

# Managing Application Secrets

Christopher McNabb  
Enterprise Systems

Hi. I'm Christopher McNabb and today I'm going to be talking about Secrets Management - What it is and how we are using it to keep our applications secure.

# Secrets Management

Secrets management refers to the tools and methods for managing digital authentication credentials, including passwords, keys, APIs, and tokens for use in applications, services, privileged accounts and other sensitive parts of the IT ecosystem.

Here is a definition of Secrets Management. Boiled down it means using technology to ensure that applications have ready access to the secrets they need to function while keeping those secrets secure.

# Secrets Management

## Why Hashicorp Vault?

- Secrets are encrypted on disk and in transit
- Secrets can be dynamically generated on-demand
- Access to secrets is tightly controlled
- Integration with other services such as AWS and gitlab

In enterprise systems we are using Hashicorp Vault to manage our secrets because it is encrypted from end to end, secrets can be dynamically created on demand, access to secrets is tightly controlled and it is well integrated with other services that we use.



# Types of Secrets Stores

- Key - Value
- Database
- AWS IAM

The types of secret stores we manage are key-value pairs, database credentials, and AWS IAM credentials.

# Key/Value Pairs

- Usually contain keys for a username and a password
- Can include other keys such as a url, notes, or other information

Key value pairs usually contain usernames and passwords. They can include other information as well.

# Key/Value Pairs

```
{
  "request_id": "9d40b709-f93a-d761-725d-2bf973f09123",
  "lease_id": "",
  "lease_duration": 0,
  "renewable": false,
  "data": {
    "data": {
      "foo": {
        "bar": "baz",
        "foo": "bar"
      },
      "password": "AustinPowersIsNotMyFriend",
      "url": "https://spectre.org",
      "username": "Dr.Evil"
    },
    "metadata": {
      "created_time": "2022-10-16T15:43:25.655275603Z",
      "custom_metadata": null,
      "deletion_time": "",
      "destroyed": false,
      "version": 2
    }
  },
  "warnings": null
}
```

This is what it looks like when you read a key-value secret from Vault. This is a KV Version 2 secret which means it has version control and the ability to roll back unintended changes. There are a few top level fields. The important one is “data” which has the subfields “data” (yes, again) and “metadata”. The data subfield contains the actual values you want to read. If you take a look at the “foo” field in this example you will notice that it has subfields of its own. The “metadata” field contains information about the secret itself - when it was created, its version, and so on.



# Database Secrets

Static Roles: One to one mapping of a database user and a Vault database role. Vault rotates the database user password on a configurable period of time.

Vault database secret engines allow vault to manage database credentials. In Vault there are two database secret types - static and dynamic. Static roles have a one-to-one mapping to a database user. Vault rotates the database user's password on a predetermined regular basis.

# Database Static Role

```
{
  "request_id": "cd68e93b-b37b-920b-fae0-d1a1711545c7",
  "lease_id": "",
  "lease_duration": 0,
  "renewable": false,
  "data": {
    "last_vault_rotation": "2022-10-16T12:03:46.16641251-04:00",
    "password": "T3-aAjFr27iBDqg5MZh0",
    "rotation_period": 60,
    "ttl": 9,
    "username": "cmcnabb"
  },
  "warnings": null
}
```

This is what it looks like when you read a static database secret. Again, there are several top level fields including the data field. In the data field we have the time of the last password rotation, the password itself, how often in seconds the password is rotated, how long before the password is rotated again, and the username. Using one of these secrets the only thing your application needs to know before run-time is how to connect to the database. It can read the username and password from Vault when it starts.



# Database Credentials

Dynamic Roles: Vault creates database users on-demand with appropriate database privileges. The created database user's credentials are returned to the requestor. Automatically dropped when the lease expires or is revoked

Dynamic roles are like static roles except that Vault creates the database user on demand. After a predetermined period of time the user is automatically removed from the database.

# Database Dynamic Role

```
{
  "request_id": "e7917359-1b7c-8edf-773b-7c79c53f2c1e",
  "lease_id": "database/creds/sw-write/1KG0wt1hJbT7ARwupmaixCct",
  "lease_duration": 60,
  "renewable": true,
  "data": {
    "password": "clafj2z8-nw8uQpusbA1",
    "username": "v-root-sw-write-foz3agJ4weBgoun1hBw8-1665936110"
  },
  "warnings": null
}
```

In a dynamic database role Vault logs into the database with a user that has just enough privileges to create another user. It creates a new user with the grants that it needs to do its job and returns those database credentials to the requester. When the `lease_duration` expires or the lease is revoked Vault will connect to the database and issue commands to drop the user in a safe way. The application can make use of the `lease_id` value to revoke its own access once its work is done.

# AWS Credentials

- Vault generates AWS access credentials dynamically
  - iam user
  - Assumed role
- Returns an access key, secret key, and session token
- Time-based and automatically revoked when the Vault lease expires.

We use the Vault AWS secrets engine to manage IAM Users and assumed roles. These are created on demand and have a predetermined time to live. Once the time to live has expired they are deleted.



# AWS Credentials

```
{
  "request_id": "576facae-e0b5-0846-7e36-7092109d5e54",
  "lease_id": "aws.iam.dit.es/creds/dbaa.database/l9lgH7b0BLC0jKGA0WBSjtLi",
  "lease_duration": 32400,
  "renewable": true,
  "data": {
    "access_key": "AKIAU2554T4HCENYw3DB",
    "secret_key": "HjX1na34p0V9WEcPu7FkMGqP3QcucP/7c+jeNbJL",
    "security_token": null
  },
  "warnings": null
}
```

This is an example of the data returned from a Vault dynamic IAM role. Again, we get the lease\_id so we can revoke it once we are done with our work. In the data field we have the access\_key, secret\_key, and a security token (which is null in this case).

# Types of Secrets Stores

- Key - Value
- Database
- AWS IAM

So, we have those three types of secrets. How do we control who can actually see them? We use policies.

# Policies

Policies provide a declarative way to grant or forbid access to paths and operations in Vault.

- Path based
- Default deny
- A token may have more than one attached policy
- In the case of conflicting policies the most restrictive applies.

Policies are attached to Vault tokens to grant access to Vault resources. A token can have more than one attached policy so that privileges can be grouped by what needs to be accessed. If there is a conflict in the attached policies - such as one granting read and update on a particular path but the other granting only read on that path - then the most restrictive policy applies. In this case that would be read only.



# Policies

A policy that allows reading the sw-read database role, to list the 'kv' secrets mount, and to both list and read secrets within the 'kv' secrets mount.

```
path "database/creds/sw-read" {
  capabilities = [ "read" ]
}
path "kv" {
  capabilities = [ "list" ]
}
path "kv/*" {
  capabilities = [ "list", "read" ]
}
```

This is an example of a policy that allows reading the sw-read dynamic database role, to list what's in the kv path and to list and read everything under the kv path.

# Authentication Methods

Perform authentication and assign identity and a set of policies to a user. Always return a token.

Authentication Methods authenticate users and generate a token with the policies the user needs attached to it. We are currently supporting authentication using Application Roles, CI\_JOB\_JWTs from code.vt.edu, Common Platform service accounts, and AWS EC2 Instances and Roles.

# Authentication Methods

## Tokens

- Every Vault action requires a token
- Can be scoped to one or more network ranges
- Have a predetermined time to live but can be renewed up to a maximum time to live.
- A token has one or more attached policies.

Authentication methods return a token on success. Every Vault action except login requires a token. Tokens can be scoped to one or more network ranges and have a predetermined time to live but can be renewed up to a predetermined maximum. Tokens can also be created manually outside of an authentication method. There aren't a lot of use cases for manually created tokens but one of those would be if your CI/CD engine doesn't have any other way to login to Vault. In our Jenkins implementation, for example, a manually created token is used to create Application Role secret IDs to be passed to applications during deployment. Tokens always have one or more policies attached to them.



# Application Roles

An application role represents a set of Vault policies and login constraints that must be met to receive a token with those policies.

```
resource "vault_approle_auth_backend_role" "read-data" {  
  backend = vault_auth_backend.approle.path  
  role_name = "read-data"  
  secret_id_ttl = 60  
  secret_id_num_uses = 1  
  token_ttl = 300  
  token_policies = [  
    "default",  
    vault_policy.sw-read.name  
  ]  
  token_num_uses = 5  
}
```

An application role represents a set of Vault policies and login constraints that must be met to receive a token. These constraints can include the IP address range that can request a secret-id, how many times the secret-id can be used, the IP address range that can use the secret-id to login to vault, how many times the resultant token can be used, where it can be used from, how long it can be used for.

# Application Roles

```
cmcnabb@n2d037972 dcss-space-weather % vault read auth/approle/role/read-data/role-id
Key      Value
----      -
role_id  63011c29-57b8-0f5c-6baf-091066966b16
```

```
cmcnabb@n2d037972 dcss-space-weather % vault write -f auth/approle/role/read-data/secret-id
Key      Value
----      -
secret_id      4d8c45be-9adf-e54f-55cc-4dfc1e1aefb4
secret_id_accessor  55fb12f6-0832-2bd7-99d8-647877b4b2be
secret_id_ttl      1m
```

The top example shows using the Vault command line to read the `role_id` for an `approle`. The `role_id` is a static value that is similar to a username. It can be either baked into an application or provided to an application at runtime. The `secret_id` is a dynamic value that is similar to a password. It is created upon request and has a time to live and a limited number of times it can be used. It's creation and usage can be restricted to specific IP address ranges. These ranges do not have to be the same so the `secret-id` can be created by one system and passed to another system for usage. The bottom example shows how to generate a `secret-id`. This is done with a `write` statement. The `-f` option forces the write since there is no accompanying data to be written. As you can see this `secret-id` must be used within one minute.

# Application Roles

```
cmcnabb@n2d037972 dcss-space-weather % vault write -format=json auth/approle/login role_id=63011c29-57b8-0f5c-6baf-091066966b16 secret_id=4d8c45be-9adf-e54f-55cc-4dfc1e1aefb4
{
  "request_id": "32d2cca4-b67d-6c51-fae8-7dbe3cd12747",
  "lease_id": "",
  "lease_duration": 0,
  "renewable": false,
  "data": null,
  "warnings": null,
  "auth": {
    "client_token": "hvs.CAESIP5G4FyJBlWi06QoPGd-I2dGRrTszk14_tqbU3V1-G8-Gh4KHGh2cy5RR0FqTWZscGRaZnVwMFVLRjBTYmduUuc",
    "accessor": "V0RIjFEbnmHRfLZJH8spXsUW",
    "policies": [
      "default",
      "sw-read"
    ],
    "token_policies": [
      "default",
      "sw-read"
    ],
    "identity_policies": null,
    "metadata": {
      "role_name": "read-data"
    },
    "orphan": true,
    "entity_id": "32db614c-9ac8-8d35-6543-21e4d1f646ca",
    "lease_duration": 300,
    "renewable": true,
    "mfa_requirement": null
  }
}
```

In this example we use the vault command line tool to generate a token from a role-id and secret-id. The token itself is returned in the auth.client\_token field. This token would be used by applications to read their secrets from Vault. Other information, such as the policies attached to the token and how long it is valid for are also returned. The auth.accessor value can be useful for troubleshooting issues since it can be used by Vault administrators to look up information about the token without having to actually have the token.



# JWT Roles

JWT roles are used to authenticate from gitlab runners  
in [code.vt.edu](https://code.vt.edu)

- Uses the `$CI_JOB_JWT` gitlab variable
- Can be bound to a namespace, a project, a branch, a tag, a user, or any of the values included in the JWT

JWT roles allow for authentication using a JWT. The gitlab implementation on [code.vt.edu](https://code.vt.edu) provides a `CI_JOB_JWT` value to every pipeline job it runs. The Vault JWT role can be bound to one or more gitlab users, namespaces, projects, branches or tags.

# JWT Roles

```
resource "vault_jwt_auth_backend_role" "sw-write" {
  backend=vault_jwt_auth_backend.jwt.path
  role_type = "jwt"
  role_name = "sw-write"
  You, 27 minutes ago | 1 author (You)
  bound_claims = {
    project_path = "es/database/dcss-space-weather"
  }
  token_policies = [
    vault_policy.sw-write.name
  ]
  token_explicit_max_ttl = 3600
  token_max_ttl = 60
  token_type = "default"
  user_claim = "user_login"
}
```

```
resource "vault_policy" "sw-write" {
  name = "sw-write"
  policy = <<EOT
path "database/creds/sw-write" {
  capabilities = [ "read" ]
}
EOT
}
```

In this example we use terraform to define a role that is scoped to just the es/database/dcss-space-weather project. Tokens generated by this role live for one minute but can be renewed for up to an hour. Since the maximum number of uses isn't specified they can be used any number of times until they expire. The policy attached to the token allows the application to read just database credentials from the sw-write Vault database role

# Kubernetes Roles

The Kubernetes auth method is used to authenticate to Vault from the Common Platform.

- Vault Kubernetes auth roles are defined using the platform-access terraform module in the es/Vault/config repository
- The vault-secret-syncer helm chart is used to configure the platform tenant resources that read required secrets and make them available to the application

Vault Kubernetes roles allow authentication from Kubernetes service accounts. For the common platform Vault Kubernetes auth roles are defined using the a platform-access terraform module In the common platform tenant the vault-secret-syncer helm chart is used to configure the resources that read required secrets and make them available to the application



# Kubernetes Roles

```
module "nis-vttrack-dvlp-dtr" {  
  source = "../modules/platform-access"  
  app_name = "nis-vttrack-dvlp-dtr"  
  tenant_identifier = "nis-vttrack-dvlp"  
  service_account_name = "vault-secret-syncer"  
  secret_paths = ["es.dbaa/data/dtr"]  
}
```

```
# Add the DTR secret to our tenant so we can pull images  
# This is referenced by our application deployments  
apiVersion: helm.toolkit.fluxcd.io/v2beta1  
kind: HelmRelease  
metadata:  
  name: vault-secret-syncer  
spec:  
  interval: 1h  
  targetNamespace: nis-vttrack-dvlp  
  serviceAccountName: flux  
  chart:  
    spec:  
      chart: vault-secret-syncer  
      sourceRef:  
        kind: HelmRepository  
        name: vault-secret-syncer  
        namespace: platform-helm-repos  
  values:  
    vault:  
      role: it-platform-nis-vttrack-dvlp-dtr  
      secret:  
        path: es.dbaa/data/dtr  
        usernameKey: username  
        passwordKey: password
```

On the left we see a terraform definition for a Vault Kubernetes role that allows the nis-vttrack-dvlp Tenant to read the secret it need to pull images from dtr. On the right is the helm chart that is used to deploy vault-secret-syncer into the tenant. It specifies the Vault role to be used, the path to the secret in Vault, and which keys it needs from the secret.

# AWS Auth Method

The AWS auth method provides for authenticating from EC2 Instances and IAM Roles

For our AWS based resources we use the AWS Auth method to connect from EC2 Instances and IAM Roles.

# AWS EC2 Instances

```
- name: Get Metadata Token
  uri:
  url: http://169.254.169.254/latest/api/token
  method: PUT
  headers:
    X-aws-ec2-metadata-token-ttl-seconds: 21600
  return_content: yes
  register: aws_token

- name: Get PKCS7 token
  uri:
  url: http://169.254.169.254/latest/dynamic/instance-identity/pkcs7
  headers:
    X-aws-ec2-metadata-token: "{{aws_token.content}}"
  return_content: yes
  register: response

- name: Remove newlines
  set_fact:
    pkcs7: "{{response.content | regex_replace('\n', '')}}"

- name: Do we already have a login?
  stat:
    path: /root/.vault-login_{{vault_role}}
  register: login_exists

- name: First Login to Vault
  shell:
    cmd: vault write -format=json -address "{{vault_url}}" auth/aws/login pkcs7="{{pkcs7}}" role="{{vault_role}}" header_value="vault.es.cloud.vt.edu" >> /root/.vault-login_{{vault_role}}
  when: not login_exists.stat.exists
```

Logging in from an EC2 instance is a little involved. In this Ansible example we first login to the instance's metadata endpoint and then retrieve the instance's PKCS7 token. With the instances pkcs7 token in hand we issue a login to Vault and save the returned token data to our home directory. On the first login part of that data is a nonce value that is required for all subsequent logins from that instance.



# AWS EC2 Instances

```
{
  "request_id": "0e99dd43-df03-4d8a-831d-f3eaafab2ba9",
  "lease_id": "",
  "lease_duration": 0,
  "renewable": false,
  "data": null,
  "warnings": [
    "TTL of \"768h\" exceeded the effective max_ttl of \"500h\"; TTL value is capped accordingly"
  ],
  "auth": {
    "client_token": "s,ahGkLJsr6qixVo2gRNpHj9rV",
    "accessor": "NORa7ArdSicT58WUU8mCxFTy",
    "policies": [
      "dit_es_bootstrap"
    ],
    "token_policies": [
      "dit_es_bootstrap"
    ],
    "identity_policies": null,
    "metadata": {
      "auth_type": "ec2",
      "instance_id": "i-0ae6c70ad586f391c",
      "nonce": "7164d76f-2be3-bf3c-1035-4d25f5395bf4",
      "role": "dit-es-bootstrap",
      "role_tag_max_ttl": "0s"
    },
    "orphan": true,
    "entity_id": "866667af-1c27-899f-fdf3-93e4a1940ba0",
    "lease_duration": 1800000,
    "renewable": true
  }
}
```

This is an example of the data returned from an EC2 instance login. The token is in the `auth.client_token` field. The `auth.metadata.nonce` value must be saved for future logins from the instance.

# Example

Authenticating to Vault with a code.vt.edu CI\_JOB\_JWT and getting credentials for a dynamic database user. Then using those credentials to login to a database and insert CSV based data from the internet into a table.

I have an example of using Vault to provide secrets at runtime to an application. This example will download a CSV file of solar flare data from the internet, login to vault, get dynamic database credentials, and use them to upload the solar flare data into a postgres table.

# How to Get Vault

Access to Vault can be requested from the Service Now catalog. You can find it in the Service Offerings tab under Security->Secrets & Password Management

Vault is available as a requestable service in the service now catalog.





# How to Get Vault

## Secrets and Password Management ☆

View a comparison of password managers and secret management tools

The screenshot shows a web interface for comparing password managers. It features a sidebar with navigation icons, a main content area with a comparison table, and a 'Service Offerings' section. The 'Service Offerings' section highlights HashiCorp Vault as a solution for securely managing secrets.

Secrets	Information	Service Offerings
 LastPass is for end users to store and generate work-related and personal usernames and passwords to websites for everyday use via browser.		
 HashiCorp Vault provides programmatic access for both humans and machines to manage credentials including passwords, keys, APIs, and tokens for use in applications, services, privileged accounts and more.		

### Service Offerings

- Secrets Management (Vault)**  
Securely manage secrets

[View Details](#)