HashiCorp Vault

Operationalizing Vault Enterprise

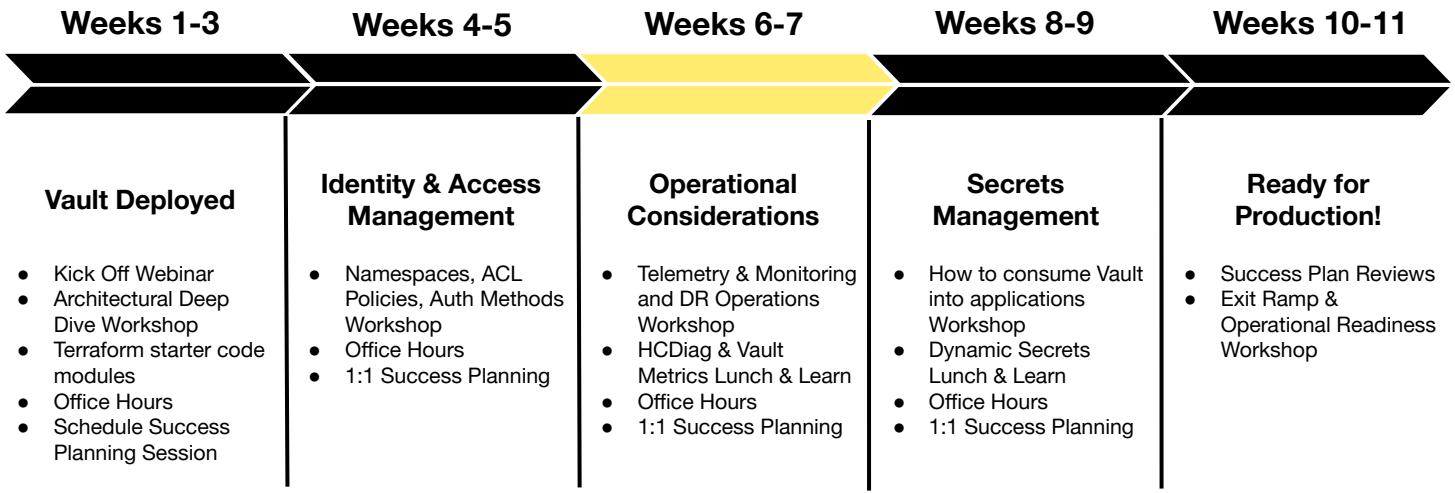
December 2021

Copyright © 2021 HashiCorp

Agenda

- Telemetry Monitoring
- DR Operations
- Vault Runbooks
- Next Steps
- Q & A

Vault Enterprise Path to Production



01

Monitoring

Telemetry and Health Monitoring

Monitoring Patterns

Organizations that have successfully adopted Vault at scale typically classify Vault as a tier 0 application as it is typically a dependency for their most critical applications. Below are the three patterns that should be adopted for monitoring the health of Vault.

1. Time-series telemetry data
2. Log Analytics
3. Active Health Checks

Vault Telemetry

Vault uses the Golang go-metrics package to export Telemetry Metrics to an upstream service, specified in the telemetry stanza. Run Time Metrics are aggregated across a 10 second interval and retained in memory for 1 minute.

Supported sinks include:

- Statsite
- Statsd
- Circonus
- Dogstatsd
- Prometheus

Metric Types

[C] Counter

Cumulative metrics that increment when an event occurs and are reset at the end of the reporting interval.

[G] Gauge

Provides measurements of current values

[S] Summary

Provide sample observations of values. Commonly used to measure timing duration of discrete events in the reporting interval.

Enable Telemetry

Specify upstream monitoring service in telemetry stanza in Vault Server configuration

Telemetry - Configuration

CODE EDITOR

```
telemetry {
  statsd_address = "statsd.company.local:8125"
}
```

Contributing Factors in Performance

- Know the expected workload
- Vault System Resources (CPU, MEM, Disk)
- Complexity of the Vault Policies
- Audit Logging
- Network for all the things

Vault CPU

Select the host VMs to handle the concurrent workload

Audit Device

Be sure that the write location won't become the bottleneck

Policies

Never run load test using root token

Network

Know all the systems that are involved and connectivity between them

Storage Backend

- Determine appropriate TTLs for tokens and leases
- Leverage batch tokens and Vault Agent Caching

Key System Metrics

Metric	Description	What to look for?
cpu.user_cpu	Percentage of CPU being used by user processes	Look for high Vault CPU consumption
cpu.iowait_cpu	Percentage of CPU time spent waiting for I/O tasks to complete	Look for `cpu.iowait_cpu` greater than 10%
net.bytes_recv	Bytes received on each network interface	Look for sudden large changes to the net metrics (greater than 50% deviation from baseline)
net.bytes_sent	Bytes transmitted on each network interface	
linux_sysctl_fs.file-nr	Number of file handles being used across all processes on the host	If the `file-nr` reaches 80% of the `file-max` then you should alert and investigate
linux_sysctl_fs.file-max	Total number of available file handles	

Key Integrated Storage Metrics

Metric	Description	What to look for?
vault.runtime.total_gc_pause_ns	Number of nanoseconds consumed by stop-the-world garbage collection (GC) pauses since Vault started	This would be considered a _warning_ if `total_gc_pause_ns` exceeds 2 seconds/minute and _critical_ if it exceeds 5 seconds/minute
vault.raft.leader.lastContact`	Time to retrieve a value for a path	Look for candidate > 0, or leader > 0, or `lastContact` greater than 200ms which indicates that consensus is unhealthy
vault.raft.state.candidate	Time to insert a log entry to the persist path	
vault.raft.state.leader	This increments whenever a raft server becomes a leader	
diskio.read_bytes	Bytes read from each block device	You will want to monitor for large changes to the diskio metrics for greater than 50% deviation from baseline, or more than 3 deviations from your standard baseline. Then you will want to monitor for over 80% utilization on block device mount points on which Vault data are persisted.
diskio.write_bytes	Bytes written to each block device	
disk.used_percent	Per-mount-point block device utilization	

Key Usage Metrics

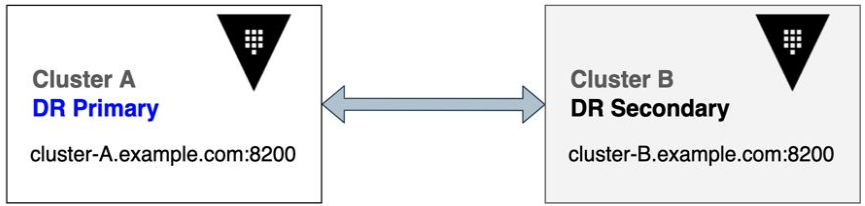
Metric	Description
vault.token.creation	A new service or batch token was created
vault.token.count	Number of service tokens available for use.
vault.token.count.by_auth	Number of existing tokens broken down by the auth method used to create them.
vault.token.count.by_policy	Number of existing tokens, counted in each policy assigned.
vault.token.count.by_ttl	Number of existing tokens, aggregated by their TTL at creation.
vault.secret.kv.count	Count of secrets in key-value stores.
vault.secret.lease.creation	Count of leases created by a secret engine (excluding leases created internally for token expiration.)

02

DR Operations

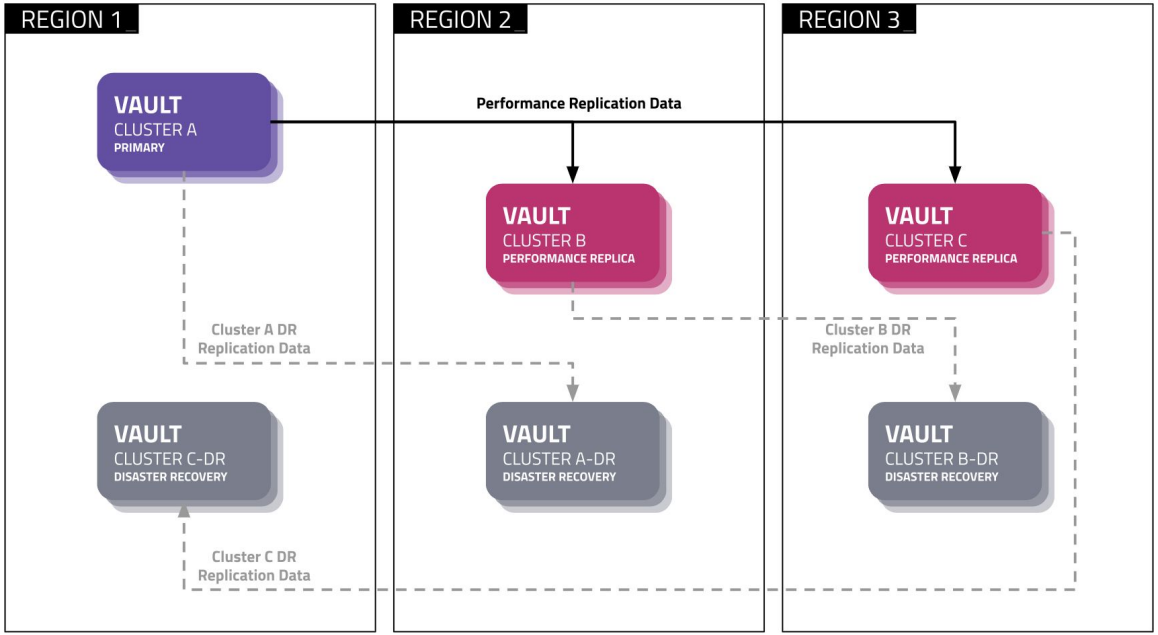
Vault Disaster Recovery Replication

Vault Enterprise offers two modes of replication to help organizations ensure Vault remains highly available. Most organizations will typically leverage combinations of both disaster recovery and performance replication to meet their SLA and meet resilience objectives.



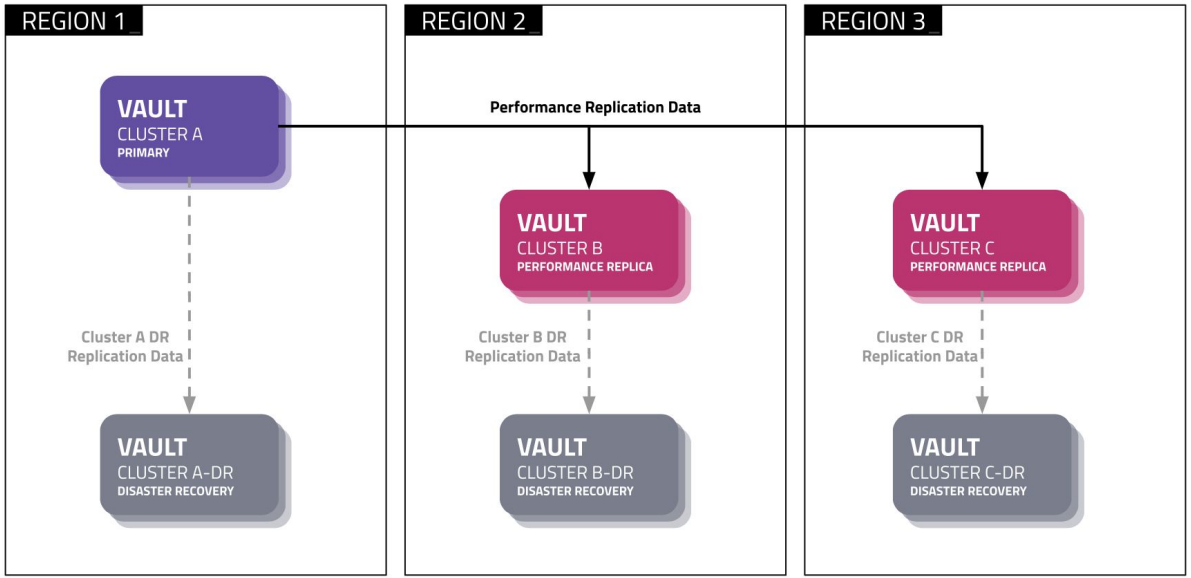
Resilience Against Region Failure

Leveraging Disaster Recovery and Performance Replication



Resilience Against Cluster Failure

Leveraging Disaster Recovery and Performance Replication



DR Operation Token Strategy

DR Operation Token

Requires a quorum of unseal/recovery keys to generate an operations token. This can introduce delays in promotion of a DR cluster as you try to bring all key holders together to generate an operations token.

Batch DR Operations Token

Available since Vault 1.4, allows for Vault Operator to prepare for a DR event by generating a batch DR operation token ahead of time. Batch tokens have a fixed TTL and will require the operator to manage the token lifecycle to ensure the token is valid prior to the DR event. If the token expires and a DR event occurs, you will need to follow the process to generate a DR operations token.

Batch DR Operations Token Policy

CODE EDITOR

```
path "sys/replication/dr/secondary/promote" {
  capabilities = [ "update" ]
}

# To update the primary to connect
path "sys/replication/dr/secondary/update-primary" {
    capabilities = [ "update" ]
}

# Only if using integrated storage (raft) as the storage
backend
# To read the current autopilot status
path "sys/storage/raft/autopilot/state" {
    capabilities = [ "update" , "read" ]
}
```

Generating Batch DR Operations Token

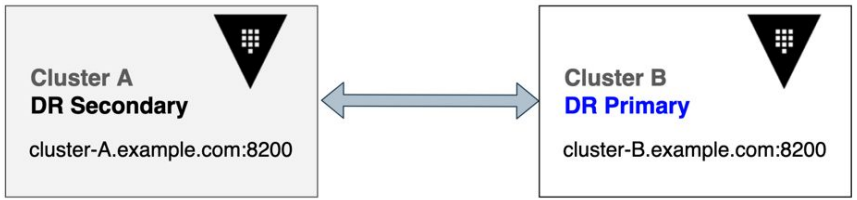
TERMINAL

```
> vault write auth/token/roles/failover-handler \
    allowed_policies=dr-secondary-promotion \
    orphan=true \
    renewable=false \
    token_type=batch

> vault token create -role=failover-handler -ttl=8h
```

DR Failback

After a DR event you may want failback to the original primary. To return to the original state, you will be performing the process in the reverse direction. The original primary will need to become a secondary and sync against the new primary. Once in-sync you can then promote the original primary. Lastly, you will need to change the mode of the DR cluster to become a secondary again.



Automated DR Failover

Vault does not support an automatic failover/promotion of a DR secondary cluster, and this is a deliberate choice due to the difficulty in accurately evaluating why a failover should or shouldn't happen. For example, imagine a DR secondary loses its connection to the primary. Is it because the primary is down, or is it because networking between the two has failed?

If the DR secondary promotes itself and clients start connecting to it, you now have two active clusters whose data sets will immediately start diverging. There's no way to understand simply from one perspective or the other which one of them is right.

Are the DR Clusters in Sync

When the clusters are fully in sync, you can expect to see:

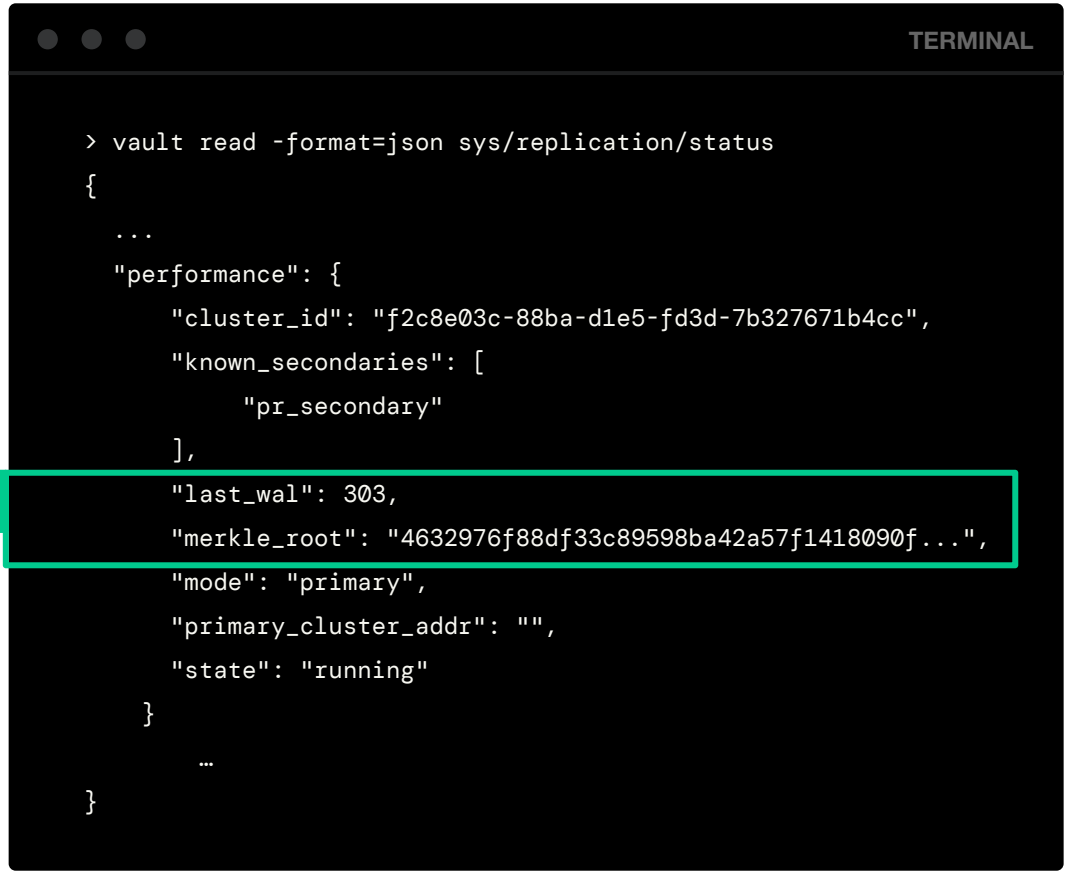
- **state** of secondary will be **stream-wals**
- **last_remote_wal** on the secondary should be very close to the **last_wal** on the primary
- **merkle_root** on the primary and secondary will match

Status Parameters on Secondaries

State	Description
stream-wals	Normal streaming (this is the value you want to see)
merkle-diff	The cluster is **determining** the sync status to see if a merkle sync is required
merkle-sync	The cluster is **syncing** - the secondary is too far behind the primary to use WALs to catch up (blocking operation)
idle	Indicates an **issue**. Needs investigation!

Verify Replication Status

Performance **Primary**



Verify Replication Status

Performance **Secondary**

TERMINAL

```
> vault read -format=json sys/replication/status
{
  ...
  "performance": {
      "cluster_id": "f2c8e03c-88ba-d1e5-fd3d-7b327671b4cc",
      "known_primary_cluster_addrs": [
          "https://perf-primary.example.com:8201"
      ],
      "last_remote_wal": 303,
      "merkle_root": "4632976f88df33c89598ba42a57f1418090f...",
      "mode": "secondary",
      "primary_cluster_addr":
"https://perf-primary.example.com:8201",
      "secondary_id": "pr_secondary",
      "state": "stream-wals"
    }
      …
}
```

WAL Metrics

Metric	Description
vault.wal_persistwals	Time taken to persist a WAL to storage
vault.wal_flushready	Time taken to flush a ready WAL to storage
wal.gc.total	Total number of WAL on disk
wal.gc.deleted	Number of WAL deleted during each garbage collection run

Replication Metrics

Metric	Description
logshipper.streamWALs.missing_guard	Number of missing guards: the Merkle tree index used to begin streaming WAL entries is not found/missing
logshipper.streamWALs.guard_found	Number of found guards
replication.fetchRemoteKeys	Time taken (in milliseconds) to perform a Merkle tree based delta generation between the clusters
replication.merkleDiff	Time taken (in milliseconds) to perform a Merkle tree based delta generation between the clusters
replication.merkleSync	Time taken (in milliseconds) to perform a Merkle tree based synchronization using the last delta generated between the clusters

Replication State Management

WAL Replays

WALs are replayed at server startup as well as during a reindex. At startup, the WAL replay blocks the incoming requests (no reads or writes). If replication is in a bad state or data has been removed from the storage backend without Vault's knowledge, reindex the Merkle tree via sys/replication/reindex endpoint

Merkle Tree

Vault uses a Merkle Tree to replicate data consistent across the Primary and Secondary Clusters.

- Use sys/replication endpoint to restore the replication state
- If replication is in an adverse state:

  ```
  vault write -f sys/replication/recover
  ```
- Manually reindex the local data storage:

  ```
  vault write -f sys/replication/reindex
  ```

03

Runbooks

Runbooks

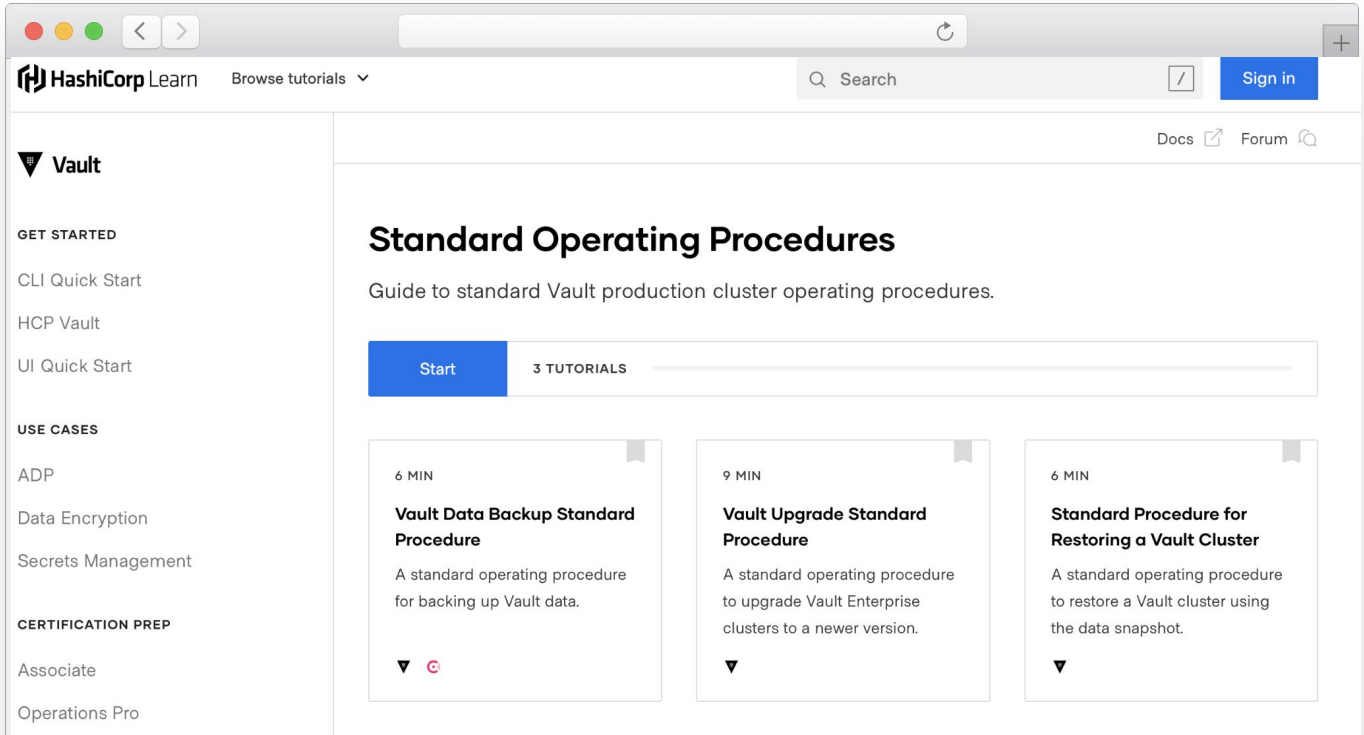
As you proceed towards production, runbooks should be created for the operations involved in managing the lifecyle of your Vault cluster. Some of the most common runbooks include

1. Backup/Restore
2. DR Cluster Promotion
3. Upgrades

Developing Runbook

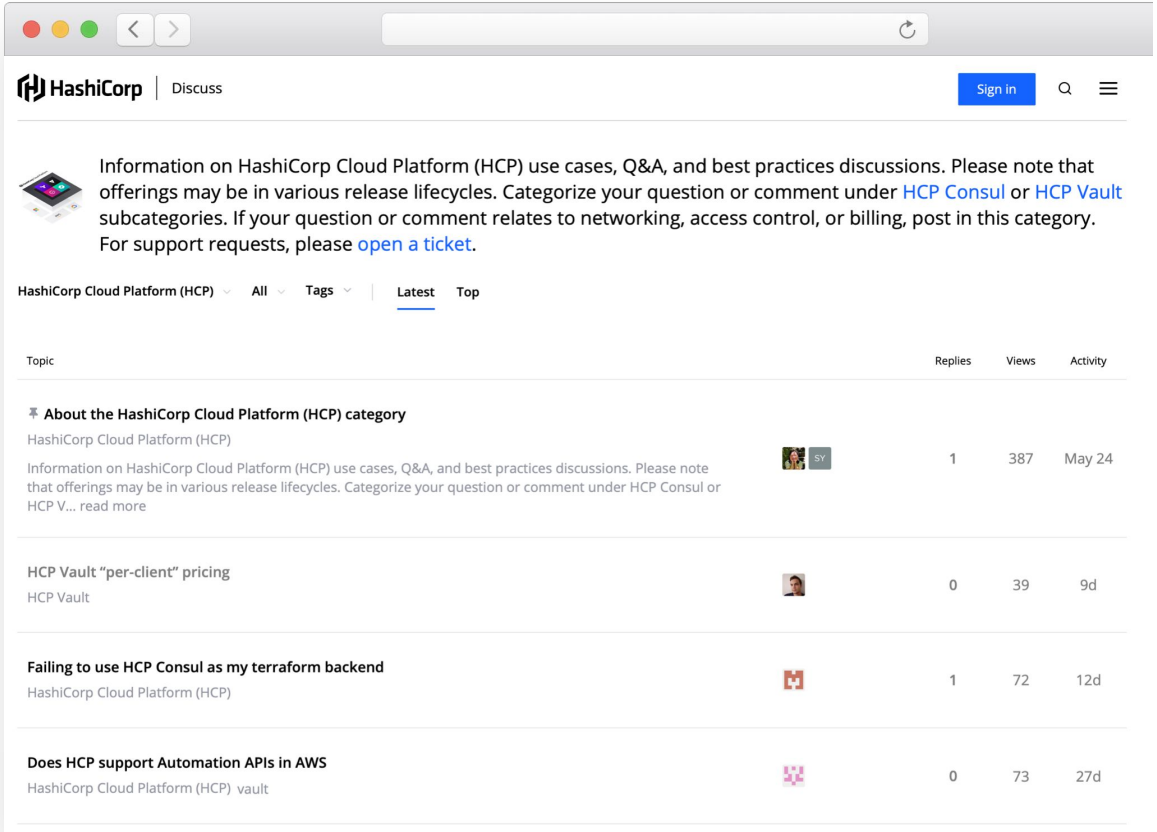
1. Identify scenarios
2. Identify RACI
3. Document tasks
4. Test in non-production
5. Implement Runbook in production
6. Lifecycle management

Resources


HashiCorp Learn
Browse tutorials
Search
/
Sign in
Vault
Docs
Forum
GET STARTED
CLI Quick Start
HCP Vault
UI Quick Start
USE CASES
ADP
Data Encryption
Secrets Management
CERTIFICATION PREP
Associate
Operations Pro
Standard Operating Procedures
Guide to standard Vault production cluster operating procedures.
Start
3 TUTORIALS
6 MIN
Vault Data Backup Standard Procedure
A standard operating procedure for backing up Vault data.
9 MIN
Vault Upgrade Standard Procedure
A standard operating procedure to upgrade Vault Enterprise clusters to a newer version.
6 MIN
Standard Procedure for Restoring a Vault Cluster
A standard operating procedure to restore a Vault cluster using the data snapshot.

04

Next Steps



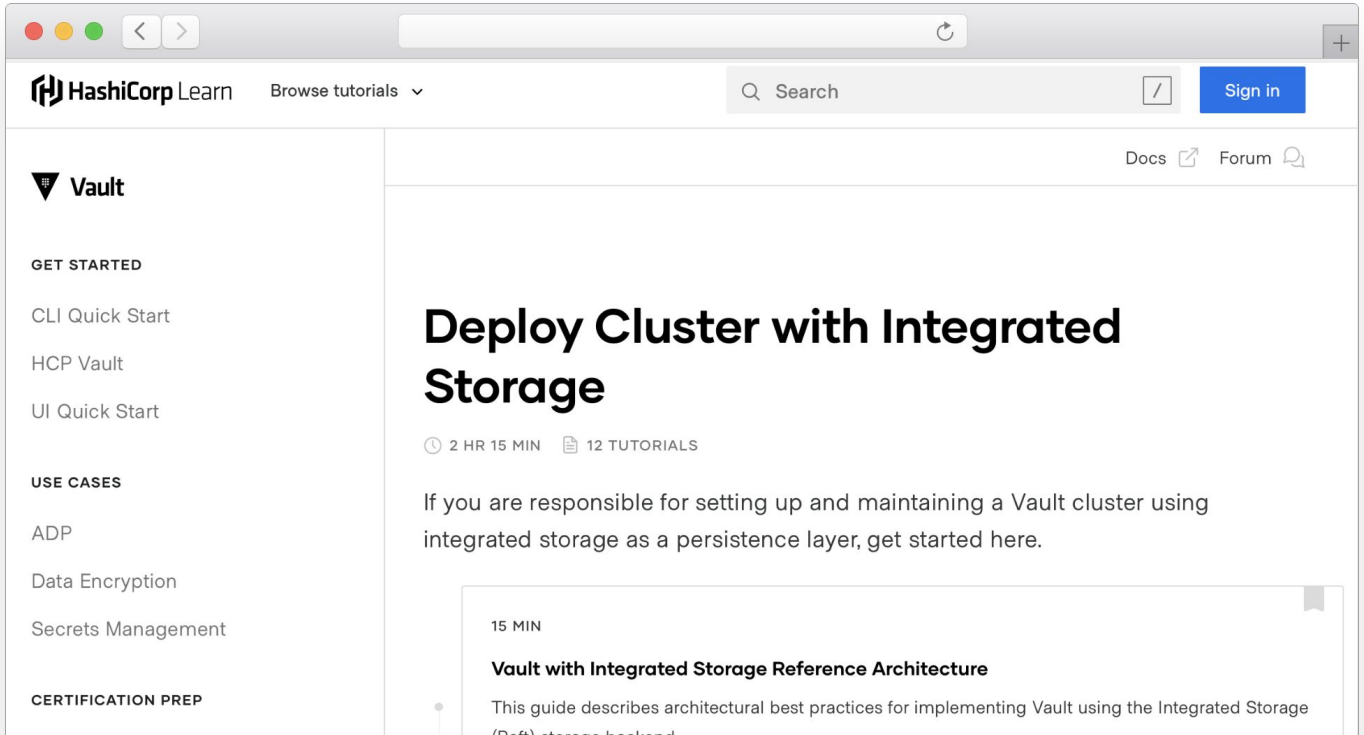
Discuss

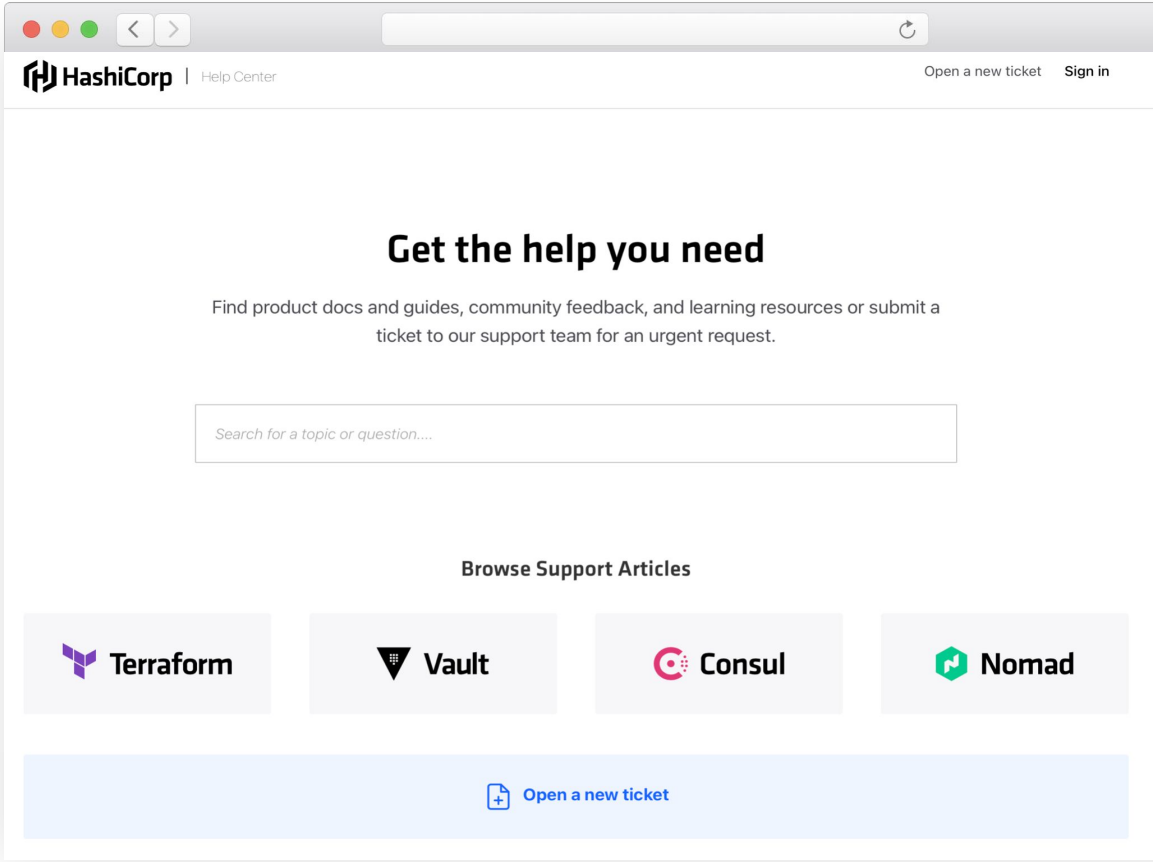
Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers.

discuss.hashicorp.com

Learn

Step-by-step guides to accelerate deployment of Vault





Support

https://support.hashicorp.com

05

Q & A

Thank You

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