

# Lunch and Learn Migrate from Vault OSS to Enterprise





#### **Agenda**

- In-place Migration
- Migration to New Vault Cluster
- Automate Vault Configuration
- Resources

### In-Place Migration



#### **Overview**



The most common path for migrating an existing Vault Open Source cluster to Vault Enterprise is via in-place migration. In-place migration follows our standard upgrade procedure by simply replacing the existing Vault Open Source binary with the Vault Enterprise version.



#### In-Place Migration Process

- 1. Backup Vault Cluster
- 2. Identify Leader Node
- 3. Replace binary on follower node
- 4. Add licensing configuration to follower node
- 5. Repeat on all follower nodes
- 6. Replace binary and add licensing to leader node



1. Backup

Consul Storage Backend







> vault operator raft snapshot save vault-oss.snapshot



#### 1. Backup

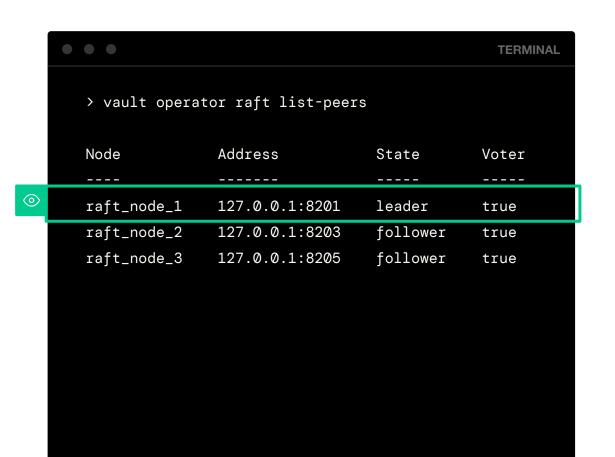
Integrated Storage



## 2. Identify Leader Node

Consul Storage Backend

```
TERMINAL
> curl $VAULT_ADDR/v1/sys/leader
 "ha enabled": true,
 "is self": false,
 "leader address": "https://172.10.16.50:8200/",
 "leader cluster address": "https://172.10.16.50:8201/",
 "performance standby": false,
```





## 2. Identify Leader Node

Integrated Storage

#### 3. Upgrade Binary on Follower Nodes



```
TERMINAL
 Stop Vault on Follower node
 systemctl stop vault
 Download ENT Binary
 wget
 Replace existing Vault binary and then validate binary version
 vault -v
Vault v1.8.4+ent (93fd4b9f7c118deaebf30f250e8be626e1121b80)
 STOP - Do not start Vault yet proceed to step 4 for licensing
```

#### 4. Add Vault License on Follower Nodes <sup>印</sup>



```
CODE EDITOR
# Three methods to autoload license, same should be used across all nodes
     Update configuration file with license_path parameter
     License_path = "/ect/vault.d/license.hclic"
     Provide license path via environment variable
     export VAULT_LICENSE_PATH = "/ect/vault.d/license.hclic"
     Provide license as a string in environment variable
     export VAULT_LICENSE = "02MV4UU43BK5HGYYT0JZ..."
```

#### 5. Start Vault on Follower Nodes



```
TERMINAL
> systemctl start vault
# Manually unseal node if not using an auto seal
> vault operator unseal <unseal_key>
# Check Vault Status
> vault status
# Verify logs are not outputting an errors
> journalctl -u vault
# Repeat steps 1 - 5 on any remaining follower nodes before proceeding to
step 6
```

# 6. Repeat steps 1 - 5 on leader once all followers have migrated successfully

### Migration to New Cluster



#### **Overview**



While most Vault customers perform in-place migrations to Vault Enterprise, you may also be considering a fresh start with your Vault Enterprise deployment.

Currently, Vault does not have built-in migration to move data from one Vault cluster to another. However, you can automate the migration using Vault's API or tooling developed by the community.



## Static Secrets

Export static secrets from current cluster and import from CSV.

```
set -e
COMMAND="vault kv put kv-v1/sample"
while IFS="," read -r key value
   COMMAND="$COMMAND $key=$value"
done < secrets.csv</pre>
eval $COMMAND
```



#### **Policies**

Export policies from current cluster and import from CSV.

```
#ignores first line
 read -r
 while IFS="," read -r name file
   vault policy write "$name" "$file"
} < policy-names.csv</pre>
```



#### **Transit Keys**

Transit keys can only be exported if they had initially been created with exportable set to true.

```
KEYS=$(vault list -format=json transit/keys
| jq .[] | sed 's/"//q')
for key in $KEYS
 vault write transit/keys/"$key"/config
allow plaintext backup=true exportable=true
 vault read -format=json
transit/backup/"$key" | jq .data >
backups/"$key"-backup.json
#Run against new cluster
#!/bin/bash
for file in backups/*.json
  vault write transit/restore @"$file"
```

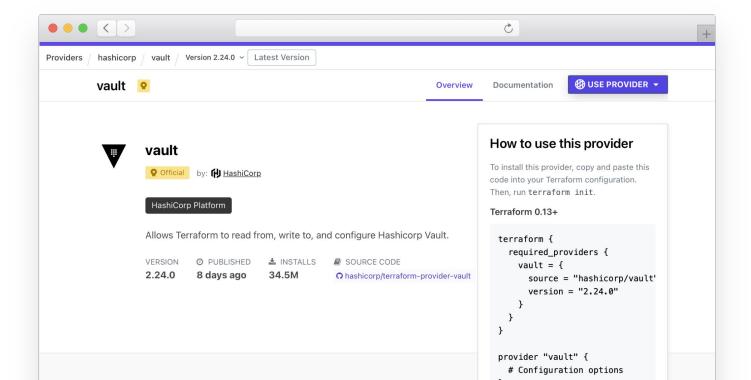
## Automate Vault Configuration



#### **Vault Provider**



Provision namespaces, policies, secrets engines, and auth methods



```
resource "vault_namespace" "infosec" {
 path = "infosec"
}
provider vault {
 alias
           = "infosec"
 namespace = vault_namespace.infosec.path
resource "vault_policy" "example" {
```

provider = vault.infosec



Namespace and Provider Alias



#### **Create Policy**

Create auth method for OIDC provider.

```
data "vault_policy_document" "dev_user_policy" {
   rule {
                     = "secret/data/development/*"
       path
       capabilities = ["create", "read", "update",
"delete", "list"]
resource "vault_policy" "devusers" {
          = "dev-policy"
   name
   policy = "${data.vault_policy_document.hcl}"
```



## **Enable User Auth Method**

Create auth method for OIDC provider.

```
resource "vault_jwt_auth_backend" "oidcauth" {
   description
                       = "Auth@ OIDC"
   path
                       = "oidc"
                       = "oidc"
   type
   oidc_discovery_url = "https://myco.auth0.com/"
   oidc_client_id
                       = "1234567890"
   oidc_client_secret = "secret123456"
   bound_issuer
                       = "https://myco.auth0.com/"
   tune {
       listing_visibility = "unauth"
```

```
resource "vault_jwt_auth_backend_role" "example" {
 backend
                 = vault_jwt_auth_backend.oidc.path
                 = "test-role"
 role_name
 token_policies = ["default", "dev", "prod"]
 user_claim
                       = "https://vault/user"
 role_type
                       = "oidc"
 allowed_redirect_uris =
["http://localhost:8200/ui/vault/auth/oidc/oidc/callbac
k"]
```



## **Create Auth Role**

Role will define the user claim to authenticate a user and which policy assignments they have in Vault.



#### Enable Secrets Engines

```
resource "vault_mount" "kvv2-infosec" {
                                 = "infosec"
 path
                                 = "kv-v2"
 type
resource "vault_mount" "pki-dev" {
 path
                                 = "pki-dev"
                                 = "pki"
 type
 default_lease_ttl_seconds
                                 = 3600
 max_lease_ttl_seconds
                                 = 86400
```

#### **Best Practices**



#### **Protect State**

Terraform, by default, stores state in the working directory where Terraform CLI is executed. Remote State should be used and encrypted. Access to state should be limited by following practice of least privilege.

#### Manage as Code

Treat Terraform configuration files as code. Store in a VCS code. Pass any secrets, such like Github and practice least as credentials or Vault token privilege for access and who can commit changes. Integrate into CI process and ensure code is tested in dev before pushing to production.

#### **Sensitive Values**

Do not put any secrets in by using environment variables. Sensitive values may appear in state if not handled correctly.

#### Resources





#### Resources

- Vault Upgrade Standard Procedure | Vault
- Vault Data Backup Standard Procedure | Vault
- Upgrading Vault Guides
- <u>License Autoloading</u>
- Terraform Registry Vault Provider
- Related Tools



#### Thank You

customer.success@hashicorp.com www.hashicorp.com