HashiCorp
Vault

Vault Self-Managed Webinar - Consuming Vault

January 2022

Vault Enterprise Path to Production

Weeks 1-3	Weeks 4-5	Weeks 6-7	Weeks 8-9	Weeks 10-11
Vault Deployed	**Identity & Access Management**	**Operational Considerations**	**Secrets Management**	**Ready for Production!**
• Kick Off Webinar • Architectural Deep Dive Workshop • Terraform starter code modules • Office Hours • Schedule Success Planning Session	• Namespaces, ACL Policies, Auth Methods Workshop • Office Hours • 1:1 Success Planning • Lunch & Learn: *Migrate from Vault OSS to Vault Enterprise*	• Telemetry & Monitoring and DR Operations Workshop • HCDiag & Vault Metrics Lunch & Learn • Office Hours • 1:1 Success Planning	• How to consume Vault into applications Workshop • Vault Governance • Office Hours • 1:1 Success Planning	• Success Plan Reviews • Exit Ramp & Operational Readiness Workshop

Agenda

- Secure Introduction

- Consuming Secrets

- Third Party Integrations

- Next Steps

- Q & A

01

Secure Introduction

Secret Originator and Consumer

If you can securely get the first secret from originator to a consumer, all subsequent secrets transmitted between them can be authenticated with the trust established by the successful distribution of the first secret.

Tokens are the core method for authentication within Vault. Every secret consumer (client) must acquire a valid token

Methods for Secure Introduction

Platform Integration

Vault establishes a trust with your trusted platforms (AWS, Azure, GCP) to use the identifier of resources (virtual instances, containers, etc) to authenticate and provide authorization to a Vault token.
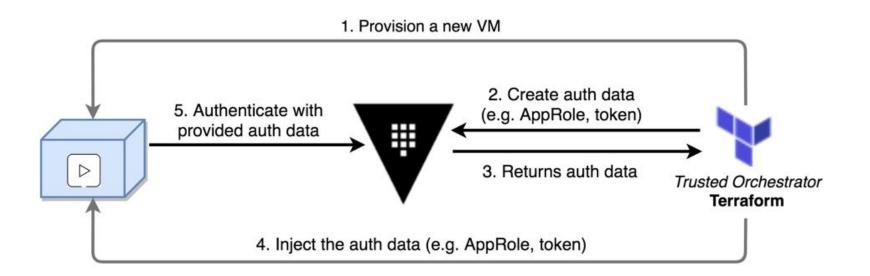
Trusted Orchestrator

Your existing trusted orchestrator (Terraform, Kubernetes, Chef) has already been authenticated to Vault with privileged permissions. During deployment of applications, orchestrator injects necessary credentials to authenticate to Vault and retrieve a Vault token.
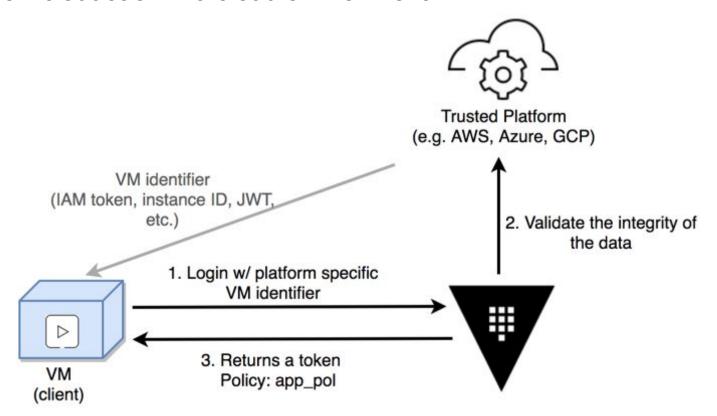
Trusted Orchestrator
Secure introduction in a VM environment

Platform Integration
Secure introduction in a cloud environment

VM identifier
(IAM token, instance ID, JWT, etc.)

Trusted Platform
(e.g. AWS, Azure, GCP)

2. Validate the integrity of the data

1. Login w/ platform specific VM identifier

3. Returns a token
Policy: app_pol

VM
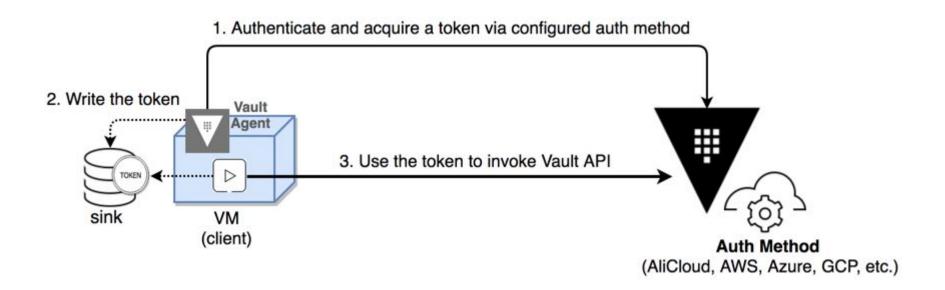(client)

Automating Introduction

Vault Agent is a client daemon which automates the client login workflow and will manage the lifecycle for the Vault token. Compatible with both platform integration and trusted orchestrator secure introduction methods.

Vault Agent is included as part of the Vault binary and can be run by starting the binary in agent mode `vault agent -config=<config-file>`. Once authentication has completed a Vault token will be written to file sink.

Automate Introduction
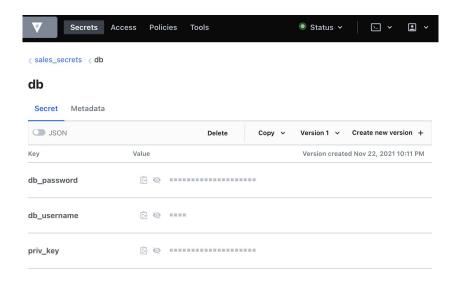**Vault Agent**

Consuming Secrets

Patterns to Consume Secrets

- UI

- CLI

- HTTP API

- Templating

- Environment Variables

- Client Libraries

Web UI

Vault UI enables users to populate and consume secrets from Vault without the need to learn about Vault CLI commands or API. This approach works well when a user is consuming secrets however it can be limiting when needing to consume a large number of secrets or if secrets are being consumed as part of an application configuration.

CLI

Similar to UI, Vault CLI is most likely being leveraged by users to consume secrets manually
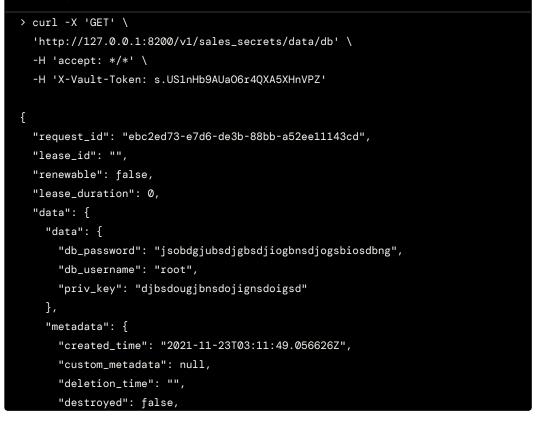
```
> vault kv get sales_secrets/db
======= Metadata =======
Key                 Value
---                 -----
created_time        2021-11-23T03:11:49.056626Z
custom_metadata     <nil>
deletion_time       n/a
destroyed           false
version             1


======= Data =======
Key                 Value
---                 -----
db_password         jsobdgjubsdjgbsdjiogbnsdjogsbiosdbng
db_username         root
priv_key            djbsdougjbnsdojignsdoigsd
```
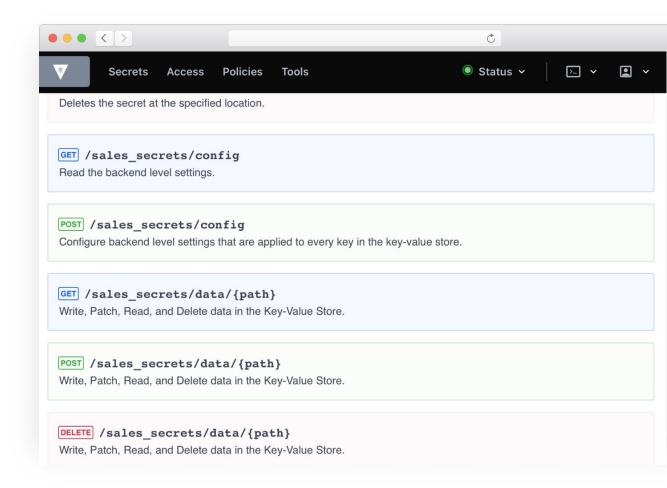
HTTP API

The Vault HTTP API gives you full access to Vault via HTTP. Every aspect of Vault can be controlled via this API.

```
> curl -X 'GET' \
  'http://127.0.0.1:8200/v1/sales_secrets/data/db' \
  -H 'accept: */*' \
  -H 'X-Vault-Token: s.US1nHb9AUaO6r4QXA5XHnVPZ'

{
  "request_id": "ebc2ed73-e7d6-de3b-88bb-a52ee11143cd",
  "lease_id": "",
  "renewable": false,
  "lease_duration": 0,
  "data": {
    "data": {
      "db_password": "jsobdgjubsdjgbsdjiogbnsdjogsbiosdbng",
      "db_username": "root",
      "priv_key": "djbsdougjbnsdojignsdoigsd"
    },
    "metadata": {
      "created_time": "2021-11-23T03:11:49.056626Z",
      "custom_metadata": null,
      "deletion_time": "",
      "destroyed": false,
```

HTTP API Explorer

<VAULT_ADDR>/ui/vault/api-explorer

Secrets Access Policies Tools

⦿ Status ⌄

Deletes the secret at the specified location.

`GET` `/sales_secrets/config`
Read the backend level settings.

`POST` `/sales_secrets/config`
Configure backend level settings that are applied to every key in the key-value store.

`GET` `/sales_secrets/data/{path}`
Write, Patch, Read, and Delete data in the Key-Value Store.

`POST` `/sales_secrets/data/{path}`
Write, Patch, Read, and Delete data in the Key-Value Store.

`DELETE` `/sales_secrets/data/{path}`
Write, Patch, Read, and Delete data in the Key-Value Store.
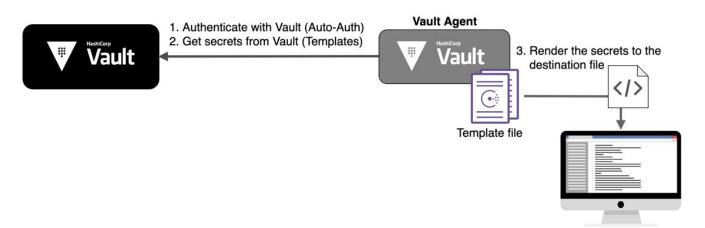
Vault Agent Templating

Vault Agent can fully automate the last mile and securely authenticate and retrieve secrets from Vault. When configured with auto-auth, templating can be configured to retrieve a secret for which the resource has authorization to and template that file to a sink. Template files are written using the Consul Template markup language.
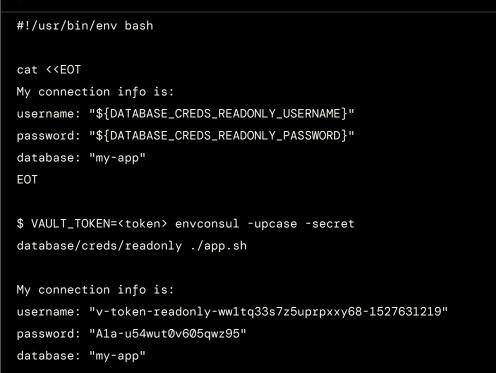
Vault Agent Templating

Example Template

```
> cat customer.tmpl

{{ with secret "secret/data/customers/acme" }}
Organization: {{ .Data.data.organization }}
ID: {{ .Data.data.customer_id }}
Contact: {{ .Data.data.contact_email }}
{{ end }}

> cat customer.txt

Organization: ACME Inc.
ID: ABXX2398YZPIE7391
Contact: james@acme.com
```

envconsul

A subprocess which dynamically populates environment variables from secrets read from Vault. Your applications then read those environment variables.

```bash
#!/usr/bin/env bash

cat <<EOT
My connection info is:
username: "${DATABASE_CREDS_READONLY_USERNAME}"
password: "${DATABASE_CREDS_READONLY_PASSWORD}"
database: "my-app"
EOT

$ VAULT_TOKEN=<token> envconsul -upcase -secret
database/creds/readonly ./app.sh

My connection info is:
username: "v-token-readonly-ww1tq33s7z5uprpxxy68-1527631219"
password: "A1a-u54wut0v605qwz95"
database: "my-app"
```

Go Client Library

Reference Documentation

```go
// get secret
secret, err := client.Logical().Read("kv-v2/data/creds")
if err != nil {
    return "", fmt.Errorf("unable to read secret:%w", err)
}

data, ok := secret.Data["data"].(map[string]interface{})
if !ok {
    return "", fmt.Errorf("data type assertion failed:%T
%#v", secret.Data["data"], secret.Data["data"])
}

// data map can contain more than one key-value pair,
// in this case we're just grabbing one of them
key := "password"
value, ok := data[key].(string)
if !ok {
    return "", fmt.Errorf("value type assertion failed:%T
%#v", data[key], data[key])
}
```

Third Party Integrations

Ecosystem

A broad ecosystem of frameworks and tooling have been created to help support integrations between third party tools and services. These frameworks and tooling can ease the burden on your end users to integrate and consume secrets from Vault.

Considerations

Support

HashiCorp is unable to provide technical support for third party frameworks and tooling. We can support you from the Vault side however any issues with the framework or tooling will need to be raised with the creator of those frameworks or tooling.

Enterprise Capabilities

We have established partnerships with a number of partners who have created tooling and framework that support enterprise capabilities (ex. namespaces). If the tooling or framework that you are attempting to use does not support enterprise capabilities, please have them reach out to us if they are interested in supporting enterprise capabilities.

Java Applications

Spring Cloud Vault client libraries

```ruby
class Person < ActiveRecord::Base
  include Vault::EncryptedModel
  vault_attribute :ssn
end

class AddEncryptedSSNToPerson < ActiveRecord::Migration
  add_column :persons, :ssn_encrypted, :string
end


person = Person.new
person.ssn = "123-45-6789"
person.save #=> true
person.ssn_encrypted #=> "vault:v0:EE3EV8P5hyo9h..."
```

Spring Cloud Vault

Java Application Demo

Vault C# Client

Integrate with your .Net Applications

[Using HashiCorp Vault C# Client with .NET Core](#)

```csharp
public VaultConfigurationProvider(VaultOptions config)
{
  _config = config;
  var vaultClientSettings = new VaultClientSettings(
      _config.Address,
      new AppRoleAuthMethodInfo(_config.Role,
                                _config.Secret)
  );
  _client = new VaultClient(vaultClientSettings);
}

public class VaultOptions
{
  public string Address { get; set; }
  public string Role { get; set; }
  public string Secret { get; set; }
  public string MountPath { get; set; }
  public string SecretType { get; set; }
}
```
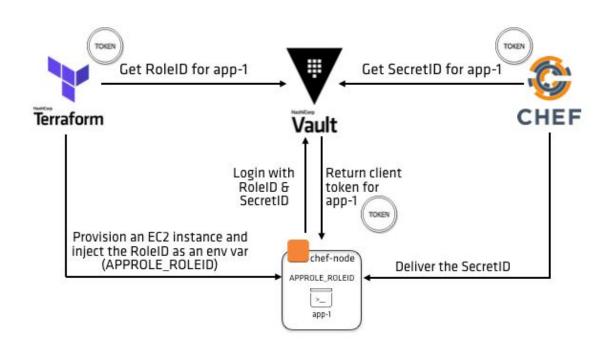
Pipeline Integration

Github Actions

Vault Secrets · Actions · GitHub Marketplace · GitHub

```yaml
jobs:
    build:
        # ...
        steps:
            # ...
            - name: Import Secrets
              uses: hashicorp/vault-action@v2.3.1
              with:
                url: https://vault.mycompany.com:8200
                token: ${{ secrets.VaultToken }}
                caCertificate: ${{ secrets.VAULTCA }}
                secrets: |
                    secret/data/ci/aws accessKey |
AWS_ACCESS_KEY_ID ;
                    secret/data/ci/aws secretKey |
AWS_SECRET_ACCESS_KEY ;
                    secret/data/ci npm_token
```
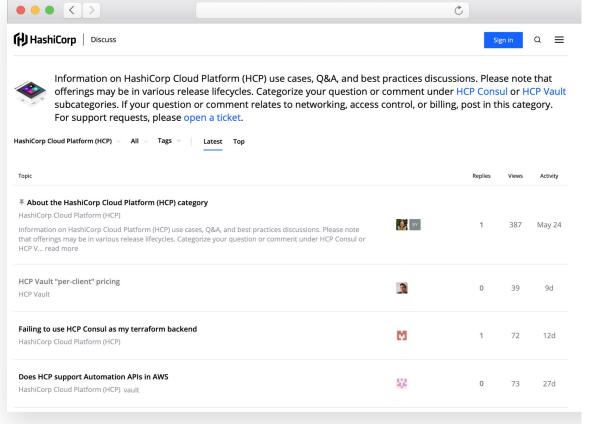
Pipeline Integration

Chef

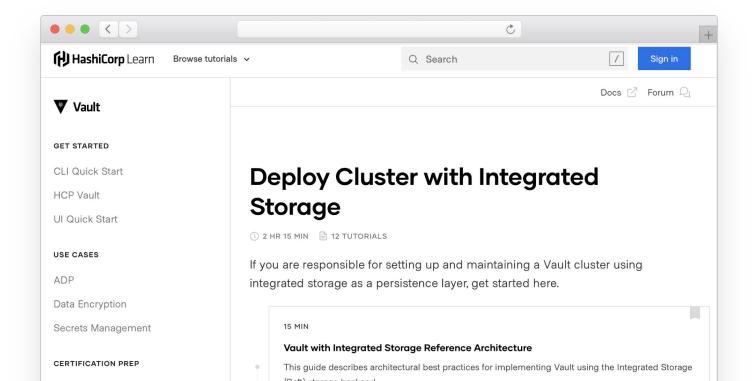[**AppRole With Terraform & Chef | Vault**](#)

Next Steps

Discuss

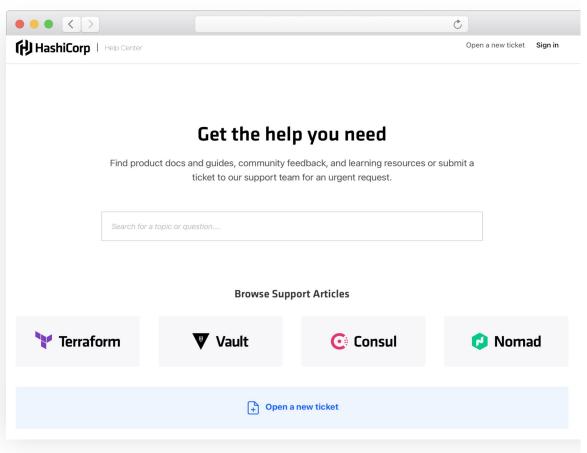Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers.

discuss.hashicorp.com

Learn

Step-by-step guides to accelerate deployment of Vault

Support

https://support.hashicorp.com

Q & A

Thank You

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