

Migrate from Vault OSS to Enterprise





Agenda

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

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.



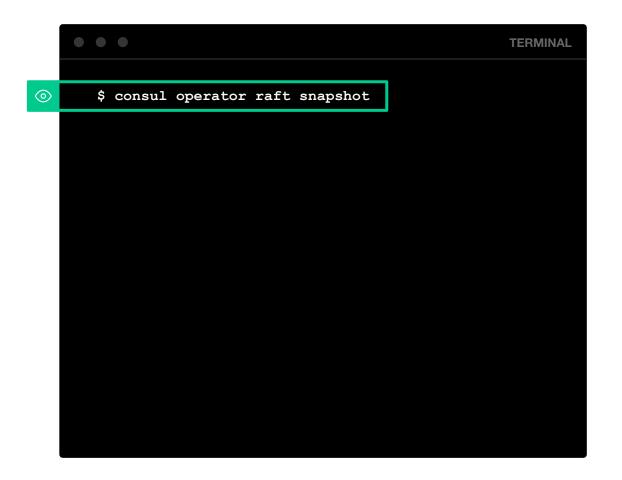
In-Place Migration Process

- 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







\$ vault operator raft snapshot save vault-oss.snapshot



1. Backup

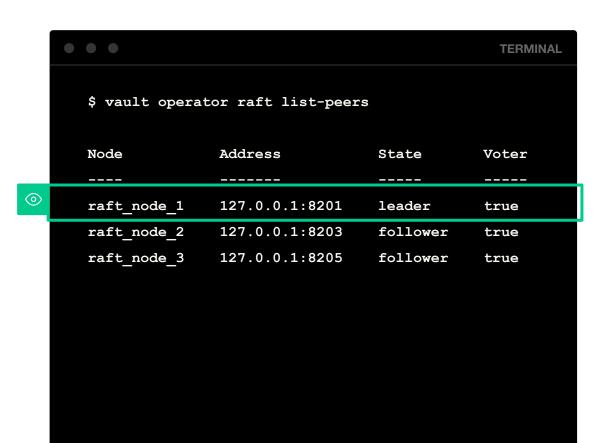
Integrated Storage



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
```





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
S 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 ^印



```
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 = "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



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

- 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

```
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/5kKFZ4YnzqNfy9UcWO
zxxzOMpqlp61rYuq6laqpLQDnB3RawKpqi7yBTrawj1P
```

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.

```
#!/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</pre>
eval $COMMAND
```



Policies

Export policies from current cluster and import from CSV.

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



Transit Keys

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

```
#Run against current cluster
#!/bin/bash
KEYS=$(vault list -format=json transit/keys
| jq .[] | sed 's/"//q')
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
```

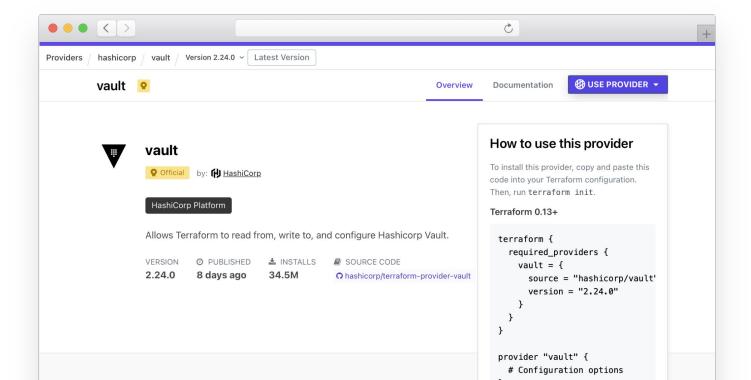
Automate Vault Configuration



Vault Provider



Provision namespaces, policies, secrets engines, and auth methods



```
resource "vault namespace" "infosec" {
 path = "infosec"
provider vault {
           = "infosec"
  alias
  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
                       = "Auth0 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"
```

CODE EDITOR

```
resource "vault_jwt auth backend role" "example" {
                 = vault jwt auth backend.oidc.path
 backend
 role name
                 = "test-role"
  token policies = ["default", "dev", "prod"]
                       = "https://vault/user"
 user claim
 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

```
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.

Resources





Resources

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



Thank You

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