# Automating PKI Certificate Management with Vault

* Potential Enterprise features to highlight
  + Namespaces
  + Managed keys for CA private keys
  + FIPS
  + Disaster recovery replication
  + Performance replication
  + Perf Standbys for horizontal scaling
  + Cert Metadata
* Existing Resources
  + Generic PKI
    - [HVD Vault PKI](https://developer.hashicorp.com/validated-designs/vault-operating-guides-scaling/pki-secrets-engine)
    - [Generate certificates with HSM or KMS managed keys | Vault | HashiCorp Developer](https://developer.hashicorp.com/vault/tutorials/pki/managed-key-pki)
  + VSO/K8s
    - [Manage Kubernetes native secrets with the Vault Secrets Operator | Vault | HashiCorp Developer](https://developer.hashicorp.com/vault/tutorials/kubernetes/vault-secrets-operator)
    - [Vault Secrets Operator](https://developer.hashicorp.com/vault/docs/platform/k8s/vso/sources/vault)
    - [Vault: Secure secret management using Kubernetes service account metadata](https://developer.hashicorp.com/validated-designs/integration-patterns-guides-vault-kubernetes-auth)
  + Vault Agent
    - [What is Vault Agent?](https://developer.hashicorp.com/vault/docs/agent-and-proxy/agent)
    - [Use Vault Agent templates](https://developer.hashicorp.com/vault/docs/agent-and-proxy/agent/template)
    - [X.509 certificate management with Vault](https://www.hashicorp.com/blog/certificate-management-with-vault)

### Outline

* PKI Setup and Architecture
  + Hierarchy
    - Root CA
      * Should be limited to signing intermediate CSRs
      * Should not be accessible for signing leaf certificates
      * Should be hardware-protected, kept offline, and require multi-party approval for signatures
      * If root CA lives outside of Vault, do not import it into Vault
      * If root CA lives in Vault, type should be kms to protect private key
        + **TODO**: Root CA outside or inside of Vault

Vault Control Groups for Root CA?

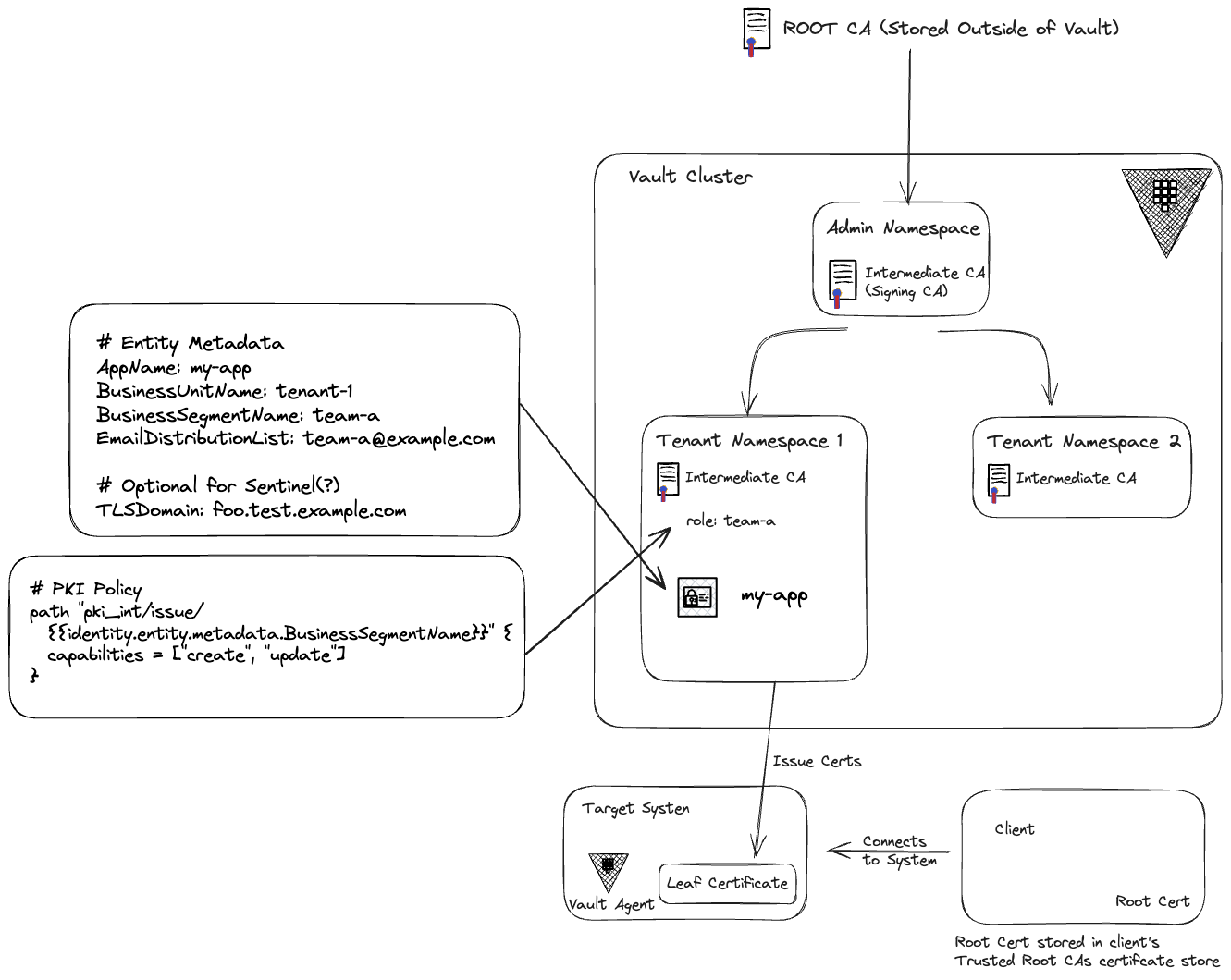
* + - * + **Decision:** Root CA outside of Vault, No Control Groups today as the UX is not great. ACL Policies to guard the Intermediate CA on Vault is sufficient.
    - Create one or more intermediate CAs to issue leaf certificates
      * Should serve their own functional domain
      * Can configure each with a precise security policy
        + Each subordinate CA should serve a single purpose and have corresponding EKUs set (ServerAuth/ClientAuth | CodeSigning | Timestamping)
        + Issuing CAs should have the pathLen attribute set to 0 to disable the issuance of further subordinate CAs
        + Use NameConstraints to minimize the potential for CA abuse (e.g., “.hashicorp.com”, “.hashidemos.io”)
        + **TODO:** Which configuration options do we recommend?
        + **Decision**:
      * Limits blast radius
        + **TODO:** Sentinel Policy for limiting common names, even further, based e.g. on metadata. Sentinel can be used to enforce certain configurations of the PKI engine within the business unit namespaces, like the maximum TTL of certificates, the use of the no\_storeflag, or even the use of allowed hosts, subdomains and wildcards.
        + **Decision:** Try to guard allowed domain names via Sentinel.
      * Scalability and flexibility
        + **TODO:** Store certificates in Vault? By setting the flags no\_store to true and generate\_lease to false, certificates are not stored in Vault, and the requests can be directly handled by Vault performance standby nodes.   
          **Decision:** We will store certs
        + **TODO:** Can we consider using audit logs to track certificates outside of Vault to improve performance of large-scale clusters?  
          **Decision:** Audit logs will not be part of the pattern
    - One PKI secret engine per CA
  + Namespaces
    - Root CA and first levels of intermediate CAs may be placed in a separate Vault namespace from issuing CAs to improve isolation.
      * **TODO:** Following the HVD, we could place the Main/Root Intermediate CA into the “admin” namespace and Intermediate CAs in each “tenant”-namespace
      * **Decision:** admin > tenant-namespaces
  + Key management
    - **TODO:** Managed key information. Probably a good practice also for the Intermediate CAs as it ensures that Vault never sees the private key, enhancing security.  
      **Decision:** Managed Keys on the Admin Namespace
    - **TODO**: Key types should we go with RSA 4096? It seems like RSA keys, especially with higher bit lengths, can significantly impact performance compared to ECDSA or Ed25519 keys.  
      **Decision:**  ECDSA P-256
  + Access controls
    - Access to root CA and first levels of intermediate CAs should be strictly controlled via Vault ACL policies.
      * **TODO:** Use Entity Metadata and Vault ACL Templates to grant access
      * **Decision:** As in the Diagram / test examples
    - Can utilize Vault Enterprise Control groups for major lifecycle operations to prevent malicious or unintentional destructive actions
      * Set signed intermediate
      * Generate root
      * Generate intermediate
      * Sign intermediate
  + Governance and auditing
    - **TODO:** Cluster URLs should be correctly configured, especially if using Performance Replication (AIA information)
    - **TODO:** Plan for distributing CRLs and OCSP responses
    - **TODO:** Automation for CRL tidying
    - **TODO:** Best practices for automatic tidying and clean-up
  + PKI Roles
    - Best practice configuration
    - TTL recommendations
      * **TODO:** Keeping the TTL short is good, but how short?
      * **Decision**: 45 days
* Consumption
  + Vault Agent
    - High-level workflow
      * Create/configure authentication method for application/agent
        + Pre-create entity with metadata?
      * Assign valid ACL policy to allow PKI certificate issuance/signing
      * Create Vault Agent configuration
        + Configure auto-auth
        + Create template
        + Add template configuration to Agent
        + Renewal flow

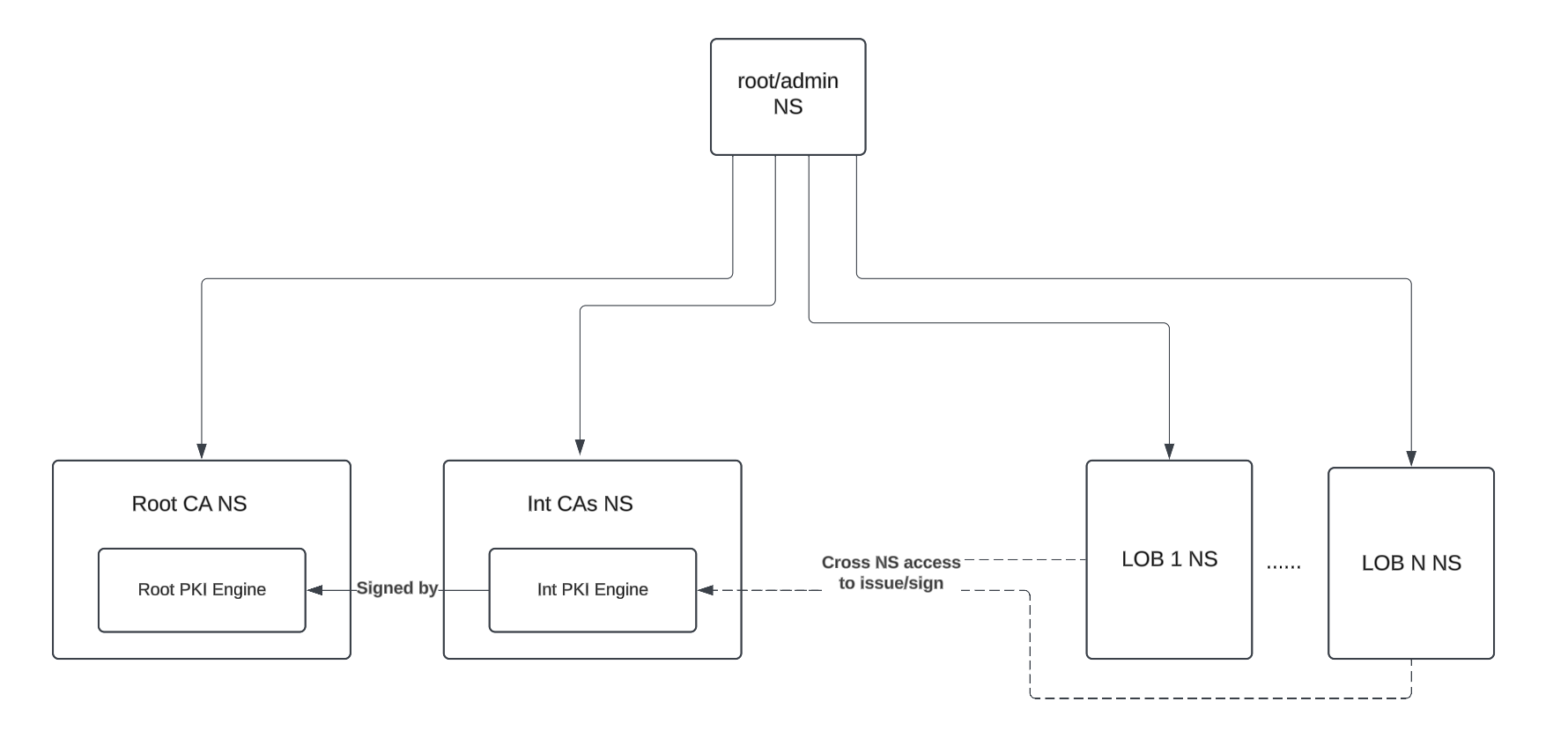
Exec or process supervisor

Control renewal time

* + - * Run Vault Agent
    - Auto-authentication
      * Utilize built-in identity if available (e.g., AWS, GCP, k8s)
      * Keep token TTL low
    - Utilize “pkiCert” template functionality
      * Preferred over the generic secret template
      * Uses existing certificate at file path to avoid re-issuing
      * Does not re-fetch on re-authentication unless the current one has expired
      * Determines refresh interval based on the certificate lifespan as opposed to a lease
      * The writeToFile helper
      * Go through the fields available on the template
        + CA
        + Cert
        + Key
    - Adding metadata to certificate request
      * **TODO:** What data would we store here, and what’s the benefit?
      * **Decision:** Andrew will test this.
    - Controlling refresh time
      * Utilizes “lease\_renewal\_threshold” to determine at what percentage of its lifetime the certificate should be renewed
      * 90 percent by default
      * Set in the template\_config stanza
    - Access controls on private key/certificate
      * Private key should be RW access to owner only
      * Set owner appropriately based on needs
    - Utilize process supervisor mode or the template exec block to trigger a certificate reload
      * Can utilize process supervisor mode to interact with a process that is dependent on the certificate
        + Restart
        + Send signal
      * It is necessary if the application/service utilizing the certificate cannot automatically pick up the new certificate
    - Deployment of Vault Agent?
      * Terraform
  + VSO
    - Namespace structure for K8s use case
    - Connection with Vault
    - Authentication
    - Setting up PKI retrieval

### Diagrams



**Alternative NS Design Proposed by Gui**

### 

### 

### Examples

**PKI cluster-wide Signing CA**

export VAULT\_NAMESPACE=admin

# Managed Keys — AWS

vault write sys/managed-keys/awskms/vault-intermediate-ca-key \

kms\_key="alias/vault-intermediate-ca-key" \

key\_type="ECDSA" \

curve="P384" \

allow\_generate\_key=true



export VAULT\_NAMESPACE=admin

vault secrets enable pki

# maximum time-to-live is 5 years

vault secrets tune \

-allowed-managed-keys=vault-intermediate-ca-key \

-max-lease-ttl=43800h \

pki



vault read sys/mounts/pki/tune

vault write pki/config/cluster \

path=${VAULT\_ADDR}/v1/${VAULT\_NAMESPACE}/pki \

aia\_path=${VAULT\_ADDR}/v1/${VAULT\_NAMESPACE}/pki

vault write pki/config/urls \

issuing\_certificates={{cluster\_aia\_path}}/issuer/{{issuer\_id}}/der \

crl\_distribution\_points={{cluster\_aia\_path}}/issuer/{{issuer\_id}}/crl/der \

ocsp\_servers={{cluster\_path}}/ocsp \

enable\_templating=true

vault write -format=json pki/intermediate/generate/kms \

managed\_key\_name="vault-intermediate-ca-key" \

common\_name="vault.ca.example.com" \

ttl=43800h \

| jq -r '.data.csr' > /tmp/vault-intermediate-ca.csr  
  
cat <<EOF | tee /tmp/openssl\_intermediate.cnf

[ v3\_ca ]

basicConstraints = critical,CA:TRUE,pathlen:1

keyUsage = critical, keyCertSign, cRLSign

subjectKeyIdentifier = hash

authorityKeyIdentifier = keyid:always,issuer

EOF  
  
openssl x509 -req \

-in /tmp/vault-intermediate-ca.csr \

-CA rootCA.crt \

-CAkey rootCA.key \

-CAcreateserial \

-out /tmp/vault-intermediate-ca.crt \

-days 1825 \

-sha256 \

-extfile /tmp/openssl\_intermediate.cnf \

-extensions v3\_ca  
  
# import back to vault

vault write pki/intermediate/set-signed \

certificate=@/tmp/vault-intermediate-ca.crt

vault write -field=imported\_issuers pki/intermediate/set-signed \

certificate=@/tmp/vault-intermediate-ca.crt

vault read pki/config/urls

vault list pki/issuers/

# name the issuer

vault write pki/issuer/default \

issuer\_name="vault-intermediate-ca"



**PKI Issuing CA per tenant namespace**

export VAULT\_NAMESPACE=admin/tenant-1

# Managed Keys — AWS

vault write sys/managed-keys/awskms/tenant-1-issuing-ca-key \

kms\_key="alias/tenant-1-issuing-ca-key" \

key\_type="ECDSA" \

curve="P384" \

allow\_generate\_key=true



export VAULT\_NAMESPACE=admin/tenant-1

vault secrets enable pki

# maximum time-to-live is 2.5 years

vault secrets tune \

-allowed-managed-keys=tenant-1-issuing-ca-key \

-max-lease-ttl=21900h \

pki

vault read sys/mounts/pki/tune

vault write pki/config/cluster \

path=${VAULT\_ADDR}/v1/${VAULT\_NAMESPACE}/pki \

aia\_path=${VAULT\_ADDR}/v1/${VAULT\_NAMESPACE}/pki

vault write pki/config/urls \

issuing\_certificates={{cluster\_aia\_path}}/issuer/{{issuer\_id}}/der \

crl\_distribution\_points={{cluster\_aia\_path}}/issuer/{{issuer\_id}}/crl/der \

ocsp\_servers={{cluster\_path}}/ocsp \

enable\_templating=true

vault read pki/config/urls



export VAULT\_NAMESPACE=admin/tenant-1

vault write -format=json pki/intermediate/generate/kms \

managed\_key\_name="tenant-1-issuing-ca-key" \

common\_name="tenant-1.ca.example.com" \

ttl="21900h" \

| jq -r '.data.csr' > /tmp/tenant-1-issuing-ca.csr

export VAULT\_NAMESPACE=admin

vault write -format=json pki/issuer/default/sign-intermediate \

csr=@/tmp/tenant-1-issuing-ca.csr \

format=pem\_bundle \

max\_path\_length="0" \

ttl="21900h" \

| jq -r '.data.certificate' > /tmp/tenant-1-issuing-ca.cert.pem

export VAULT\_NAMESPACE=admin/tenant-1

# import back to vault

vault write pki/intermediate/set-signed \

certificate=@/tmp/tenant-1-issuing-ca.cert.pem

# name the issuer

vault write pki/issuer/default \

issuer\_name="tenant-1-issuing-ca"



cat <<EOF | tee /tmp/pki.sentinel

## this policy is to restrict the common name while issuing a pki cert

import "strings"

# Only care about write and update operations against pki/issue

precond = rule {

request.operation in ["write", "update"] and

strings.has\_prefix(request.path, "pki/issue")

}

# check if the trusted orchestrator makes the request

trusted\_orchestrator\_check = func() {

print ("trace:identity.entity.name", identity.entity.name)

# check identity

if identity.entity.name matches "terraform" {

return true

}

return false

}

# check common\_name matches the entity metadata

common\_name\_check = func() {

print ("trace:Request.data:", request.data)

print ("trace:TLSDomain", identity.entity.metadata.TLSDomain)

# Make sure there is request data

if length(request.data else 0) is 0 {

return false

}

# Make sure request data includes common\_name

if length(request.data.common\_name else 0) is 0 {

return false

}

# check common\_name matches app name

if request.data.common\_name matches identity.entity.metadata.TLSDomain {

return true

}

return false

}

# Check the precondition before executing all above functions

main = rule when precond {

trusted\_orchestrator\_check() or common\_name\_check()

}

EOF



POLICY=$(base64 -i /tmp/pki.sentinel)

vault write sys/policies/egp/restrict-common-name \

policy="${POLICY}" \

paths="pki/issue/team-a" \

enforcement\_level="hard-mandatory"

vault read sys/policies/egp/restrict-common-name



**AWS Auth**

export VAULT\_NAMESPACE=admin/tenant-1

tee /tmp/pki-policy.hcl <<EOF

path "pki/issue/{{identity.entity.metadata.BusinessSegmentName}}" {

capabilities = ["create", "update"]

}

EOF

vault policy write pki /tmp/pki-policy.hcl

vault auth enable aws

vault write -force auth/aws/config/client

# Configure Vault client source

vault write auth/aws/config/identity \

iam\_alias="canonical\_arn"

vault write auth/aws/role/jrx-test-role \

auth\_type=iam \

bound\_iam\_principal\_arn="arn:aws:iam::143332470013:role/jrx-test-role" \

policies=pki \

ttl=24h



**Entities**

export VAULT\_NAMESPACE=admin/tenant-1

vault write -format=json identity/entity \

name="my-app" \

metadata=AppName="my-app" \

metadata=BusinessUnitName="tenant-1" \

metadata=BusinessSegmentName="team-a" \

metadata=EmailDistributionList="team-a@example.com" \

metadata=TLSDomain="foo.tenant-1.example.com" \

| jq -r ".data.id" > /tmp/entity\_id.txt

vault auth list -format=json \

| jq -r '.["aws/"].accessor' \

> /tmp/accessor\_aws.txt

vault write identity/entity-alias \

name="arn:aws:iam::143332470013:role/jrx-test-role" \

canonical\_id=$(cat /tmp/entity\_id.txt) \

mount\_accessor=$(cat /tmp/accessor\_aws.txt)



**Vault Agent Configuration**

pid\_file = "/opt/vault/pidfile"

auto\_auth {

method "aws" {

mount\_path = "auth/aws"

config = {

type = "iam"

role = "jrx-test-role"

}

}

}

vault {

address = "{{ VAULT\_ADDR }}"

namespace = "{{ VAULT\_NAMESPACE }}"

ca\_cert = "/etc/vault.d/vault-ca.pem"

}

template {

source = "/etc/vault.d/certificate.tpl"

destination = "/opt/vault/vault-agent-tmp.pem"

command = "nginx -s reload"

}

####  **Vault Agent Template for Rendering PKI**

# certificate.tpl  
{{- with pkiCert "pki/issue/team-a" "common\_name=foo.tenant-1.example.com" "ttl=45d" -}}

{{- .Key -}}

{{- .Cert -}}

{{- .Key | writeToFile "templates/output/private.key" "" "" "0600" -}}

{{- .Cert | writeToFile "templates/output/server.crt" "" "" "0644" -}}

{{- range .CAChain -}}

{{- . -}}

{{- . | writeToFile "templates/output/server.crt" "" "" "0644" "append" -}}

{{- end -}}

{{- end -}}



#### **VSO**

* https://github.com/jrx/tf-vault-kubernetes-auth/blob/main/app-pki.tf

apiVersion: v1

automountServiceAccountToken: true

kind: ServiceAccount

metadata:

annotations:

vault.hashicorp.com/alias-metadata-AppName: my-app

vault.hashicorp.com/alias-metadata-BusinessSegmentName: team-a

vault.hashicorp.com/alias-metadata-BusinessUnitName: tenant-1

vault.hashicorp.com/alias-metadata-TLSDomain: one.test.example.com

name: my-app

namespace: app-1

--

apiVersion: secrets.hashicorp.com/v1beta1

kind: VaultAuth

metadata:

name: static-auth

namespace: app-1

spec:

kubernetes:

audiences:

- https://kubernetes.default.svc

role: my-app

serviceAccount: my-app

tokenExpirationSeconds: 600

method: kubernetes

mount: kubernetes

namespace: admin/tenant-1/

--

apiVersion: secrets.hashicorp.com/v1beta1

kind: VaultPKISecret

metadata:

name: vault-pki-app

namespace: app-1

spec:

commonName: one.test.example.com

destination:

create: true

name: secretpki

overwrite: false

expiryOffset: 10s

format: pem

mount: pki\_int

namespace: admin/tenant-1/

revoke: true

role: team-a

ttl: 120s

vaultAuthRef: static-auth

