HashiCorp Vault

# Migrate from Vault OSS to Enterprise

August, 2022

# Agenda

- In-Place Migration
- Storage Migration
- Migration to New Vault Cluster
- Automate Vault Configuration
- Resources

01

# In-Place Migration

# Overview

1. The most common path for migrating an existing Vault Open Source cluster to Vault Enterprise is **via in-place migration**.
2. 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

TERMINAL

```
$ consul operator raft snapshot
```

TERMINAL

```
$ 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,
 "performance_standby_last_remote_wal": 0
}
```

TERMINAL

```
$ vault operator raft list-peers

Node            Address            State       Voter
----            -------            -----       -----
raft_node_1     127.0.0.1:8201     leader      true
raft_node_2     127.0.0.1:8203     follower    true
raft_node_3     127.0.0.1:8205     follower    true
```

# 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
https://releases.hashicorp.com/vault/1.11.2+ent/vault_1.11.2+ent_freebsd_amd64.zip

# Replace existing Vault binary and then validate binary version
$ vault -v
Vault v1.11.2+ent
(cb51bfbd015e4f5ea2fe5b49b28f2d6035229638cc20438b4ba8308ef926c0e7)

# STOP - Do not start Vault yet proceed to step 4 for licensing
```

# 4. Add Vault License on Follower Nodes

CODE EDITOR

```
# Three methods to autoload license, same should be used across all nodes
1.   Update configuration file with license_path parameter

     License_path = "/ect/vault.d/license.hclic"

2.   Provide license path via environment variable

     export VAULT_LICENSE_PATH = "/ect/vault.d/license.hclic"

3.   Provide license as a string in environment variable

     export VAULT_LICENSE = "02MV4UU43BK5HGYYTOJZ…"
```

# 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 been migrated successfully

02

# Storage Migration

# Vault Enterprise

Vault Enterprise supports two storage backends:

- **Integrated Storage**
- **Consul Storage**

If using an OSS supported storage backend you will need to migrate storage **prior** to upgrading to Vault Enterprise

# Storage Migration

- Vault's "`operator migrate`" command copies data between storage backends
- Operates directly at the storage level, with no decryption involve
- Destination **should not** be initialized prior to the migrate operation
- Source data is not modified, except for a lock key added during migration
- This is an offline operation for data consistency, thus requires downtime

# Storage Migration Process

1. Backup Vault Cluster
2. Create migration configuration file
3. Identify node to use for migration operation
4. Stop Vault
5. Run the migration
6. Update Vault configuration file(s)
7. Start and unseal Vault
8. Join additional nodes

CODE EDITOR

```
# migrate.hcl
storage_source "mysql" {
 username = "user1234"
 password = "secret123!"
 database = "vault"
}

storage_destination "raft" {
 path = "/path/to/raft/data"
 node_id = "raft_node_1"
}
cluster_addr = "http://192.168.72.10:8201"
```

# Example Migration Configuration File

# Migration Operation

TERMINAL

```
$ vault operator migrate -config migrate.hcl



2018-09-20T14:23:23.656-0700 [INFO ] copied key:
data/core/seal-config

2018-09-20T14:23:23.657-0700 [INFO ] copied key:
data/core/wrapping/jwtkey

2018-09-20T14:23:23.658-0700 [INFO ] copied key:
data/logical/fd1bed89-ffc4-d631-00dd-0696c9f930c6/31c8e6d
9-2a17-d98f-bdf1-aa868afa1291/archive/metadata

2018-09-20T14:23:23.660-0700 [INFO ] copied key:
data/logical/fd1bed89-ffc4-d631-00dd-0696c9f930c6/31c8e6d
9-2a17-d98f-bdf1-aa868afa1291/metadata/5kKFZ4YnzgNfy9UcWO
zxxzOMpqlp61rYuq6laqpLQDnB3RawKpqi7yBTrawj1P

...
```

03

# 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.

CODE EDITOR

```
#!/bin/bash
set -e

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

# Policies

Export policies from current cluster and import from CSV.

CODE EDITOR

```
#!/bin/bash
{
  #ignores first line
  read -r
  while IFS="," read -r name file
  do
    vault policy write "$name" "$file"
  done
} < policy-names.csv
```

# Transit Keys

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

CODE EDITOR

```
#Run against current cluster
#!/bin/bash
KEYS=$(vault list -format=json transit/keys
| jq .[] | sed 's/"//g')
for key in $KEYS
do
 vault write transit/keys/"$key"/config
allow_plaintext_backup=true exportable=true
 vault read -format=json
transit/backup/"$key" | jq .data >
backups/"$key"-backup.json
done

#Run against new cluster
#!/bin/bash
for file in backups/*.json
do
  vault write transit/restore @"$file"
done
```

04

# Automate Vault Configuration

# Vault Provider

## Provision namespaces, policies, secrets engines, and auth methods

Providers / hashicorp / vault / Version 2.24.0 Latest Version

vault

Overview Documentation USE PROVIDER

vault

Official by: HashiCorp

HashiCorp Platform

Allows Terraform to read from, write to, and configure Hashicorp Vault.

| VERSION | PUBLISHED | INSTALLS | SOURCE CODE |
| --- | --- | --- | --- |
| 2.24.0 | 8 days ago | 34.5M | hashicorp/terraform-provider-vault |

### How to use this provider

To install this provider, copy and paste this code into your Terraform configuration. Then, run `terraform init`.

Terraform 0.13+

```
terraform {
  required_providers {
    vault = {
      source = "hashicorp/vault"
      version = "2.24.0"
    }
  }
}

provider "vault" {
  # Configuration options
```

CODE EDITOR

```
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.

CODE EDITOR

```
data "vault_policy_document" "dev_user_policy" {
    rule {
        path         = "secret/data/development/*"
        capabilities = ["create", "read", "update",
"delete", "list"]
    }
}

resource "vault_policy" "devusers" {
    name   = "dev-policy"
    policy = "${data.vault_policy_document.hcl}"
}
```

# Enable User Auth Method

Create auth method for OIDC provider.

CODE EDITOR

```
resource "vault_jwt_auth_backend" "oidcauth" {
    description         = "Auth0 OIDC"
    path                = "oidc"
    type                = "oidc"
    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"
    }
}
```

CODE EDITOR

```
resource "vault_jwt_auth_backend_role" "example" {
  backend         = vault_jwt_auth_backend.oidc.path
  role_name       = "test-role"
  token_policies  = ["default", "dev", "prod"]

  user_claim            = "https://vault/user"
  role_type             = "oidc"
  allowed_redirect_uris =
["http://localhost:8200/ui/vault/auth/oidc/oidc/callback"]
}
```

# Create Auth Role

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

# Enable Secrets Engines

CODE EDITOR

```
resource "vault_mount" "kvv2-infosec" {
  path                          = "infosec"
  type                          = "kv-v2"
}

resource "vault_mount" "pki-dev" {
  path                          = "pki-dev"
  type                          = "pki"
  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 like Github and practice least 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 code. Pass any secrets, such as credentials or Vault token by using environment variables. Sensitive values may appear in state if not handled correctly.

05

# Resources

# Resources

- Vault Upgrade Standard Procedure
- Vault Data Backup Standard Procedure
- Upgrading Vault - Guides
- Operator Migrate
- License Autoloading
- Terraform Registry Vault Provider
- Related Tools

# Thank You

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