then created the Hashicorp vault secrets in production cluster (AKS Cluster) as shown in the screenshot attached below.

```
[root@ ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault secrets enable -path=bankapp kv-v2
Success! Enabled the kv-v2 secrets engine at: bankapp/
/ $ vault kv put bankapp/database/config password=Dexter@123
===== Secret Path =====
bankapp/data/database/config

===== Metadata =====
Key      Value
---      ---
created_time 2025-05-21T03:20:45Z
custom_metadata <nil>
deletion_time n/a
destroyed     false
version       1
/ $ vault kv get bankapp/database/config
===== Secret Path =====
bankapp/data/database/config

===== Metadata =====
Key      Value
---      ---
created_time 2025-05-21T03:20:45Z
custom_metadata <nil>
deletion_time n/a
destroyed     false
version       1

===== Data =====
Key      Value
---      ---
password  Dexter@123
/ $
```

You can verify the same secrets in your production cluster (AKS Cluster) Hashicorp Vault UI as shown in the screenshot attached below.

The screenshot shows the Hashicorp Vault UI interface. On the left is a dark sidebar menu with options like 'Vault', 'Dashboard', 'Secrets Engines', 'Access', 'Policies', 'Tools', 'Monitoring', 'Raft Storage', 'Client Count', and 'Seal Vault'. The 'Secrets Engines' section is currently selected. The main content area has a breadcrumb navigation path: 'Secrets / bankapp / database / config'. Below this, there's a sub-navigation bar with tabs: 'Overview', 'Secret' (which is highlighted), 'Metadata', 'Paths', and 'Version History'. Underneath is a table with two columns: 'Key' and 'Value'. A single row is visible, showing 'password' as the key and 'Dexter@123' as the value. To the right of the table, it says 'Version 1 created May 21, 2025 03:20 PM'. At the bottom of the page, there are links for 'Vault 1.19.0', 'Upgrade to Vault Enterprise', 'Documentation', 'Support', 'Terms', 'Privacy', 'Security', 'Accessibility', and a copyright notice '© 2025 HashiCorp'.

Create and apply the policy in vault to read secrets from Hashicorp Vault in Private AKS Cluster

Next, I had created a policy which allowed reading secrets. This policy was attached to a role, which could be used to grant access to specific Kubernetes service accounts within a namespace. This namespace should be the same in which the pod exists which tries to get secrets from vault.

```
vault policy write bankapp-policy -<<EOF
path "bankapp/data/database/config" {
  capabilities = ["read"]
}
EOF
```

```
[root@XXXXXXXXXX ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault policy write bankapp-policy -<<EOF
> path "bankapp/data/database/config" {
>   capabilities = ["read"]
>
> EOF
Success! Uploaded policy: bankapp-policy
/ $ vault policy list
admin
bankapp-policy
default
root
/ $ vault policy read bankapp-policy
path "bankapp/data/database/config" {
  capabilities = ["read"]
}
/ $
```

vault policy list

vault policy read <policy-name>

Enable the Kubernetes authentication method in Vault in EKS and AKS Cluster as shown in the screenshot attached below.

Vault auth enable kubernetes

```
[root@XXXXXXXXXX ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault auth enable kubernetes
Success! Enabled kubernetes auth method at: kubernetes/
/ $
```

Configure Vault to communicate with the Kubernetes API server on EKS and AKS Cluster as shown in the screenshot attached below.

```
vault write auth/kubernetes/config \
  token_reviewer_jwt="$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \
  kubernetes_host=https:// ${KUBERNETES_PORT_443_TCP_ADDR}:443 \
  kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
```

```
[root@XXXXXXXXXX ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault write auth/kubernetes/config \
>   token_reviewer_jwt="$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" \
>   kubernetes_host=https:// ${KUBERNETES_PORT_443_TCP_ADDR}:443 \
>   kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
Success! Data written to: auth/kubernetes/config
/ $
```

Then I created a role named as bankapp that binds the policy bankapp-policy to a Kubernetes service account(bankapp-sa) in a specific namespace mysql. This allows the service account to access secrets stored in Hashicorp Vault

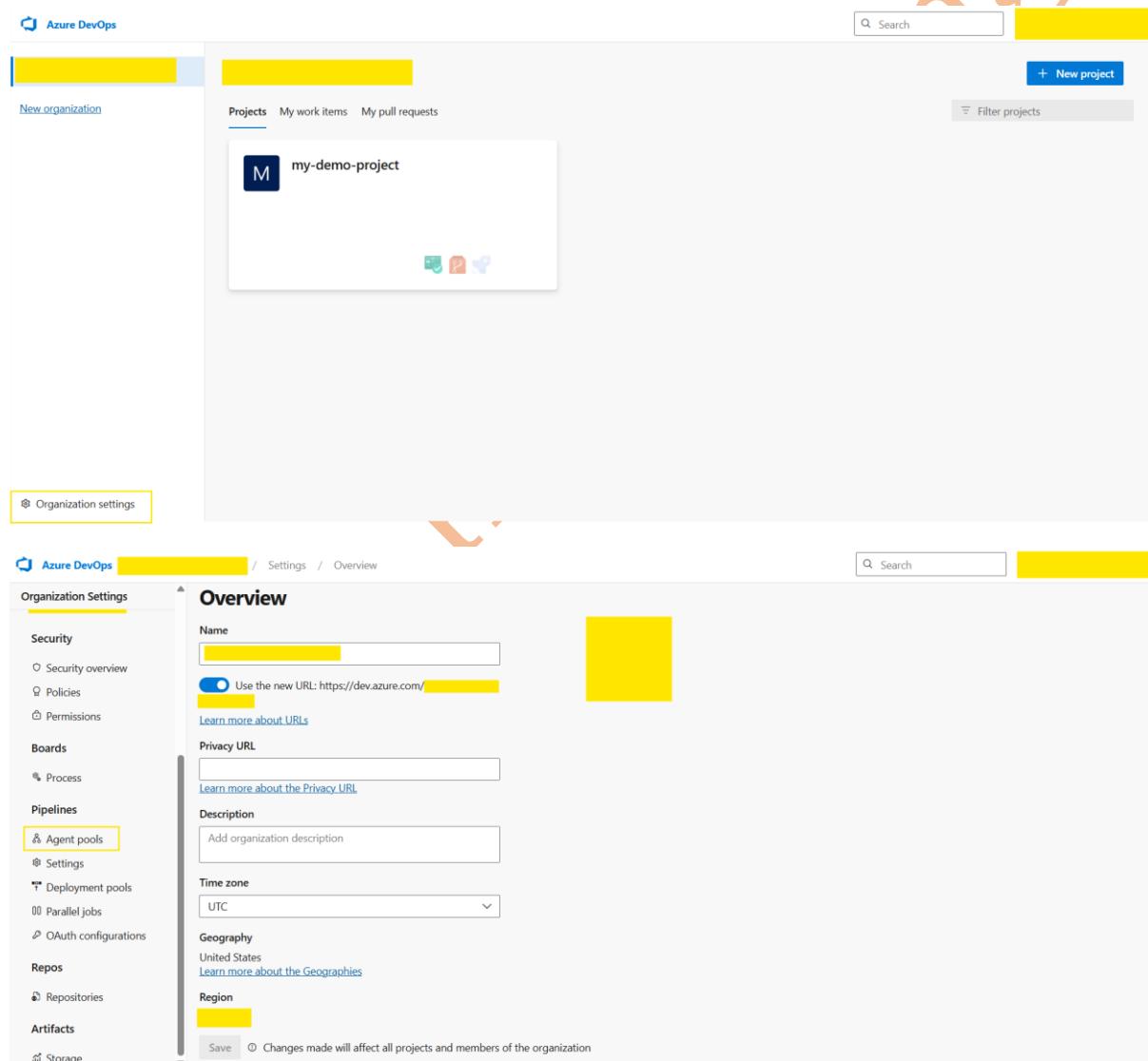
```
vault write auth/kubernetes/role/bankapp bound_service_account_names=bankapp-sa
bound_service_account_namespaces=mysql policies=bankapp-policy ttl=1h
```

```
[root@... ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault write auth/kubernetes/role/bankapp bound_service_account_names=bankapp-sa bound_service_account_namespaces=mysql policies=bankapp-policy ttl=1h
Success! Data written to: auth/kubernetes/role/bankapp
/ $
```

Installation of Azure DevOps Agent of Azure DevOps Agent Azure VM

In production for CI/CD pipeline I used Azure DevOps Pipeline. For Azure DevOps Pipeline I had used Self-hosted-Agent and followed the below procedure to install it.

To install the self-hosted agent for Azure DevOps pipeline first I opened the Azure DevOps UI and went to Organisations Settings > Agent Pools > Add Pool.



The screenshot shows the Azure DevOps interface. At the top, there's a search bar and a 'New project' button. Below that, a navigation bar has 'Projects' selected. A 'my-demo-project' card is visible. On the left, a sidebar lists organization settings like Security, Boards, Pipelines (with 'Agent pools' highlighted), Repos, Artifacts, and Storage. The main area is titled 'Organization settings' and shows an 'Overview' section with fields for Name, Privacy URL, Description, Time zone, Geography (United States), and Region. A note at the bottom says 'Changes made will affect all projects and members of the organization'. There are 'Save' and 'Cancel' buttons at the bottom.

Agent pools

Name	Queued jobs	Running jobs
Azure Pipelines		
Default		

Add agent pool

Agent pools are shared across an organization.

Managed DevOps Pool
Reduce the effort spent in maintaining custom agents by creating a Microsoft managed pool of scalable agents. [Learn more](#).

Self-hosted
Create a pool of custom agents hosted on your own infrastructure for maximum control and flexibility. [Learn more](#).

Azure virtual machine scale set
Create a pool of custom agents based on an Azure Virtual machine scale set hosted in your own Azure subscription. [View configuration instructions](#).

Name:

Description (optional):

Pipeline permissions:

Auto-provision this agent pool in all projects

Create

New Agent Pool had been added as shown in the screenshot attached below.

Agent pools

Name	Queued jobs	Running jobs
Azure Pipelines		
Default		
demo		

Then added the agent as shown in the screenshot attached below.

demo

Jobs Agents Details Security Settings Maintenance History Analytics

Update all agents **New agent**

No jobs have run on this agent pool

Run a pipeline on this agent pool to see more details

For Azure DevOps Pipeline I had used Self-hosted-Agent and followed the below procedure to install it.

Ritesh Kumar Singh || Email Address: - riteshkumarsingh9559@gmail.com || LinkedIn: - <https://www.linkedin.com/in/ritesh-kumar-singh-41113128b/> || GitHub: - <https://github.com/singhritesh85>

```
[root@devopsagent-vm ~]# cd /opt && mkdir myagent && cd myagent
[root@devopsagent-vm myagent]# wget https://vstsagentpackage.azureedge.net/agent/XXXXXX/vsts-agent-linux-x64-XXXXXX.tar.gz
[root@devopsagent-vm myagent]# tar -xvf vsts-agent-linux-x64-XXXXXX.tar.gz
[root@devopsagent-vm myagent]# rm -f vsts-agent-linux-x64-XXXXXX.tar.gz
[root@devopsagent-vm myagent]# ./bin/installdependencies.sh
[demo@devopsagent-vm myagent]$ sudo chown -R demo:demo /opt/myagent/
[demo@devopsagent-vm myagent]$ ./config.sh

Azure Pipelines
agent [XXXXXX] [XXXXXX]

>> End User License Agreements:
Building sources from a TFVC repository requires accepting the Team Explorer Everywhere End User License Agreement. This step is not required for building sources from Git repositories.

A copy of the Team Explorer Everywhere license agreement can be found at:
/opt/myagent/license.html

Enter (Y/N) Accept the Team Explorer Everywhere license agreement now? (press enter for N) > Y
>> Connect:
Enter server URL > [XXXXXX]
Enter authentication type (press enter for PAT) >
Enter personal access token > [XXXXXX]
Connecting to server ...
>> Register Agent:
Enter agent pool (press enter for default) > demo
Enter agent name (press enter for devopsagent-vm) > demo
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2025 [XXXXXX] Settings Saved.

[demo@devopsagent-vm myagent]$ ./env.sh
[demo@devopsagent-vm myagent]$
[demo@devopsagent-vm myagent]$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin:/opt/sonar-scanner/bin:/opt/apache-maven/bin:/opt/node-v16.0.0/bin:/opt/dependency-check/bin:/usr/local/bin
```

[demo@devopsagent-vm myagent]\$ sudo ./svc.sh install

[demo@devopsagent-vm myagent]\$ sudo ./svc.sh start

Now the Agent came in online state as shown in the screenshot attached below.

The screenshot shows the 'Agents' tab selected in the Azure DevOps interface. A table lists one agent named 'demo'. The 'Current status' column shows 'Idle' and the 'Agent version' column shows '2025'. The 'Enabled' switch is turned 'On'. There are buttons for 'Update all agents' and 'New agent'.

Name	Last run	Current status	Agent version	Enabled
demo ● Online		Idle	2025	<input checked="" type="checkbox"/> On

Then I went into the **Azure DevOps project > Project Settings** and created the **Service connections** for GitHub Account, Docker Registry as shown in the screenshot attached blow.

About this project

Help others to get on board!
Describe your project and make it easier for other people to understand it.

+ Add Project Description

Service connections

Convert your existing Azure Resource Manager service connections which use secrets to authenticate to leverage Workload identity federation instead, for improved security and simplified maintenance.

New service connection

New GitHub service connection

Authentication method: Grant authorization Personal Access Token

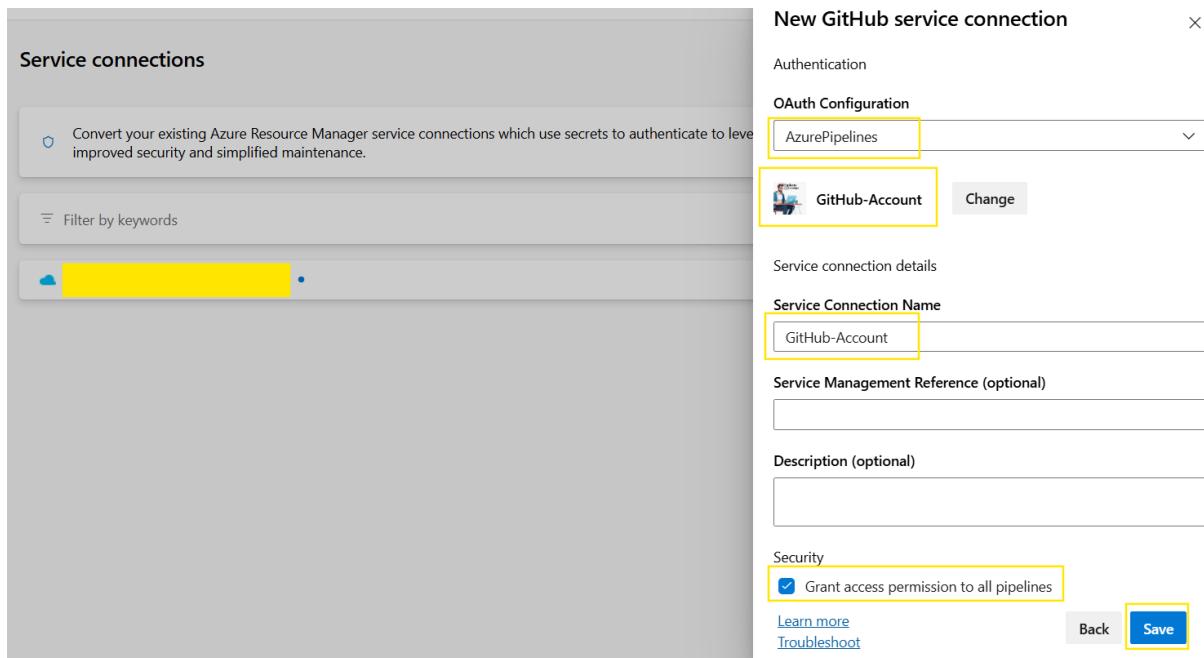
OAuth Configuration: AzurePipelines

Authorize

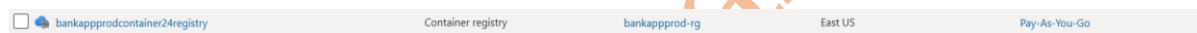
Service connection details:

- Service Connection Name: GitHub-Account
- Service Management Reference (optional):
- Description (optional):

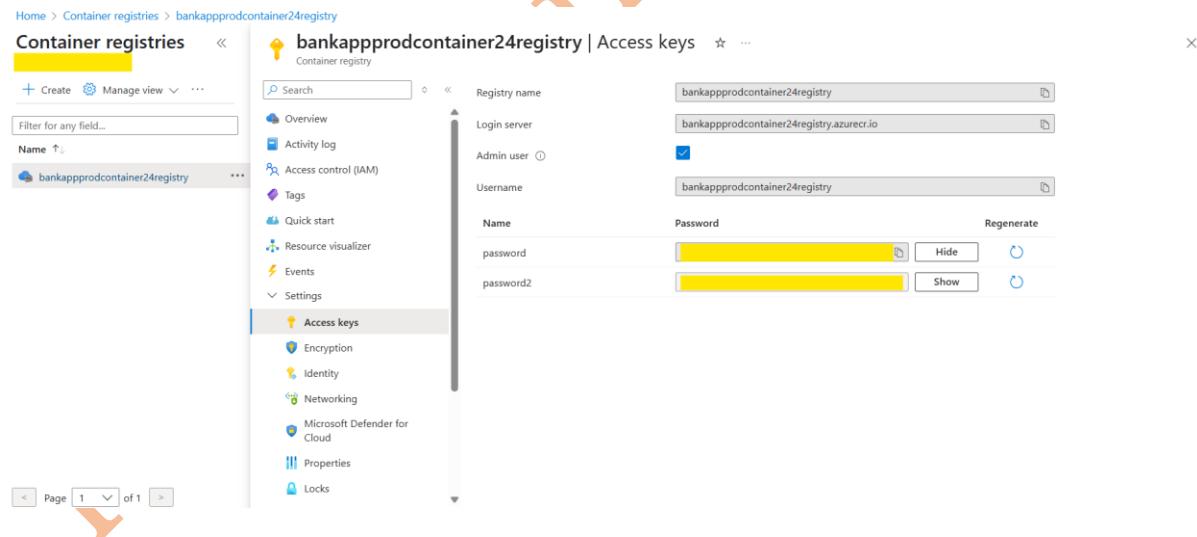
Authorize your GitHub Account

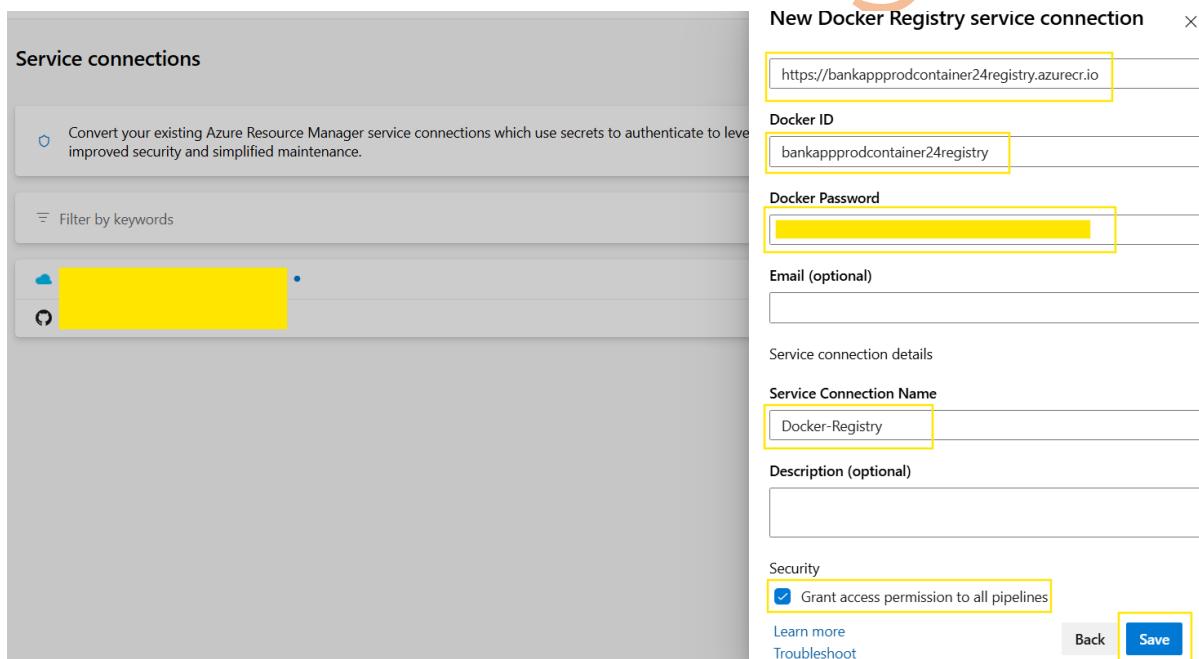
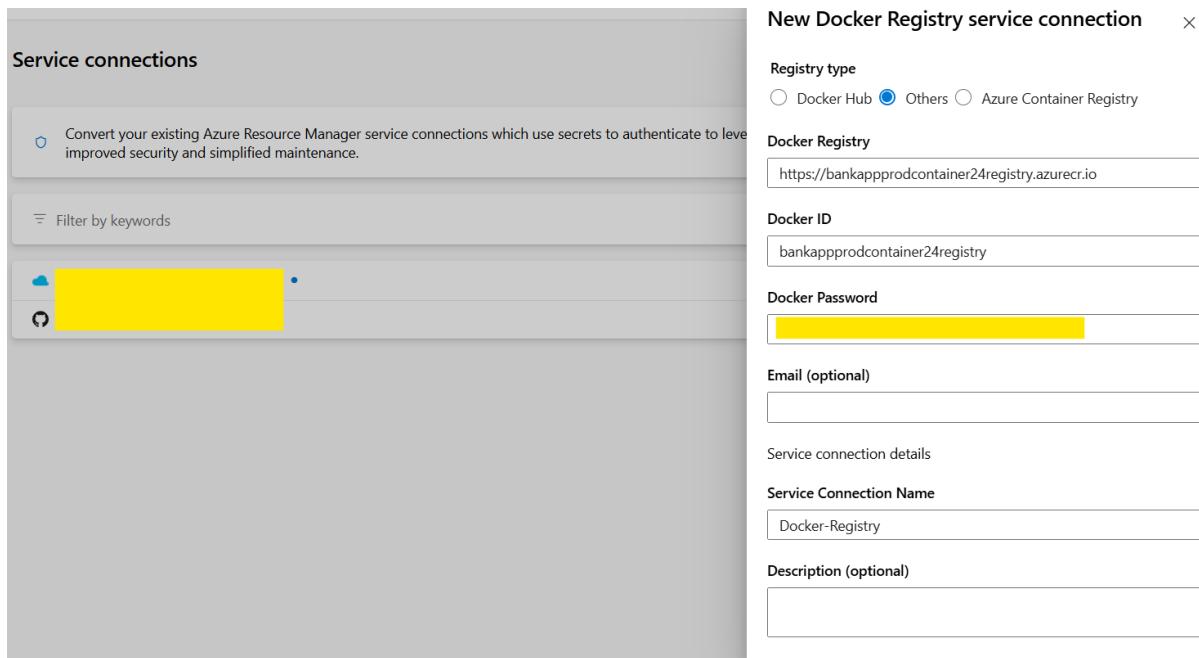


The Azure Container Registry which I had created for this project is as shown in the screenshot attached below.

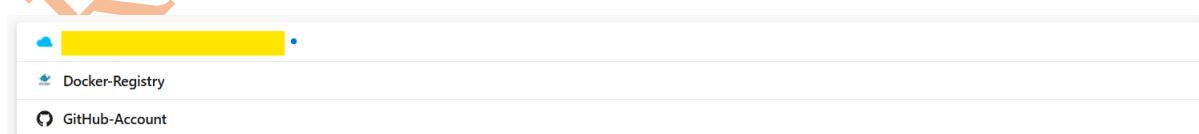


Then I had created the Service Connection for Docker Registry as shown in the screenshot attached below.





Finally the service connections had been created as shown in the screenshot attached below.



Then I created the Azure DevOps CI/CD pipeline as shown in the screenshot attached below.

Create your first Pipeline

Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.

Create Pipeline

Where is your code?

- Azure Repos Git (YAML)
- Bitbucket Cloud (YAML)
- GitHub (YAML)** Home to the world's largest community of developers
- Github Enterprise Server (YAML) The self-hosted version of GitHub Enterprise
- Other Git Any generic Git repository
- Subversion Centralized version control by Apache

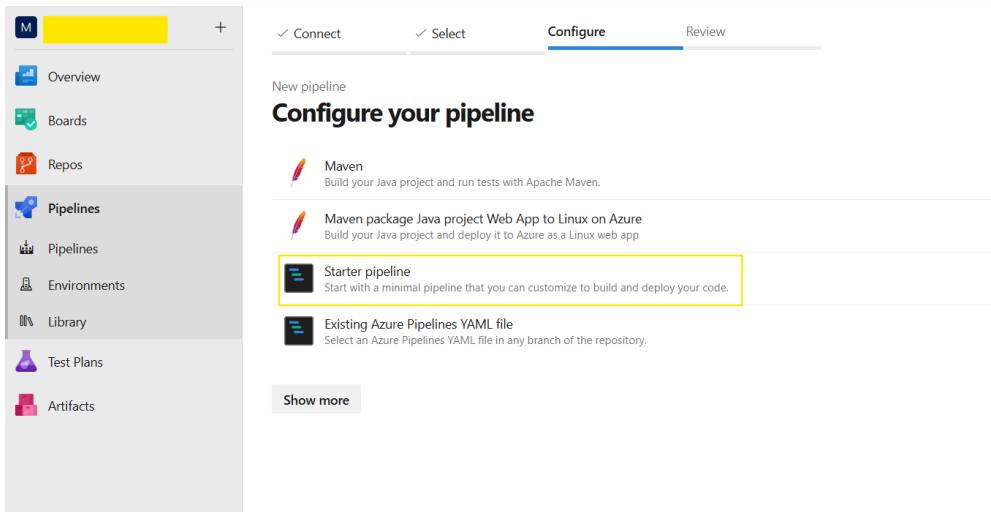
Use the [classic editor](#) to create a pipeline without YAML.

At this stage it will ask to authorize your GitHub Account and after authorizing your GitHub Account select the desired GitHub Repo as shown in the screenshot attached below.

Select a repository

Filter by keywords My repositories

- singhritesh85/kubernetes-manifests** Just now
- singhritesh85/Bank-App-using-Hashicorp-Vault 1h ago
- singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-MultiCloud 2h ago
- singhritesh85/DevOps-Project-BankApplication-Multibranch-MultiCloud 6h ago
- singhritesh85/Bank-App-multibranch Thursday
- singhritesh85/DevOps-Project-BloggingApp-Deployment-Monitoring-PrometheusGrafana-Log... May 7
- singhritesh85/DevOps-Project-BankApplication-BlueGreen-Deployment-MultiCloud May 4



If you need to apply webhook in Azure DevOps Pipeline then in `azure-pipelines.yaml` trigger: provide the branch name as shown in the screenshot attached below.

```
trigger:
```

```
- main
```

This pipeline triggers for main branch only if you need to apply for other branches then provide the branch names as shown in the screenshot attached below.

```
trigger:
```

```
- main
- stage
- dev
```

For this Project I had created two Azure DevOps pipelines named as **mysql** and **bankapp**. The **mysql** pipeline will be created first and then **bankapp** pipeline. As I had not applied webhook for **mysql** pipeline and so **mysql** pipeline needs to run manually. If you want to apply the webhook for **bankapp** pipeline then use the method which I used to apply the webhook in production for Azure DevOps Pipeline (with the help of `trigger:`). Before running the Azure DevOps Pipeline for **mysql** create a kubernetes namespace with the name of mysql using the command `kubectl create ns mysql`.

The mysql pods got the mysql root password from Hashicorp vault.

```
[root@yellow ~]# kubectl create ns mysql
namespace/mysql created
```

After successfully running the mysql pipeline pods, pvc and services had been created as shown in the screenshot attached below.

```
[root@yellow ~]# kubectl get pods -n mysql --watch
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          3m13s
mysql-1   2/2     Running   0          2m16s
[root@yellow ~]#
[root@yellow ~]# kubectl get pvc -n mysql --watch
NAME           STATUS   VOLUME
mysql-data-mysql-0   Bound   pvc-
mysql-data-mysql-1   Bound   pvc-
[root@yellow ~]#
[root@yellow ~]# kubectl get svc -n mysql --watch
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
mysql-svc   ClusterIP  10.0.237.106  <none>       3306/TCP  3m26s
mysql-svc-headless ClusterIP  None        <none>       3306/TCP  3m25s
```

The screenshot shows the Azure Pipelines interface. At the top right, there's a blue button labeled 'New pipeline'. Below it, a search bar says 'Filter pipelines'. On the left, a sidebar has 'Recent' selected under 'Pipelines'. The main area is titled 'Recently run pipelines' and lists one pipeline: 'mysql'. It shows the last run details: '#2025 [REDACTED] - Create azure-pipelines.yaml', triggered manually for the 'main' branch, 6m ago, and a duration of 2m 14s.

Then I had created the pipeline for bankapp. I had created the Docker Image for bankapp deployment in that the base image was used as eclipse-temurin. **OpenJDK Docker Image is deprecated so in this project eclipse-temurin docker image has been used as a base image in the Dockerfile.**

To access Docker Image from the Azure Container Registry during Kubernetes Deployment I created a kubernetes Secrets as shown in the screenshot attached below.

```
kubectl create secret docker-registry bankapp-auth --docker-server=https://bankappprodcontainer24registry.azurecr.io --docker-username=bankappprodcontainer24registry --docker-password=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX -n bankapp
```

```
[root@REDACTED ~]# kubectl create secret docker-registry bankapp-auth --docker-server=https://bankappprodcontainer24registry.azurecr.io --docker-username=bankappprodcontainer24registry --docker-password=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX -n bankapp
secret/bankapp-auth created
```

The below screenshot shows how I created the pipeline for **bankapp**.

This screenshot shows the Azure DevOps interface for the 'my-demo-project'. The left sidebar has 'Pipelines' selected. The main area is titled 'Pipelines' and shows a list of recently run pipelines. One pipeline, 'mysql', is listed with its last run details: '#2025 [REDACTED] - Create azure-pipelines.yaml', triggered manually for the 'main' branch, 17m ago, and a duration of 2m 14s.

The screenshot shows the 'my-demo-project' pipeline configuration in Azure DevOps. The left sidebar has 'Pipelines' selected. The top navigation bar shows 'Connect', 'Select', 'Configure', and 'Review'. The main area says 'New pipeline' and 'Where is your code?'. It lists several options: 'Azure Repos Git' (YAML), 'Bitbucket Cloud' (YAML), 'GitHub' (YAML) [highlighted with a yellow box], 'GitHub Enterprise Server' (YAML), 'Other Git', and 'Subversion'. Below this, it says 'Use the classic editor to create a pipeline without YAML.'

The screenshot shows the 'my-demo-project' pipeline configuration in Azure DevOps. The left sidebar has 'Pipelines' selected. The top navigation bar shows '✓ Connect', 'Select', 'Configure', and 'Review'. The main area says 'New pipeline' and 'Select a repository'. It shows a list of repositories under 'My repositories': singhritesh85/kubernetes-manifests (42m ago), singhritesh85/Bank-App-using-Hashicorp-Vault (2h ago) [highlighted with a yellow box], singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud (3h ago), singhritesh85/DevOps-Project-BankApplication-Multibranch-MultiCloud (6h ago), singhritesh85/Bank-App-multibranch (Thursday), singhritesh85/DevOps-Project-BloggingApp-Deployment-Monitoring-PrometheusGrafana-Log... (May 7), singhritesh85/DevOps-Project-BankApplication-BlueGreen-Deployment-MultiCloud (May 4), and singhritesh85/DevOps-Project-Kubernetes-Multicluster-Multicloud (May 4).

After success execution of bankapp pipeline the pods and service had been created as shown in the screenshot attached below.

```
[root@REDACTED ~]# kubectl get all -n bankapp
NAME                                         READY   STATUS    RESTARTS   AGE
pod/bankapp-REDACTED                      1/1     Running   0          2m33s

NAME                           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/bankapp-service       ClusterIP  10.REDACTED.114  <none>        80/TCP    2m33s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/bankapp      1/1     1           1           2m33s

NAME                           DESIRED  CURRENT   READY   AGE
replicaset.apps/bankapp-REDACTED  1        1         1        2m33s
```

The screenshot shows the CircleCI Pipelines interface. At the top right, there are buttons for "New pipeline" and a three-dot menu. Below that is a search bar labeled "Filter pipelines". The main area is titled "Recently run pipelines". It lists two pipelines: "bankapp" and "mysql".

Pipeline	Last run
bankapp	#2025 [REDACTED] • Update bankapp.yaml ⌚ Individual CI for [REDACTED] main 🕒 3m ago 🕒 1m 46s
mysql	#2025 [REDACTED] • Create azure-pipelines.yaml ⌚ Manually triggered for [REDACTED] main 🕒 1h ago 🕒 2m 14s

Below screenshot shows the ingress rule which I used to access the Bank Application using the URL.

You can use the Ingress rule which I provided in the GitHub Repo

<https://github.com/singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud.git> with the file name **ingress-rule-prod.yaml**. First you need to create the kubernetes secrets as shown in the screenshot attached below.

kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace bankapp

```
[root@REDACTED ~]# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace bankapp
secret/ingress-tls created

[root@REDACTED ~]# kubectl apply -f ingress-rule-prod.yaml
ingress.networking.k8s.io/bankapp-ingress created
[root@REDACTED ~]# cat ingress-rule-prod.yaml
# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace bankapp
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress
  namespace: bankapp
  annotations:
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: azure-application-gateway
  tls:
  - secretName: ingress-tls
  rules:
  - host: bankapp.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: bankapp-service
            port:
              number: 80

[root@REDACTED ~]# kubectl get ing -A --watch
NAMESPACE     NAME           CLASS
bankapp       bankapp-ingress   azure-application-gateway
vault         vault-ingress    azure-application-gateway
HOSTS
bankapp.singhritesh85.com  4. [REDACTED].25  80, 443  11s
vault.singhritesh85.com    4. [REDACTED].25  80, 443  4h48m
ADDRESS
PORTS
AGE
```

```

cat ingress-rule-prod.yaml

# kubectl create secret tls ingress-tls --key mykey.key --cert STAR_singhritesh85_com.crt --namespace
bankapp

---

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bankapp-ingress
  namespace: bankapp
  annotations:
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: azure-application-gateway
  tls:
    - secretName: ingress-tls
  rules:
    - host: bankapp.singhritesh85.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: bankapp-service
                port:
                  number: 80

```

I did the entry for HOST **bankapp.singhritesh85.com** with Public IP Address as shown in the screenshot attached above in the Azure DNS Zone to create the Record Set of A Type.

singhritesh85.com | Recordsets

A record set is a collection of records in a zone that have the same Name and Type. They are used to define how your domain is resolved. If you don't see what you're looking for, try [Learn more](#).

Name	Type
@	NS
@	SOA

Add record set

Name: bankapp

Type: A – IPv4 Address records

Alias record set: No

TTL: 1

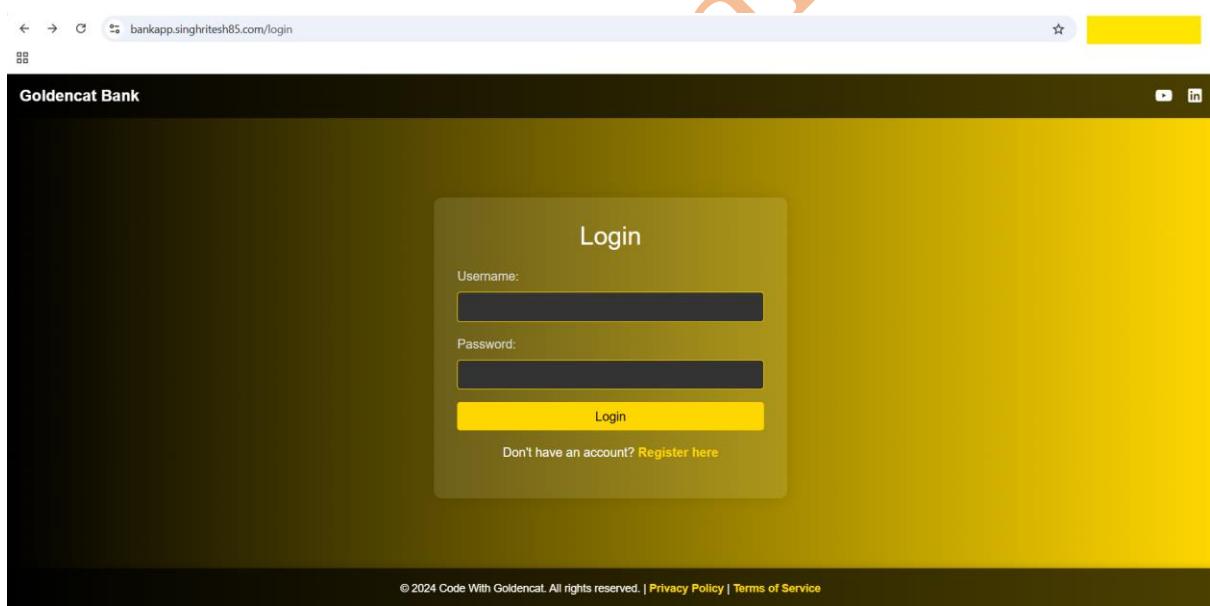
TTL unit: Hours

IP address: 4.25

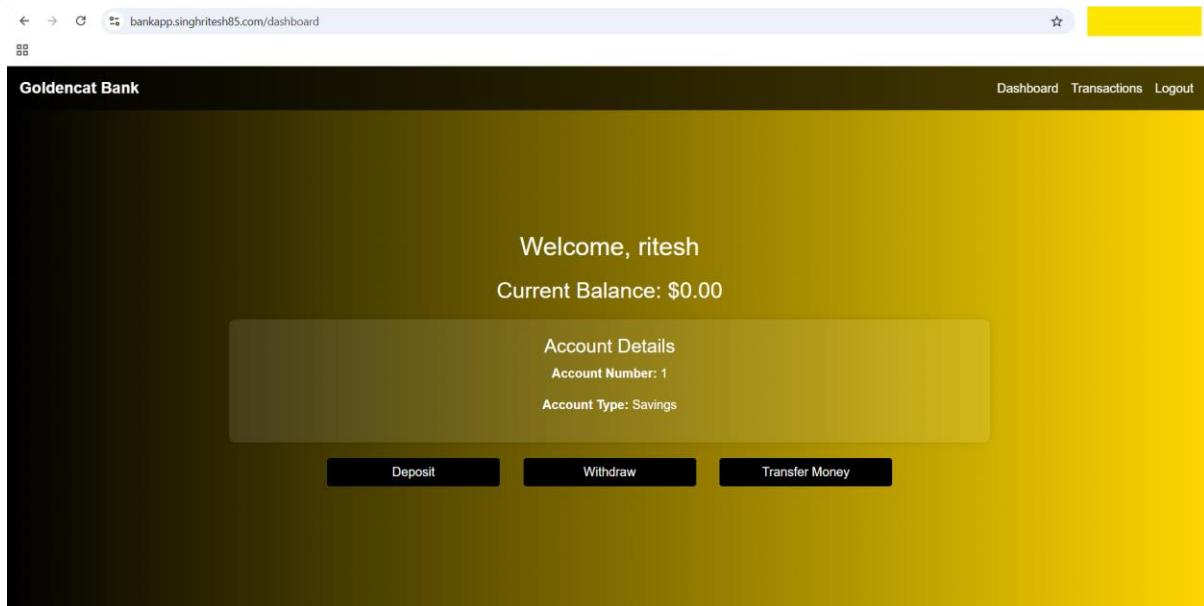
0.0.0.0

Add Cancel Give feedback

Finally, I was able to access the Bank Application using the URL as shown in the screenshot attached below.



I registered a user ritesh in the Bank Application as discussed in the production environment and then logged-in with the same user and I had also checked its entry in the MySQL database and found the same as shown in the screenshot attached below.



```
[root@██████████ ~]# kubectl exec -it mysql-0 -n mysql sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
Defaulted container "mysql" out of: mysql, vault-agent, vault-agent-init (init)
# mysql -h localhost -u root --password
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 103
Server version: 10.7.3-MariaDB-1:10.7.3+maria~focal mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| #mysql150#lost+found |
| bankappdb |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
6 rows in set (0.000 sec)

MariaDB [(none)]> use bankappdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [bankappdb]> show tables;
+-----+
| Tables_in_bankappdb |
+-----+
| account |
| transaction |
+-----+
2 rows in set (0.000 sec)

MariaDB [bankappdb]> select * from account;
+-----+-----+-----+-----+
| id | balance | password | username |
+-----+-----+-----+-----+
| 1 | 0.00 | ██████████ | ritesh |
+-----+-----+-----+-----+
1 row in set (0.000 sec)

MariaDB [bankappdb]>
```

Create snapshot of Hashicorp Vault and store it in Azure Storage Account Container

To create snapshot of Hashicorp vault and store it in Azure Storage Account Container I used the kubernetes CronJob which is present in the Github Repo <https://github.com/singhritesh85/kubernetes-manifests.git> at the path **snapshot-store**. I deployed this CronJob using Azure DevOps Pipeline. To run this pipeline, I used the VAULT_TOKEN and SAS Token for Azure Storage Account Container. I created an approle in Hashicorp Vault and its Token I used for VAULT_TOKEN and If you noticed in initial stage of the project when I created resources using Terraform script then I got the SAS Token for Azure Storage Account Container which you can get printed using the command **terraform output -json acr_azure_loki_vm_private_ip_container_sas_url**. The vault token and SAS Token I provided as Pipeline variable to run pipeline successfully. For this demonstration I ran this CronJob per minute but in your organisation, you need to run it periodically in non-production hours. In this CronJob I used a Docker Image **peterdavehello/azcopy:10** form Docker Hub Public Repo to run the **azcopy** command successfully. For your organisation, you can use your own Docker Image. Below screenshot shows the Image layers for this Docker Image. I would like to clearly mention here that the Vault Token which I used here was not the Root Token but the **Approle** based vault token.

Image Layers		?
1 ADD alpine-minirootfs-3.19.4-x86.tar.gz / # buildkit		3.1 MB
2 CMD ["/bin/sh"]		0 B
3 ARG AZCOPY_VERSION=10.27.1		0 B
4 LABEL name=docker-azcopy		0 B
5 LABEL version=10.27.1		0 B
6 LABEL maintainer=Peter Dave Hello <hsu@peterdavehello.org>		0 B
7 COPY /azcopy/azcopy /usr/local/bin # buildkit		21.01 MB
8 WORKDIR /WORKDIR		97 B
9 CMD ["azcopy"]		0 B

It is not suggestable to hardcode any Token or SAS Token in kubernetes manifests file directly So I used the Azure DevOps Pipeline variable.

First, I create the Policy and then the approle by attaching the policy to the approle as shown in the screenshot attached below.

```
vault policy write backup_policy - <<EOF
path "sys/storage/raft/snapshot" {
  capabilities = ["read"]
}
EOF
```

vault auth enable approle

vault write auth/approle/role/backup-role policies=backup_policy

vault read auth/approle/role/backup-role/role-id

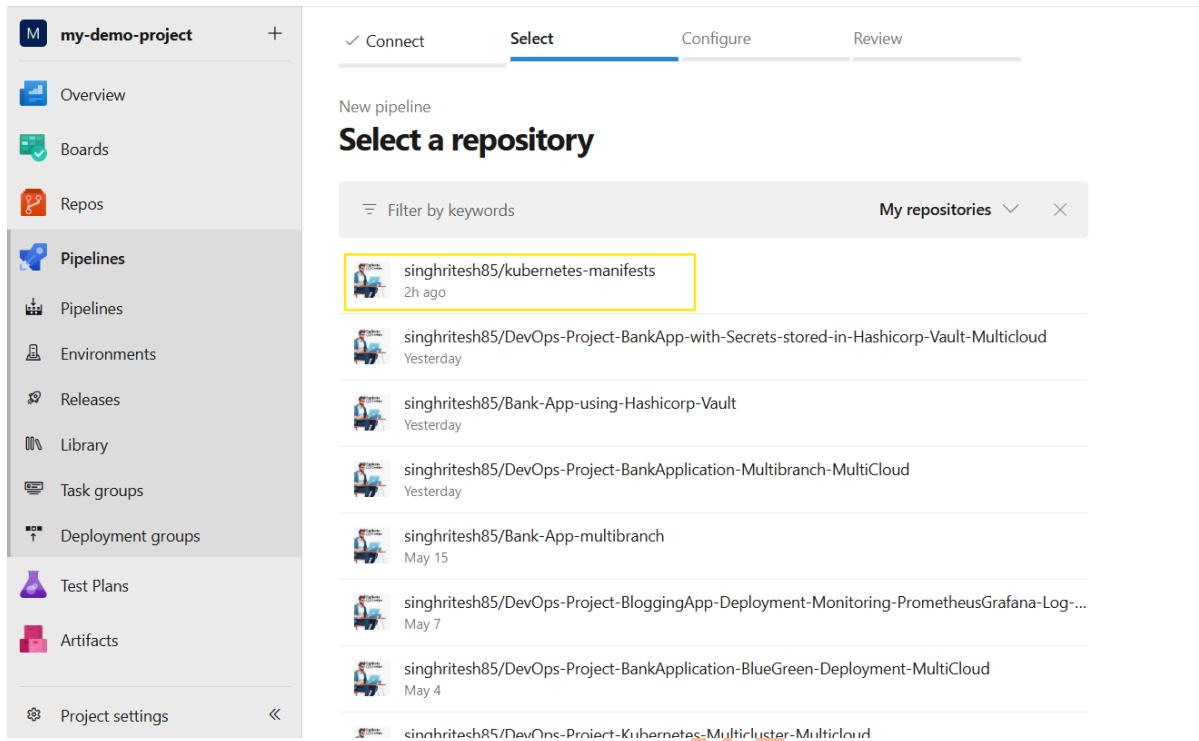
vault write -f auth/approle/role/backup-role/secret-id

Finally get the Token using the command as written below (don't run this command second time otherwise the older secret_id will be lost)

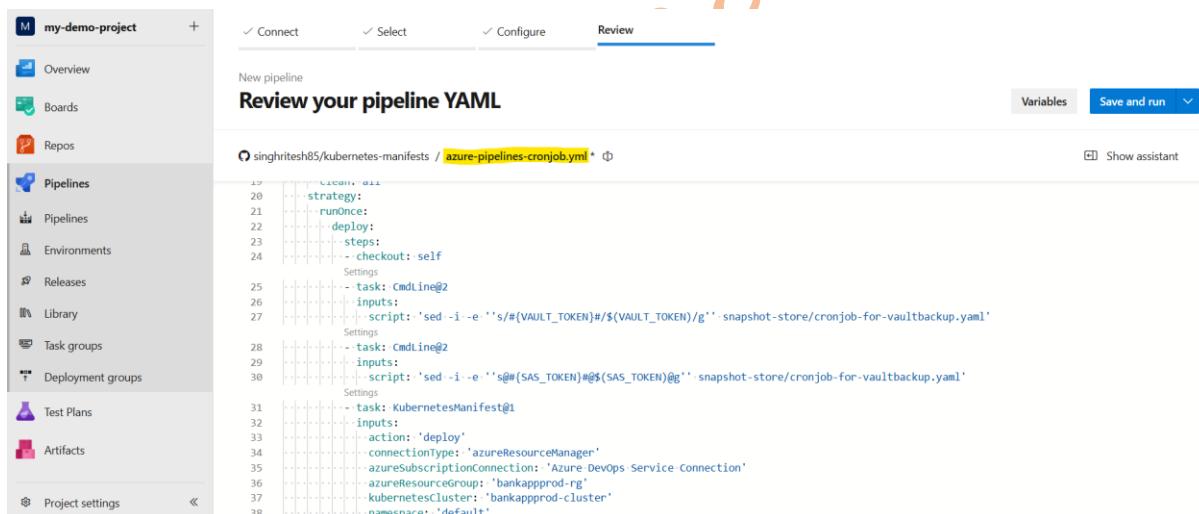
vault write auth/approle/login role_id=<role-id> secret_id=<secret-id>

```
[root@[REDACTED] ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault policy write backup_policy - <<EOF
> path "sys/storage/raft/snapshot" {
>   capabilities = ["read"]
>
> EOF
Success! Uploaded policy: backup_policy
/ $
/ $ vault auth enable approle
Success! Enabled approle auth method at: approle/
/ $
/ $ vault write auth/approle/role/backup-role policies=backup_policy
Success! Data written to: auth/approle/role/backup-role
/ $
/ $ vault read auth/approle/role/backup-role/role-id
Key      Value
--- -----
role_id [REDACTED]
/ $
/ $ vault write -f auth/approle/role/backup-role/secret-id
Key      Value
--- -----
secret_id [REDACTED]
secret_id_accessor [REDACTED]
secret_id_num_uses  0
secret_id_ttl     0s
/ $ vault write auth/approle/login role_id=[REDACTED] secret_id=[REDACTED]
Key      Value
--- -----
token
token_accessor [REDACTED]
token_duration  768h
token_renewable true
token_policies  ["backup_policy" "default"]
identity_policies []
policies       ["backup_policy" "default"]
token_meta_role_name backup-role
/ $
```

Below screenshots show how I had created the Azure DevOps Pipeline for cronjob to run.



The screenshot shows the 'Select a repository' step in the Azure DevOps pipeline creation process. A list of repositories is displayed, with 'singhritesh85/kubernetes-manifests' selected and highlighted by a yellow border. Other repositories listed include 'singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud', 'singhritesh85/Bank-App-using-Hashicorp-Vault', 'singhritesh85/DevOps-Project-BankApplication-Multibranch-MultiCloud', 'singhritesh85/Bank-App-multibranch', 'singhritesh85/DevOps-Project-BloggingApp-Deployment-Monitoring-PrometheusGrafana-Log...', 'singhritesh85/DevOps-Project-BankApplication-BlueGreen-Deployment-MultiCloud', and 'cinnhritesh85/DevOps-Project-Kubernetes-MultiCluster-MultiCloud'.



The screenshot shows the 'Review your pipeline YAML' step in the Azure DevOps pipeline creation process. It displays the YAML code for the pipeline:

```

19:   strategy:
20:     runOnce:
21:       deploy:
22:         steps:
23:           - checkout: self
24:             Settings:
25:               - task: CmdLine@2
26:                 inputs:
27:                   - script: 'sed -i -e ''$@#{VAULT_TOKEN}#${VAULT_TOKEN}/g'' snapshot-store/cronjob-for-vaultbackup.yaml'
28:             Settings:
29:               - task: CmdLine@2
30:                 inputs:
31:                   - script: 'sed -i -e ''$@#{SAS_TOKEN}#${SAS_TOKEN}@g'' snapshot-store/cronjob-for-vaultbackup.yaml'
32:             Settings:
33:               - task: KubernetesManifest@1
34:                 inputs:
35:                   - action: 'deploy'
36:                     connectionType: 'azureDevOpsServiceConnection'
37:                     azureSubscriptionConnection: 'Azure DevOps Service Connection'
38:                     azureResourceGroup: 'bankappprod-rg'
39:                     kubernetesCluster: 'bankappprod-cluster'
40:                     namespace: 'default'

```

Below screenshot show how I provided vault token and SAS Token in Azure DevOps Pipeline.

The screenshot shows two separate instances of the 'New variable' dialog box, each with its own 'Name' and 'Value' fields, a checked 'Keep this value secret' checkbox, and an unchecked 'Let users override this value when running this pipeline' checkbox. The 'OK' button is highlighted in blue in both dialogs.

Review your pipeline YAML

```

23   steps:
24     - checkout: self
      Settings
25       task: KubernetesManifest@1
26       inputs:
27         action: 'deploy'
28         connectionType: 'azureResourceManager'
29         azureSubscriptionConnection: 'Azure DevOps Service Connection'
30         azureResourceGroup: 'bankappprod-rg'
31         kubernetesCluster: 'bankappprod-cluster'
32         namespace: 'default'
33         manifests: 'snapshot-store/*'
34

```

New variable

Name: VAULT_TOKEN

Value: [REDACTED]

Keep this value secret

Let users override this value when running this pipeline

To reference a variable in YAML, prefix it with a dollar sign and enclose it in parentheses. For example: \$(VAULT_TOKEN)

To use a variable in a script, use environment variable syntax. Replace . and space with _, capitalize the letters, and then use your platform's syntax for referencing an environment variable. Examples:

- Batch script: %VAULT_TOKEN%
- PowerShell script: \${env:VAULT_TOKEN}
- Bash script: \$(VAULT_TOKEN)

To use a secret variable in a script, you must explicitly map it as an environment variable.

Learn about variables Cancel OK

New variable

Name: SAS_TOKEN

Value: [REDACTED]

Keep this value secret

Let users override this value when running this pipeline

To reference a variable in YAML, prefix it with a dollar sign and enclose it in parentheses. For example: \$(SAS_TOKEN)

To use a variable in a script, use environment variable syntax. Replace . and space with _, capitalize the letters, and then use your platform's syntax for referencing an environment variable. Examples:

- Batch script: %SAS_TOKEN%
- PowerShell script: \${env:SAS_TOKEN}
- Bash script: \$(SAS_TOKEN)

To use a secret variable in a script, you must explicitly map it as an environment variable.

Learn about variables Cancel OK

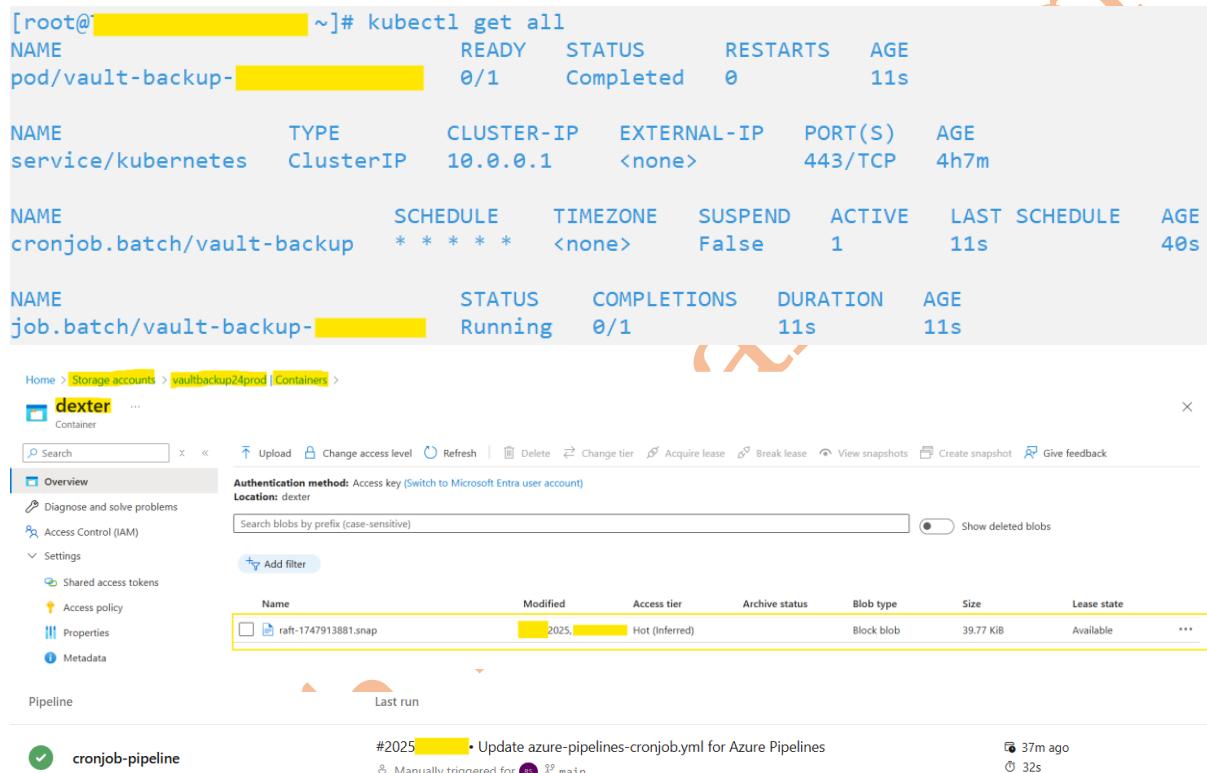
It is important to mention here how I provided variables for SAS Token in Azure DevOps Pipeline variable. Below is the original SAS Token which I got from **terraform output -json acr_azure_loki_vm_private_ip_container_sas_url** command.

```
"https://vaultbackup24prod.blob.core.windows.net/dexter?XX=2018-11-09\XXXXXXXX=X\XXXXXXXX=2025-03-21\XXXXXXXX=2029-03-21\XXXXXXXXXXXXrawl\XXXXXXXXXr=https\XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"XXXXXXX"
```

Below variable's value I provided in the Pipeline variable. Please see how I provided / (forward slash) and \ (backslash) otherwise Pod created after CronJob runs will through Error.

```
\\"https://vaultbackup24prod.blob.core.windows.net/dexter?XX=2018-11-09\\xxxxxxxx=X\\xxxxxxxx=2025-03-21\\xxxxxxxx=2029-03-21\\xxxxxxxxxxrawl\\xxxxxxxxxR=https\\xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\\\"
```

Below screenshot shows that CronJob ran successfully and snapshot had been copied to the Asure Storage Account container.



```
[root@xxxxxxxx ~]# kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/vault-backup-xxxxxxxxxxxxxxxxxxxxx      0/1     Completed   0          11s

NAME           TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.0.0.1    <none>       443/TCP   4h7m

NAME           SCHEDULE   TIMEZONE   SUSPEND   ACTIVE   LAST SCHEDULE   AGE
cronjob.batch/vault-backup   * * * * *  <none>     False     1        11s   40s

NAME           STATUS    COMPLETIONS   DURATION   AGE
job.batch/vault-backup-xxxxxxxxxxxxx       Running   0/1         11s       11s
```

Home > Storage accounts > vaultbackup24prod > Containers > dexter Container

Overview

Authentication method: Access key (Switch to Microsoft Entra user account)

Location: dexter

Search blobs by prefix (case-sensitive)

Show deleted blobs

Name: raft-1747913881.snap

Modified: 2025-03-21, Access tier: Hot (Inferred), Blob type: Block blob, Size: 39.77 KiB, Lease state: Available

Pipeline

Last run:

- cronjob-pipeline: #2025 - Update azure-pipelines-cronjob.yml for Azure Pipelines (Manually triggered for main) 37m ago
- cronjob-pipeline: #2025 - Update azure-pipelines-cronjob.yml for Azure Pipelines (Manually triggered for main) 32s ago

By default, Approle exists for 32 days if you want it for a greater number of days then you need to mention the same with `token_ttl` and `token_max_ttl`.

Restore the Hashicorp Vault Backup

Consider the situation if the secrets from the vault had been deleted and you want to restore the secrets from the backup you had taken. As mentioned in the above section I took the backup of vault and stored it in Azure Storage Account Container. I had created a secret at the path **bankapp/database/config** for demonstration purpose I had deleted it as shown in the screenshot attached below.

The screenshot displays two separate views of a HashiCorp Vault secrets store named 'bankapp' (version 2).
 Top View: Shows a single secret entry named 'database/' with a value of '1'. The interface includes a search bar, a 'Create secret +' button, and navigation controls (1-1 of 1, < >).
 Bottom View: Shows a message 'No secrets yet' indicating no secrets have been created. It also includes a 'Create secret +' button.

In backup section I had created a policy named as backup-policy which had only read access which is not sufficient to restore backup So I had created a policy with the privileges as shown below.

```
vault policy write backup_policy - <<EOF
path "sys/storage/raft/snapshot" {
  capabilities = ["update"]
}

path "sys/storage/raft/snapshot-force" {
  capabilities = ["update"]
}

EOF

vault write auth/approle/role/backup-role policies=backup_policy

vault read auth/approle/role/backup-role/role-id

vault write -f auth/approle/role/backup-role/secret-id

vault write auth/approle/login role_id=<provide role id> secret_id=<provide secret id>
```

```
[root@... ~]# kubectl exec -it vault-0 -n vault sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
/ $ vault policy delete backup_policy
Success! Deleted policy: backup_policy
/ $ vault policy write backup_policy - <<EOF
> path "sys/storage/raft/snapshot" {
>   capabilities = ["update"]
> }
> path "sys/storage/raft/snapshot-force" {
>   capabilities = ["update"]
> }
> EOF
Success! Uploaded policy: backup_policy
/ $ vault write auth/approle/role/backup-role policies=backup_policy
Success! Data written to: auth/approle/role/backup-role
/ $ vault read auth/approle/role/backup-role/role-id
Key      Value
---    -----
role_id  [REDACTED]
/ $ vault write -f auth/approle/role/backup-role/secret-id
Key      Value
---    -----
secret_id [REDACTED]
secret_id_accessor [REDACTED]
secret_id_num_uses 0
secret_id_ttl     0s

/ $ vault write auth/approle/login role_id=[REDACTED] secret_id=[REDACTED]
Key      Value
---    -----
token   [REDACTED]
token_accessor [REDACTED]
token_duration 768h
token_renewable true
token_policies  ["backup_policy" "default"]
identity_policies []
policies       ["backup_policy" "default"]
token_meta_role_name backup-role
/ $
```

Created the Approle and got the token in the same way which I discussed above in the snapshot.

The screenshot shows the Azure DevOps Pipelines interface for a project named 'my-demo-project'. The left sidebar has a 'Pipelines' section selected. The main area displays a table for 'Recently run pipelines'. One row is visible for the 'cronjob-pipeline', which was triggered manually and completed 32 seconds ago. The pipeline name is highlighted with a yellow box.

Pipeline	Last run
cronjob-pipeline	#2025 [REDACTED] • Update azure-pipelines-cronjob.yml for Azure Pipelines Manually triggered for [REDACTED] main

my-demo-project

Connect Select Configure Review

New pipeline

Where is your code?

- Azure Repos Git** YAML
Free private Git repositories, pull requests, and code search
- Bitbucket Cloud** YAML
Hosted by Atlassian
- GitHub** YAML
Home to the world's largest community of developers
- GitHub Enterprise Server** YAML
The self-hosted version of GitHub Enterprise
- Other Git**
Any generic Git repository
- Subversion**
Centralized version control by Apache

Use the [classic editor](#) to create a pipeline without YAML.

my-demo-project

✓ Connect **Select** Configure Review

New pipeline

Select a repository

Filter by keywords My repositories X

- singhritesh85/kubernetes-manifests**
9m ago
- singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud
Yesterday
- singhritesh85/Bank-App-using-Hashicorp-Vault
Yesterday
- singhritesh85/DevOps-Project-BankApplication-Multibranch-MultiCloud
Yesterday
- singhritesh85/Bank-App-multibranch
May 15
- singhritesh85/DevOps-Project-BloggingApp-Deployment-Monitoring-PrometheusGrafana-Log-...
May 7
- singhritesh85/DevOps-Project-BankApplication-BlueGreen-Deployment-MultiCloud
May 4
- singhritesh85/DevOps-Project-Kubernetes-Multicluster-Multicloud

New pipeline

Configure your pipeline

- Starter pipeline**
Start with a minimal pipeline that you can customize to build and deploy your code.
- Existing Azure Pipelines YAML file**
Select an Azure Pipelines YAML file in any branch of the repository.

Show more

Then I created an Azure DevOps Pipeline and ran it as shown in the screenshot attached below. Variable VAULT_TOKEN and SAS_TOKEN_RESTORE was created as discussed below.

New variable

Name: VAULT_TOKEN

Value: (redacted)

Keep this value secret

Let users override this value when running this pipeline

To reference a variable in YAML, prefix it with a dollar sign and enclose it in parentheses. For example: \${VAULT_TOKEN}

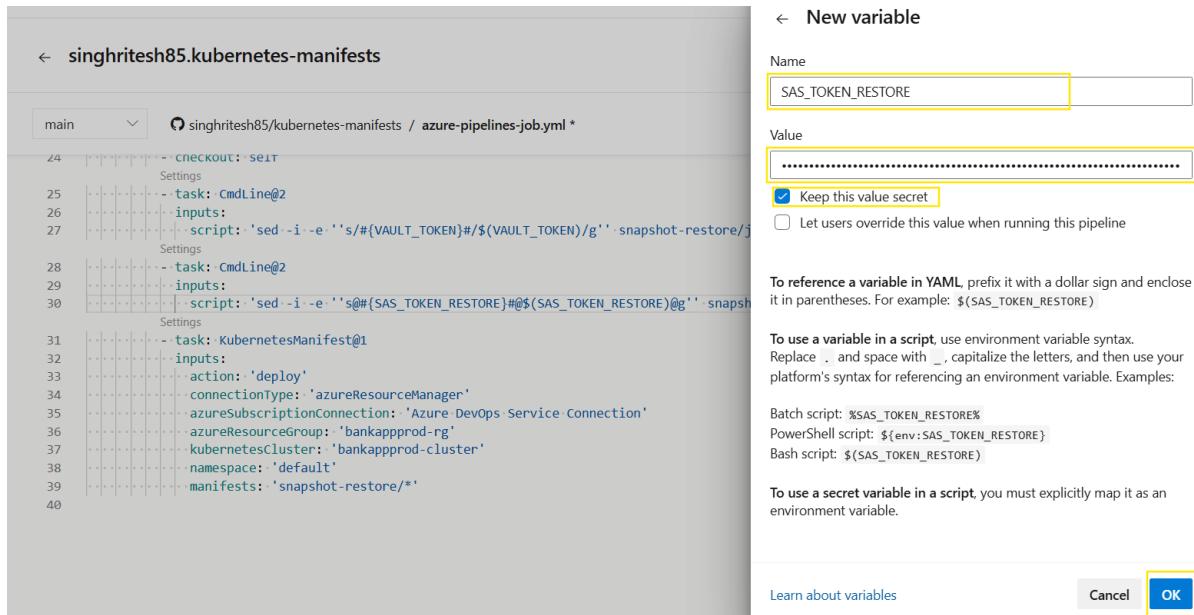
To use a variable in a script, use environment variable syntax. Replace `_` and space with `_`, capitalize the letters, and then use your platform's syntax for referencing an environment variable. Examples:

- Batch script: %VAULT_TOKEN%
- PowerShell script: \${env:VAULT_TOKEN}
- Bash script: \$(VAULT_TOKEN)

To use a secret variable in a script, you must explicitly map it as an environment variable.

Learn about variables

Cancel OK



It is important to mention here how I provided variables for SAS Token in Azure DevOps Pipeline variable. Below is the original SAS Token which I got from **terraform output -json acr_azure_loki_vm_private_ip_container_sas_url** command.

"<https://vaultbackup24prod.blob.core.windows.net/dexter?XX=2018-11-09\xxxxxxx=X\xxxxxxx=2025-03-21\xxxxxxx=2029-03-21\xxxxxxxxxxrawl\xxxxxxxr=https\xxx>"

Below variable's value I provided in the Pipeline variable. Please see how I provided / (forward slash) and \ (backslash) otherwise Pod created after CronJob runs will through Error.

\\"<https://vaultbackup24prod.blob.core.windows.net/dexter/<name-of-the-vault-snapshot-file-which-needs-to-be-restored>?XX=2018-11-09\\xxxxxxx=X\\xxxxxxx=2025-03-21\\xxxxxxx=2029-03-21\xxxxxxxxxxrawl\xxxxxxxr=https\xxx>"

Below screenshot shows that Kubernetes Job ran successfully and snapshot had been restored.

Pipeline	Last run
job-pipeline	#2025 [redacted] • Update job-for-vault-restore.yaml ↳ Manually triggered for [user] [branch] main [green checkmark] 27s
cronjob-pipeline	#2025 [redacted] • Update azure-pipelines-cronjob.yaml for Azure Pipelines ↳ Manually triggered for [user] [branch] main [green checkmark] 32s

Before running the Azure DevOps Pipeline **job-pipeline** as shown in the screenshot attached above you need to be assured that the name of the snapshot file given in the variable **SNAPSHOT_TOKEN_RESTORE** in pipeline's variable and in the file **job-for-vault-restore.yaml** present

at the path **snapshot-restore** in GitHub Repo <https://github.com/singhritesh85/kubernetes-manifests.git> should be the one which we wanted to be restored.

```
[root@...] ~]# kubectl get all
NAME                      READY   STATUS    RESTARTS   AGE
pod/vault-restore-[...]   0/1     Completed   0          29s

NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.0.0.1   <none>       443/TCP   8h

NAME           STATUS    COMPLETIONS   DURATION   AGE
job.batch/vault-restore   Complete   1/1          12s        30s
```

Then I checked the vault UI and found that secrets had been restored successfully as shown in the screenshot attached below.

The screenshot shows the HashiCorp Vault UI for the 'bankapp' application. The 'Secrets' tab is selected. A single secret named 'database/' is listed. The UI includes a search bar, a 'Create secret +' button, and navigation controls (1-1 of 1, < >).

Source Code: - <https://github.com/singhritesh85/Bank-App-using-Hashicorp-Vault.git>

GitHub Repo: - <https://github.com/singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud.git>

Kubernetes Manifests File: - <https://github.com/singhritesh85/kubernetes-manifests.git>

Terraform Script: - <https://github.com/singhritesh85/DevOps-Project-BankApp-with-Secrets-stored-in-Hashicorp-Vault-Multicloud.git>

Reference: - <https://github.com/Goldencat98/Bank-App.git>