

Vault

소개 및 적용방안

2021-11-10

ck 1-3 우태건



Why?

- Kubernetes Secret 의 보안이 취약 (Only Base64 Encode)

```
[root@ck172-1 /root]$ kubectl get secret passwords -n hyperauth -o jsonpath="{[['data']][ 'DB_PASSWORD']}" | base64 -d
keycloak[root@ck172-1 /root]$
```

```
[root@ck172-1 /root]$ kubectl get secret hyperauth-https-secret -n hyperauth -o jsonpath="{['data']['tls\.crt']}" | base64 -d
-----BEGIN CERTIFICATE-----
MIIEYGCACA7CgAwIBAgIUdNsuFtB6xHgPXVPRzfH0DtmL8mUwDQYJKoZIhvcNAQEL
CgAwUzELMAkGA1UEBhMCs1IxCzAJBgNVBAGMAkNBMRyYwFAYDVQQHDA1TYW4gRnJh
bmNpc2NvMQ0wCwYDVQQKDARUbWF4MR4wDgYDVQQDDAdUbWF4IENBMB4XDTE1MDYw
MjA1MjE0NVowXDTIzMDYyMjA1MjE0NVowVTELMAGAw1UEBhMCs1IxCzAJBgNVBAGM
AkNBMRyYwFAYDVQQHDA1TYW4gRnJhbmNpc2NvMQ0wCwYDVQQKDARUbWF4MR4wDgYD
VQQDDAloeXBldGwggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDT
UH06s8iOD7MxuTboL6Xxhnl/jXvgir0JcWkI9q53yWkvsPB3BP4v4L8DP3sZkDuy
9xmvHdixN1NdKvdDqir+ibeeG7au8KwWSUluSCcsG5uVSCUnTP6FGHqT/1CUBvix
```

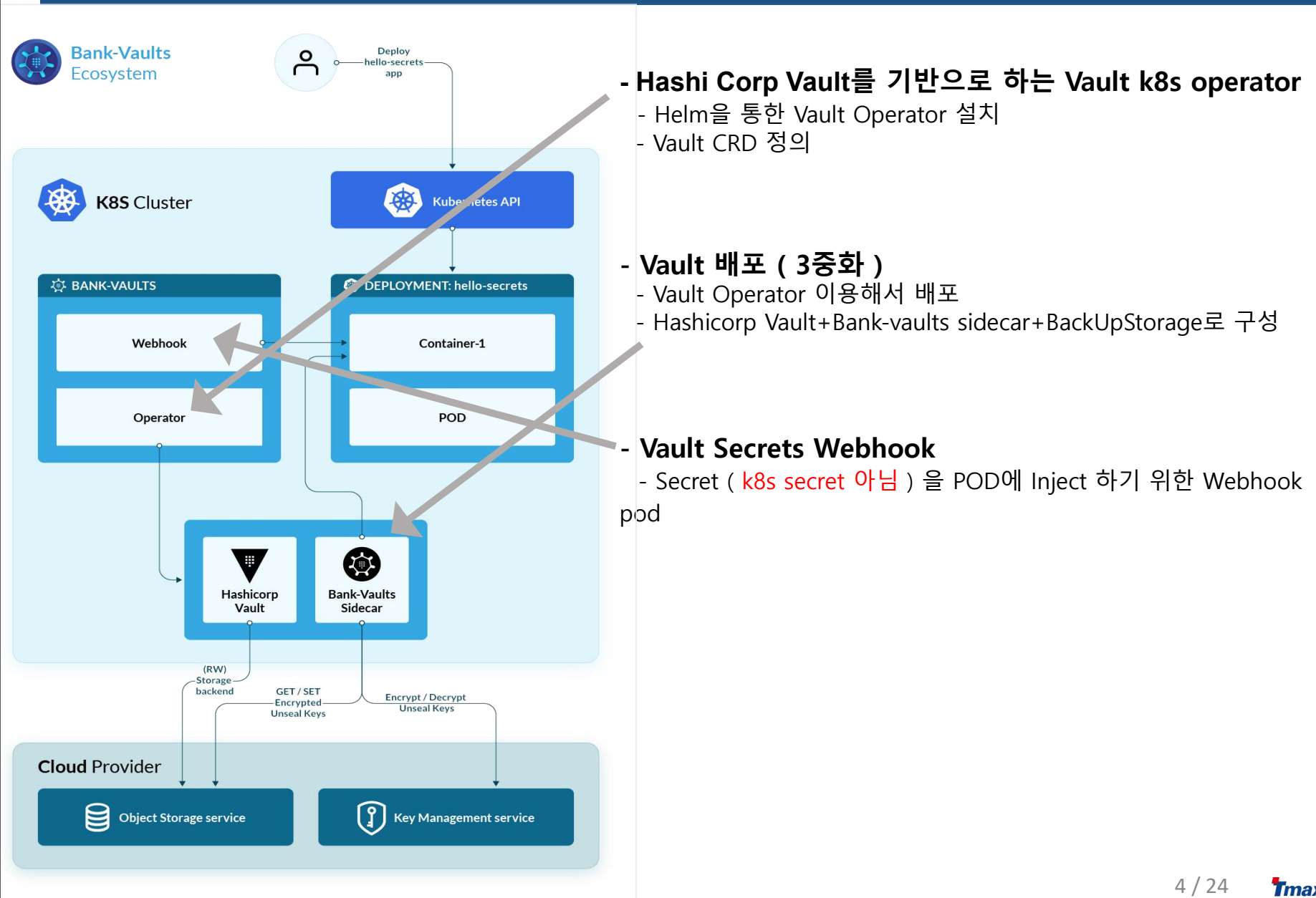
- Secret으로도 사용하지 않는 경우도 많음
- User 별 권한 Secret 권한 관리도 제대로 되고 있지 않음
- Template을 이용한 Catalog 에서 유저의 민감정보가 쉽게 노출되고 있음
- **KMS (Key Management Service)** 의 중요성 부각

What is HashiCorp Vault

- Secret을 잘 관리하고 민감한 데이터를 보관하는 저장소 (금고)
- 중앙에서 한번에 Secret을 관리하므로, 여러 어플리케이션과 Infrastructure에서 쉽게 사용가능
- 인증 및 권한 인가 가능
 - Keycloak을 통한 인증 + Kubernetes ServiceAccount를 이용한 인가 도입 예정
- 인증서 발급 가능



What is Banzai Bank-Vaults



Install Vault 1

* 참조 : <https://github.com/tmax-cloud/vault/tree/main/install-vault-operator>

```
$ git clone https://github.com/tmax-cloud/vault.git
```

Vault Operator, Vault, Secret Webhook 모두 Vault Namespace에 설치하는것으로 간주한다.

1. Install Banzai Vault Operator (Helm 이용)

```
$ kubectl create namespace vault
```

```
$ helm repo add banzaicloud-stable https://kubernetes-charts.banzaicloud.com
```

vault-operator 와 vault CRD 생성

```
$ helm upgrade --install vault-operator banzaicloud-stable/vault-operator -n vault
```

2. Install RBAC for Vault

```
$ kubectl apply -f install-vault-operator/rbac.yaml
```

Install Vault 2

3. Deploy Vault

Dev 모드 (Vault 1개)


\$ kubectl apply -f install-vault-operator/cr.yaml

Prod 모드 (Vault 3개)

\$ kubectl apply -f install-vault-operator/cr-raft.yaml

Vault 의 Service Type (default : LoadBalancer) 의 변경을 원하는 경우 cr 파일에서 수정

```
59 # Specify the Service's type where the Vault Service is exposed
60 # Please note that some Ingress controllers like https://git
61 # forces you to expose your Service on a NodePort
62 serviceType: LoadBalancer
```



4. Check Vault (외부(작업한 노드)에서 Vault 명령어 날리기)

\$ export VAULT_SKIP_VERIFY=true

외부에서 vault에 접근할 경우 외부 IP 명시

\$ export VAULT_ADDR=https://vault.vault:8200

Vault Root Token (운영환경에서는 vault-unseal-keys Secret 지우는 것 권고), UI 로그인시에 사용

\$ export VAULT_TOKEN=\$(kubectl get secrets -n vault vault-unseal-keys -o jsonpath={.data.vault-root} | base64 --decode)

Install Vault 3

5. Vault UI 접근

Vault LoadBalancer IP:8200 으로 접근

앞서 확인했던 Vault Root Token
으로 접속 가능

Sign in to Vault

Method

Token

Token

Sign In

Contact your administrator for login credentials.



Secrets Engines

<div><div></div><div>cubbyhole/</div><div>cubbyhole_a7407b84</div></div>
<div><div></div><div>secret/</div><div>v2 kv_fff3f3dc</div></div>

Install Vault 4

6. Deploy Vault Secret Webhook (Helm 이용)

Vault Secret Webhook은 반드시 Vault와 같은 Cluster 내부에 설치될 필요가 없음
External Cluster의 Vault를 바라보게 설정 가능 --> 더욱 중앙화된 Vault 운용가능

```
$ helm repo add banzaicloud-stable https://kubernetes-charts.banzaicloud.com
```

env.VAULT_ADDR 은 외부 Loadbalancer IP 로 치환가능, 빼도됨

env.VAULT_ADDR을 세팅하면, annotations에서 vault.security.banzaicloud.io/vault-addr: 생략가능

```
$ helm upgrade --namespace vault --install vault-secrets-webhook banzaicloud-stable/vault-secrets-webhook --set env.VAULT_ADDR=https://vault.vault:8200
```

< Vault 구성 >

vault	vault-0	3/3	Running
vault	vault-1	3/3	Running
vault	vault-2	3/3	Running
vault	vault-configurer-bc8c6f458-zjh6l	1/1	Running
vault	vault-operator-7fb5c6c664-lmck9	1/1	Running
vault	vault-secrets-webhook-5446896d5b-l7575	1/1	Running
vault	vault-secrets-webhook-5446896d5b-pqpgv	1/1	Running

Vault 활용 방안 Overview 1

- K8s Secret 사용 패턴 별 활용

1. Secret 의 정보를 POD 의 ENV로 사용

```
- name: KEYCLOAK_PASSWORD
  valueFrom:
    secretKeyRef:
      name: passwords
      key: HYPERAUTH_PASSWORD
- name: CERTS_PASSWORD
  valueFrom:
    secretKeyRef:
      name: passwords
      key: CERTS_PASSWORD
```

2. Secret 자체를 Volume Mount 해서 사용

```
volumes:
- name: ssl
  secret:
    secretName: hyperauth-certs-by-certmanager
- name: kafka
  secret:
    secretName: hyperauth-kafka-jks-by-certmanager
```

Vault 활용 방안 Overview 2

*참조 : <https://github.com/tmax-cloud/vault/tree/main/example/sandbox-scenario/aaa>

- Secret 의 정보를 POD 의 ENV로 사용하는 경우

- Secret Webhook을 이용해서 Env로 Vault의 Secret Data를 넣어줄수 있다.
- InitContainer가 추가되어서 Vault-Env Binary를 /vault 경로에 Inject 해준다.
- Vault에 Data를 저장한 후, Env의 Value로 Vault 내부 Data의 경로를 명시
 - 1) Env의 Value로 경로 명시
 - 2) 경로를 저장한 ConfigMap을 Env로 사용
 - 3) 경로를 Base64Encode 해서 만든 Secret 을 Env로 사용

```
env:  
- name: AAA_SECRET_DATA_DIRECT_ENV  
  value: vault:secret/data/sandbox/aaa#AAA_SECRET_DATA_DIRECT_ENV  
- name: AAA_SECRET_DATA_WITH_CM_ENV  
  valueFrom:  
    configMapKeyRef:  
      name: key-configmap  
      key: AAA_SECRET_DATA_WITH_CM_ENV  
- name: AAA_SECRET_DATA_WITH_SECRET_ENV  
  valueFrom:  
    secretKeyRef:  
      name: key-secret  
      key: AAA_SECRET_DATA_WITH_SECRET_ENV
```

Vault 활용 방안 Overview 3

- Secret 자체를 Volume Mount 해서 사용

- Consul Template ConfigMap을 사용해서 Pod의 특정경로에 원하는 포맷으로 Vault 로 부터 가져온 Secret Data File 생성가능

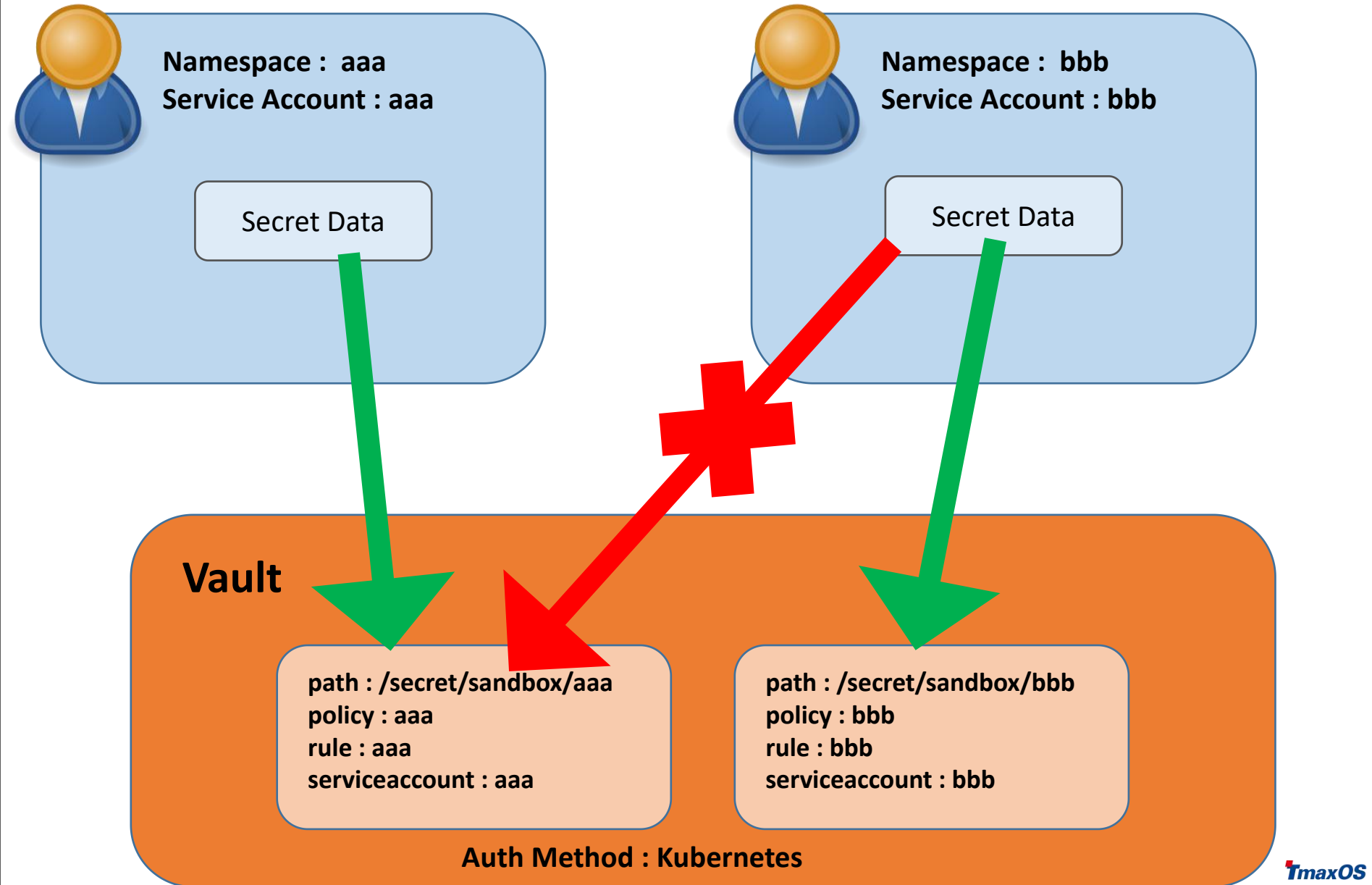
```
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: aaa
  name: aaa-consul-template
  namespace: aaa
data:
  config.hcl: |
    template {
      contents = <<EOH
      {{- with secret "secret/data/sandbox/aaa" }}
      {{ .Data.data.AAA_SECRET_DATA_WITH_MOUNT_FILE }}
      {{ end }}
      EOH
      destination = "/vault/secret/AAA_SECRET_DATA_WITH_MOUNT_FILE"
```

- Pod Spec에 Volume Mount 절 생략가능
- SideCar Container가 Secret File을 EmptyDir를 이용해서 Mount 시켜준다.

- POD 의 Annotation으로 Secret Injection 관리

```
annotations:
  #vault.security.banzaicloud.io/vault-addr: "https://vault.vault:8200" # optional, the address of the Vault service, default values
  vault.security.banzaicloud.io/vault-role: "aaa" # optional, the default value is the name of the ServiceAccount the Pod runs in, i
  vault.security.banzaicloud.io/vault-skip-verify: "true" # optional, skip TLS verification of the Vault server certificate
  #vault.security.banzaicloud.io/vault-tls-secret: "vault-tls" # optional, the name of the Secret where the Vault CA cert is, if not
  #vault.security.banzaicloud.io/vault-agent: "false" # optional, if true, a Vault Agent will be started to do Vault authentication,
  vault.security.banzaicloud.io/vault-path: "kubernetes" # optional, the Kubernetes Auth mount path in Vault the default value is "k
  vault.security.banzaicloud.io/vault-ct-configmap: "aaa-consul-template"
```

Sandbox Scenario 정리



Sandbox Scenario Example 1

*참조 : <https://github.com/tmax-cloud/vault/tree/main/example/sandbox-scenario/aaa>

root token으로 vault admin 계정 로그인

```
$ export VAULT_TOKEN=$(kubectl get secrets vault-unseal-keys -n vault -o jsonpath={.data.vault-root} | base64 --decode)
```

aaa 에 대한 작업

```
$ kubectl create namespace aaa
```

```
$ kubectl create serviceaccount aaa -n aaa
```

sandbox/aaa path밑의 kv store에 대한 모든 권한을 aaa SA에 발급

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

```
path "secret/data/sandbox/aaa" {  
  capabilities = ["create", "read", "update", "delete", "list"]  
}  
EOF
```

```
$ vault write auth/kubernetes/role/aaa ⚭  
  bound_service_account_names=aaa ⚭  
  bound_service_account_namespaces=aaa ⚭  
  policies=aaa ⚭  
  ttl=2400h
```

Secret Data 를 Vault 에 저장

```
vault kv put secret/sandbox/aaa ⚭  
AAA_SECRET_DATA_DIRECT_ENV=aaa_direct_env ⚭  
AAA_SECRET_DATA_WITH_CM_ENV=aaa_cm_env ⚭  
AAA_SECRET_DATA_WITH_SECRET_ENV=aaa_secret_env ⚭  
AAA_SECRET_DATA_WITH_MOUNT_FILE=aaa_mount_file_env
```

데이터 확인

```
vault kv get secret/sandbox/aaa
```

Sandbox Scenario Example 2

*참조 : <https://github.com/tmax-cloud/vault/tree/main/example/sandbox-scenario/aaa>

vault 에 저장한 data 에 대한 경로를 담고 있는 configMap, Secret, Consul Template ConfigMap 생성

\$ kubectl apply -f configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: key-configmap
  namespace: aaa
data:
  AAA_SECRET_DATA_WITH_CM_ENV: vault:secret/data/sandbox/aaa#AAA_SECRET_DATA_WITH_CM_ENV
```

\$ kubectl apply -f secret.yaml # Vault 경로를 Base64 Encode

```
apiVersion: v1
kind: Secret
metadata:
  name: key-secret
  namespace: aaa
data:
  AAA_SECRET_DATA_WITH_SECRET_ENV: dmFlbHQ6c2VjcmlVbGZlRmVvc2FuZGJveC9hYWVjQUFBX1NFQ1JFVF9EQVRBX1dJVEhfU0V0Vg==
type: Opaque
```

\$ kubectl apply -f consul-template.yaml

contents에 Vault Data 경로 기입

destination에 Secret File이
떨어질 경로 기입

```
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: aaa
  name: aaa-consul-template
  namespace: aaa
data:
  config.hcl: |
    template {
      contents = <<EOH
        {{- with secret "secret/data/sandbox/aaa" }}
        {{ .Data.data.AAA_SECRET_DATA_WITH_MOUNT_FILE }}
        {{ end }}
      EOH
      destination = "/vault/secret/AAA_SECRET_DATA_WITH_MOUNT_FILE"
```


Sandbox Scenario Example 3

*참조 : <https://github.com/tmax-cloud/vault/tree/main/example/sandbox-scenario/aaa>

Test Deployment 생성

\$ kubectl apply -f deployment.yaml

```
annotations:
  #vault.security.banzaicloud.io/vault-addr: "https://vault.vault:8200" #
  vault.security.banzaicloud.io/vault-role: "aaa" # optional, the default
  vault.security.banzaicloud.io/vault-skip-verify: "true" # optional, skip
  #vault.security.banzaicloud.io/vault-tls-secret: "vault-tls" # optional,
  #vault.security.banzaicloud.io/vault-agent: "false" # optional, if true,
  vault.security.banzaicloud.io/vault-path: "kubernetes" # optional, the K
  vault.security.banzaicloud.io/vault-ct-configmap: "aaa-consul-template"
```

```
spec:
  serviceAccountName: aaa
  containers:
  - name: alpine
    image: alpine
    command: ["sh", "-c", "echo Direct Env : $AAA_SECRET_DATA_DIRECT_ENV
    && echo going to sleep... && sleep 10000"]
    env:
    - name: AAA_SECRET_DATA_DIRECT_ENV
      value: vault:secret/data/sandbox/aaa#AAA_SECRET_DATA_DIRECT_ENV
    - name: AAA_SECRET_DATA_WITH_CM_ENV
      valueFrom:
        configMapKeyRef:
          name: key-configmap
          key: AAA_SECRET_DATA_WITH_CM_ENV
    - name: AAA_SECRET_DATA_WITH_SECRET_ENV
      valueFrom:
        secretKeyRef:
          name: key-secret
          key: AAA_SECRET_DATA_WITH_SECRET_ENV
```

Sandbox Scenario Example 4

*참조 : <https://github.com/tmax-cloud/vault/tree/main/example/sandbox-scenario/aaa>

Log 확인

\$ kubectl kubectl logs -f -n aaa vault-test-aaa-6f6dcc5599-k6flv -c alpine

```
Direct Env : aaa_direct_env  
From ConfigMap Env : aaa_cm_env  
From Secret Env : aaa_secret_env  
From Mount File : aaa_mount_file_env  
going to sleep...
```

POD 내부에서 ENV 확인

\$ kubectl exec -it -n aaa vault-test-aaa-6f6dcc5599-k6flv -c alpine env | grep AAA_SECRET

```
AAA_SECRET_DATA_WITH_CM_ENV=vault:secret/data/sandbox/aaa#AAA_SECRET_DATA_WITH_CM_ENV  
AAA_SECRET_DATA_WITH_SECRET_ENV=vault:secret/data/sandbox/aaa#AAA_SECRET_DATA_WITH_SECRET_ENV  
AAA_SECRET_DATA_DIRECT_ENV=vault:secret/data/sandbox/aaa#AAA_SECRET_DATA_DIRECT_ENV
```

프로세스 동작시에만 실제 Value 값이 적용됨을 알수 있음.

/vault/vault-env Binary에 의해서 실시간으로 Vault 서버에 Secret Data 를 요청

Sandbox Scenario Example 5

*참조 : <https://github.com/tmax-cloud/vault/tree/main/example/sandbox-scenario/bbb>

같은 방식으로 bbb에 대해서도 모든 작업을 진행한다.

```
$ kubectl create namespace bbb
$ kubectl create serviceaccount bbb -n bbb
$ vault policy write bbb - <<EOF
path "secret/data/sandbox/bbb" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
EOF
$ vault write auth/kubernetes/role/bbb \
  bound_service_account_names=bbb \
  bound_service_account_namespaces=bbb \
  policies=bbb \
  ttl=2400h
$ vault kv put secret/sandbox/bbb \
BBB_SECRET_DATA_DIRECT_ENV=bbb_direct_env \
BBB_SECRET_DATA_WITH_CM_ENV=bbb_cm_env \
BBB_SECRET_DATA_WITH_SECRET_ENV=bbb_secret_env \
BBB_SECRET_DATA_WITH_MOUNT_FILE=bbb_mount_file_env
$ kubectl apply -f configmap.yaml
$ kubectl apply -f secret.yaml
$ kubectl apply -f consul-template.yaml
$ kubectl apply -f deployment.yaml
```

Deployment Log

```
Direct Env : bbb_direct_env
From ConfigMap Env : bbb_cm_env
From Secret Env : bbb_secret_env
From Mount File : bbb_mount_file_env
```

Sandbox Scenario Example 6

*참조 : <https://github.com/tmax-cloud/vault/tree/main/example/sandbox-scenario/bbb>

\$ kubectl apply -f deployment-fail.yaml

```
labels:
  app.kubernetes.io/name: vault
  app: bbb
annotations:
  vault.security.banzaicloud.io/vault-role: "bbb" # optional, the default value is the name of the ServiceAccount the Pod runs in, in case of Secrets and ConfigMaps it is "default"
  vault.security.banzaicloud.io/vault-skip-verify: "true" # optional, skip TLS verification of the Vault server certificate
  vault.security.banzaicloud.io/vault-path: "kubernetes" # optional, the Kubernetes Auth mount path in Vault the default value is "kubernetes"
spec:
  serviceAccountName: bbb
  containers:
  - name: alpine
    image: alpine
    command: ["sh", "-c", "echo Direct Env : $BBB_SECRET_DATA_DIRECT_ENV && echo going to sleep... && sleep 10000"]
    env:
    - name: AAA_SECRET_DATA_DIRECT_ENV
      value: vault:secret/data/sandbox/aaa#AAA_SECRET_DATA_DIRECT_ENV
```

bbb Service Account 로 aaa가 저장한 데이터의 경로를 알아 내서 읽으려고 한다해도 permission denied가 되는 것을 확인 할수 있다. (Data 격리, 인가)

```
[root@ck172-1 /root/vault/banzaicloud/install-tmax-vault/vault/example/sandbox-scenario/bbb]$ kubectl logs -f -n bbb vault-test-bbb-fail-5476bfd46b-kbdlr
time="2021-11-09T09:41:24Z" level=info msg="received new Vault token" addr= app=vault-env path=kubernetes role=bbb
time="2021-11-09T09:41:24Z" level=info msg="initial Vault token arrived" app=vault-env
time="2021-11-09T09:41:24Z" level=fatal msg="failed to inject secrets from vault: failed to read secret from path: secret/data/sandbox/aaa: Error making API request.\n\nURL: GET https://172.22.6.19:8200/v1/secret/data/sandbox/aaa?version=-1\n\nCode: 403. Errors:\n\n* 1 error occurred:\n\t* permission denied\n\n" app=vault-env
```

Sandbox Scenario Template 적용

< Tmax Template 에 적용 >

- Template 생성시에 사용자가 민감한 정보라고 체크할 수 있게 한다.
- TemplateInstance 생성시에 parameter 중 사용자가 민감한 Data라고 표시한 데이터를 Vault에 사용자의 Path에 저장하고, 해당 값을 Vault 내부 Data의 Path로 치환한다.
- 한계점
 - Template의 Object는 K8S 의 모든 resource를 정의 할수 있음
 - 하지만 Pod 내부의 Env 및 File 로 쓰이는 변수를 제외하고는 Secret Data를 사용할 수 없음
- 제안
 - Template을 만드는 개발자가 민감한 정보로 쓰일 수 있는 parameter를 제한 해야 할 것으로 보임

Q&A