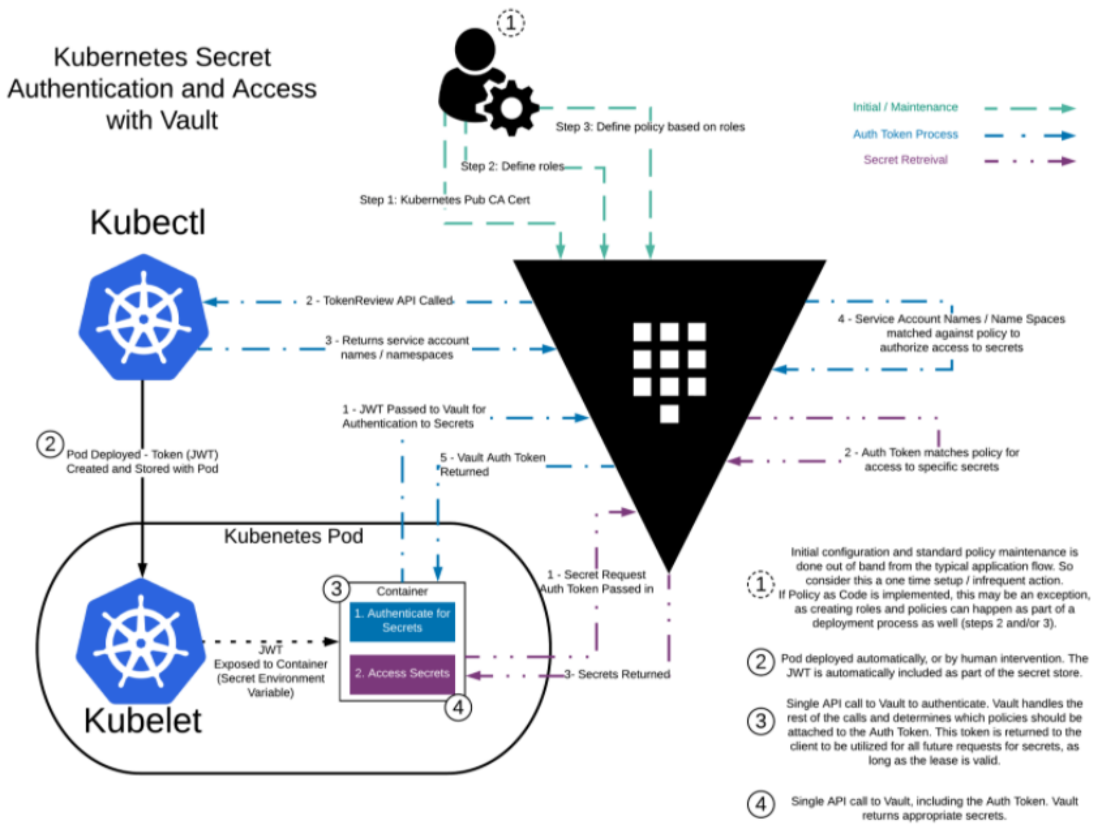
Vault installation

Architecture.

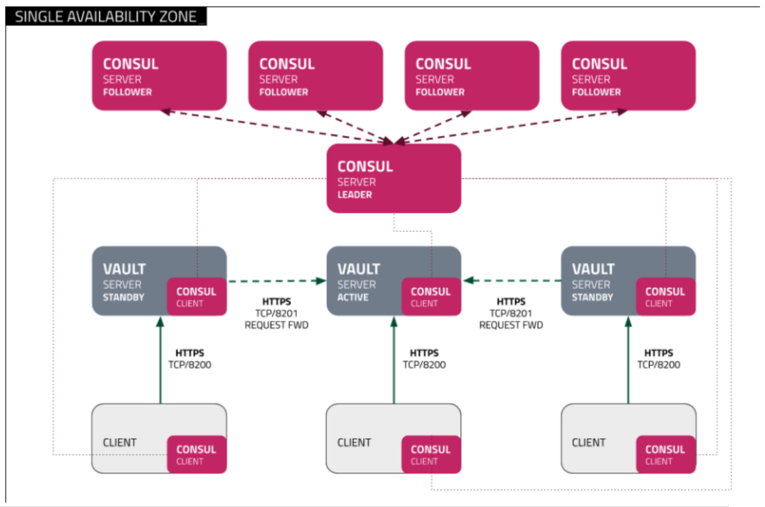

Kubernetes Secret Authentication and Access with Vault
1
Step 3: Define policy based on roles
Step 2: Define roles
Step 1: Kubernetes Pub CA Cert
Initial / Maintenance
Auth Token Process
Secret Retreival
Kubectl
2 - TokenReview API Called
3 - Returns service account names / namespaces
4 - Service Account Names / Name Spaces matched against policy to authorize access to secrets
1 - JWT Passed to Vault for Authentication to Secrets
5 - Vault Auth Token Returned
2 - Auth Token matches policy for access to specific secrets
2
Pod Deployed - Token (JWT) Created and Stored with Pod
Kubenetes Pod
1 - Secret Request Auth Token Passed in
3
Container
1. Authenticate for Secrets
2. Access Secrets
4
JWT Exposed to Container (Secret Environment Variable)
3- Secrets Returned
Kubelet

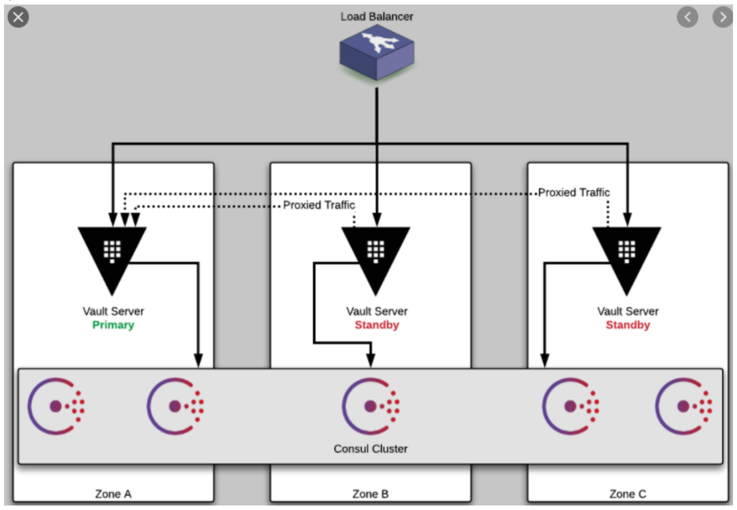
1 Initial configuration and standard policy maintenance is done out of band from the typical application flow. So consider this a one time setup / infrequent action. If Policy as Code is implemented, this may be an exception, as creating roles and policies can happen as part of a deployment process as well (steps 2 and/or 3).

2 Pod deployed automatically, or by human intervention. The JWT is automatically included as part of the secret store.

3 Single API call to Vault to authenticate. Vault handles the rest of the calls and determines which policies should be attached to the Auth Token. This token is returned to the client to be utilized for all future requests for secrets, as long as the lease is valid.

4 Single API call to Vault, including the Auth Token. Vault returns appropriate secrets.





This page will walk step by step on Vault installation on multi k8s cluster environment with consul backend and vault injector.

This installation will require Helm 3 installed on k8s cluster or on local if you are interacting over kubeconfig with clusters.

Install helm on Linux over script

```
$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

Install helm on macOS

```
$ brew install helm
```

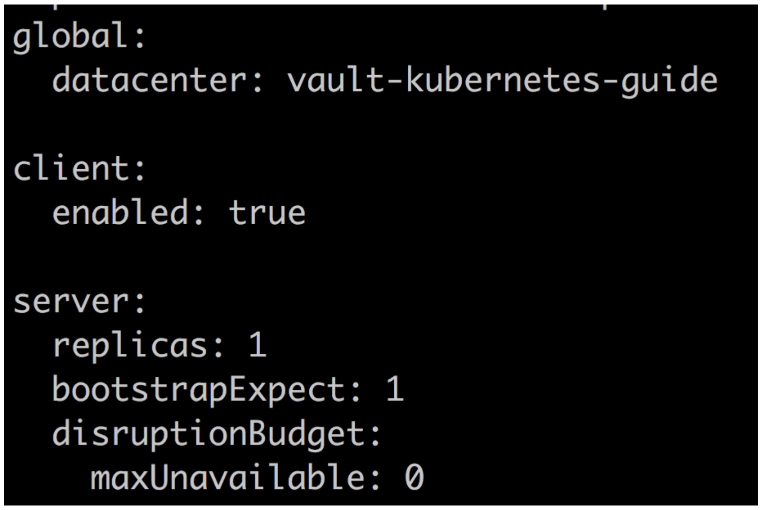
This is 2 repos which have consul, vault helm charts required clone it on your local (modify values.yaml it based on your environment).

https://github.com/hashicorp/vault-helm

https://github.com/hashicorp/consul-helm

Install consul as backend for vault using bellow values.yaml configuration file from consul-helm repo location to the vault server k8s cluster (tools).

```
$ helm install consul -f  helm-consul-values.yaml . -n vault
```



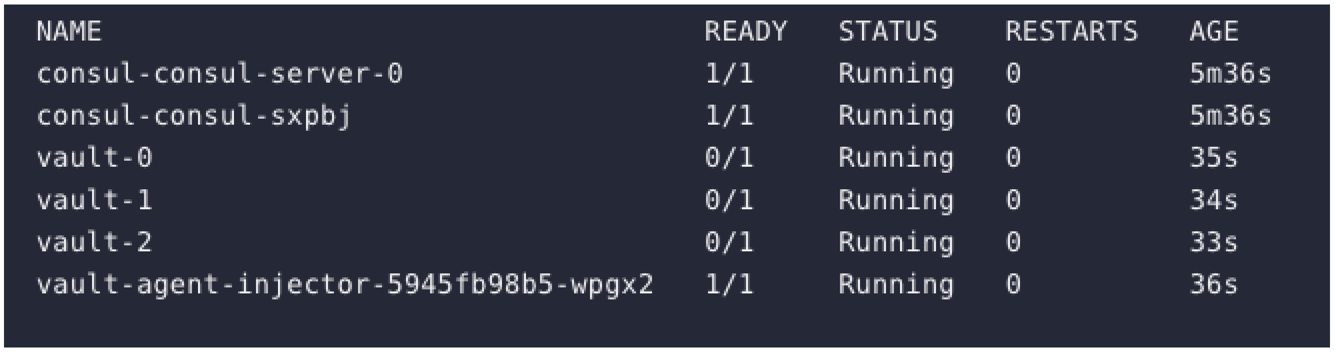
```
global:
  datacenter: vault-kubernetes-guide

client:
  enabled: true

server:
  replicas: 1
  bootstrapExpect: 1
  disruptionBudget:
    maxUnavailable: 0
```

Install vault after you modify values.yaml from vault-helm repo location to the vault server k8s cluster (tools).

```
$ helm install vault -f  values.yaml . -n vault
```

Unseal stage

Get all the pods within the vault namespace.

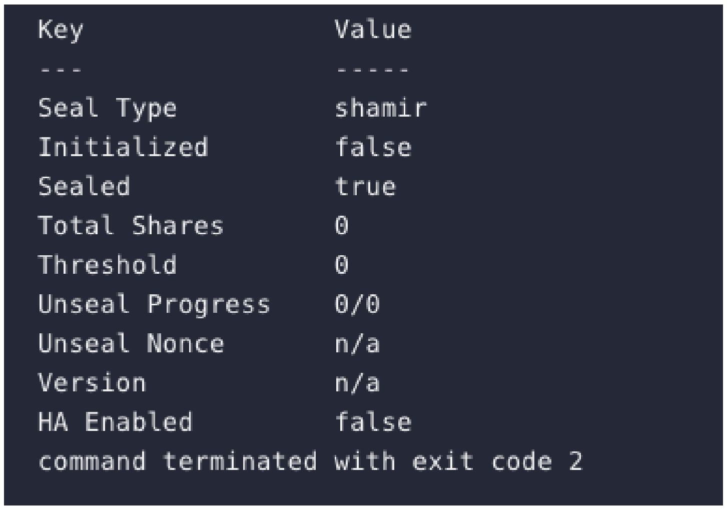


```
NAME                                   READY   STATUS    RESTARTS   AGE
consul-consul-server-0                 1/1     Running   0          5m36s
consul-consul-sxpbj                    1/1     Running   0          5m36s
vault-0                                0/1     Running   0          35s
vault-1                                0/1     Running   0          34s
vault-2                                0/1     Running   0          33s
vault-agent-injector-5945fb98b5-wpgx2  1/1     Running   0          36s
```

The `vault-0`, `vault-1`, `vault-2`, and `vault-agent-injector` pods are deployed. The Vault servers report that they are `Running` but they are not ready (`0/1`). That is because Vault in each pod is executes a status check defined in a readinessProbe.

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

kubectl exec -it vault-0 -n vault -- vault status



```
Key                Value
---                -----
Seal Type          shamir
Initialized        false
Sealed             true
Total Shares       0
Threshold          0
Unseal Progress    0/0
Unseal Nonce       n/a
Version            n/a
HA Enabled         false
command terminated with exit code 2
```

So, let’s initialize the Vault instance (copy output to your notepad there will be unseal keys and root key).

```
# Initialize
$ kubectl exec -it vault-0 -n vault -- vault operator init -n 1 -t 1
```

Install vault injector on external cluster from vault-helm repository location.

$ k create ns vault-injector

$ helm install vault-injector . --set "injector.externalVaultAddr=http://<vault_ip>:8200" -n vault-injector

Finally, let’s unseal the vault so we can use it.

```
# Unseal vault
$ kubectl exec -it vault-0 -n vault -- vault operator unseal <unsealkey>

# Unseal vault $ kubectl exec -it vault-1 -n vault -- vault operator unseal <unsealkey>

# Unseal vault $ kubectl exec -it vault-2 -n vault -- vault operator unseal <unsealkey>
```

Login to the vault

```
$ kubectl exec -it vault-0 -n vault -- vault login <root token>
```

Vault-injector installation and integration with vault server.

$ k create ns vault-injector

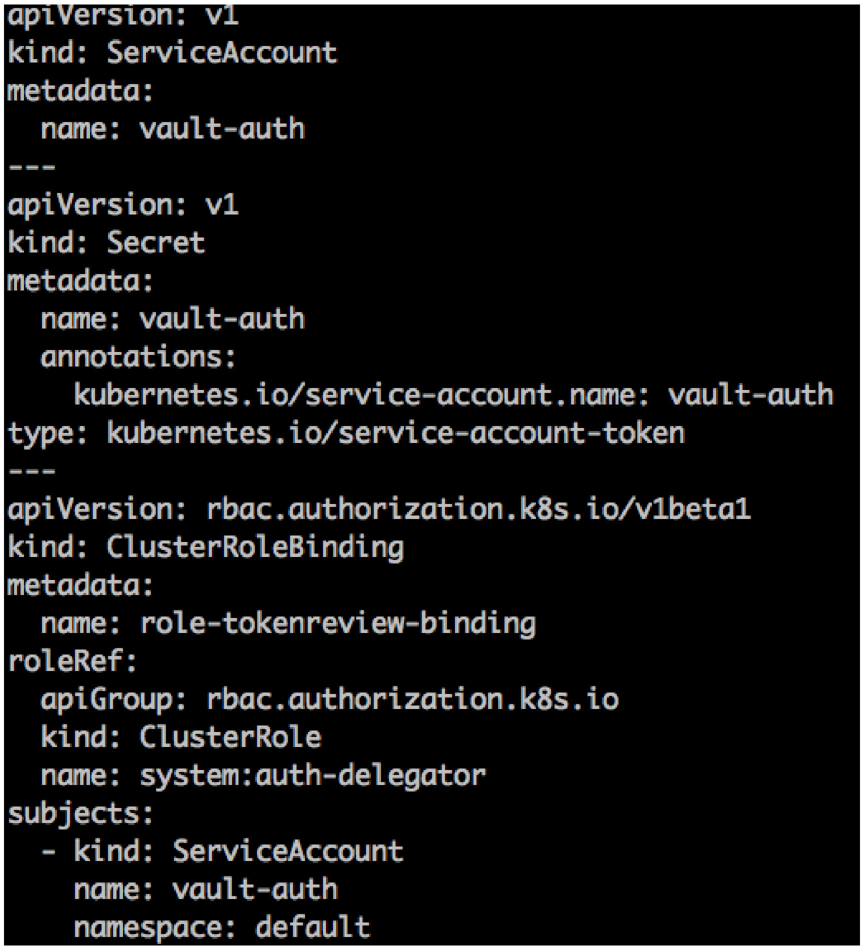
Install vault injector from vault-helm repo location on external cluster.

$helm install vault-injector . --set "injector.externalVaultAddr=http://<vault_ip>:8200" -n vault-injector

»Define a Kubernetes service account on external cluster.

Create a service account, secret, and ClusterRoleBinding with the necessary permissions to allow Vault to perform token reviews with Kubernetes. This SA will be used by vault-server to interract with external kubernetes cluster.

$ k apply -f rbak.yaml



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vault-auth
---
apiVersion: v1
kind: Secret
metadata:
  name: vault-auth
  annotations:
    kubernetes.io/service-account.name: vault-auth
type: kubernetes.io/service-account-token
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: role-tokenreview-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
  - kind: ServiceAccount
    name: vault-auth
    namespace: default
```

Kubernetes Auto-Injector Setup (Vault)

Enable and Configure Kubernetes Auth Method

Run bellow commands on external (vault-injector) cluster.

$ export VAULT_SA_NAME=$(kubectl get sa vault-auth -o jsonpath="{.secrets[*]['name']}")

$ export SA_JWT_TOKEN=$(kubectl get secret $VAULT_SA_NAME -o jsonpath="{.data.token}" | base64 --decode; echo)

$ export SA_CA_CRT=$(kubectl get secret $VAULT_SA_NAME -o jsonpath="{.data['ca\.crt']}" | base64 --decode; echo)

Run bellow commands on vault-server (tools cluster)

Take output from SA_CA_CRT, SA_JWT_TOKEN on vault external cluster and save it inside vault-0 pod on vault server cluster and define bellow variables.

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

$ vi /tmp/SA_CA_CRT

$ vi /tmp/SA_JWT_TOKEN

Variables to define

# determine Kubernetes master IP address (no https://) via `kubectl cluster-info`

$ export K8S_HOST=<K8S_MASTER_IP>

# set VAULT_TOKEN & VAULT_ADDR before next steps

$ export VAULT_ADDR=http://172.29.41.101:8200

$ export VAULT_TOKEN=<root_token>

$ vault auth enable --path="${CLUSTER_NAME}" kubernetes.

example (vault auth enable --path=perfcluster kubernetes)

Enable auth based on variables created

```

vault write auth/kubernetes/config  kubernetes_host="${K8S_HOST}" kubernetes_ca_cert=@/tmp/SA_CA_CRT token_reviewer_jwt=@/tmp/SA_JWT_TOKEN

```

If you will have issues wit accessing check your SA_CA_CRT you might wanna copy it in original content to vault server pod and export it with cat during creation.

https://docs.armory.io/docs/armory-admin/secrets/vault-k8s-configuration/

Read what you created

vault read auth/kubernetes/config

Create Vault policy

```
$ vault policy write internal-app-perf - <<EOH
path "internal/data/database/config" {
  capabilities = ["read"]
}
EOH
```

Create Vault authentication role

```
vault write auth/perfcluster/kubernetes/role/internal-app \
    bound_service_account_names=internal-app \
    bound_service_account_namespaces=default \
    policies=internal-app \
    ttl=24h
```

Note: Vault policy and vault role easier to create from vault UI.

Additional configs:

Method 1. Set sidecar-injector container environment variable

```
- name: AGENT_INJECT_VAULT_AUTH_PATH
  value: "auth/kubernetes"
```

Method 2. Add target pod annotation

```
vault.hashicorp.com/auth-path: "auth/kubernetes"

Istio config:
vault.hashicorp.com/agent-init-first: "true"
```