# hvac

***Release 0.10.13***

**Ian Unruh, Jeffrey Hogan**

**May 20, 2021**

# CONTENTS

CHAPTER

ONE

# HVAC


HashiCorp Vault API client for Python 3.x


Tested against the latest release, HEAD ref, and 3 previous minor versions (counting back from the latest release) of Vault. Current official support covers Vault v1.4.7 or later.

## 1.1 Installation

```
pip install hvac
```

If you would like to be able to return parsed HCL data as a Python dict for methods that support it:

```
pip install "hvac[parser]"
```

## 1.2 Getting Started

### 1.2.1 Initialize the Client

Using TLS:

```
>>> client = hvac.Client(url='https://localhost:8200')
>>> client.is_authenticated()
True
```

Using TLS with client-side certificate authentication:

```
>>> client = hvac.Client(
...     url='https://localhost:8200',
...     token=os.environ['VAULT_TOKEN'],
...     cert=(client_cert_path, client_key_path),
...     verify=server_cert_path,
... )
>>> client.is_authenticated()
True
```

Using Vault Enterprise namespace:

```
>>> client = hvac.Client(
...     url='https://localhost:8200',
...     namespace=os.getenv('VAULT_NAMESPACE'),
... )
```

Using plaintext / HTTP (not recommended for anything other than development work):

```
>>> client = hvac.Client(url='http://localhost:8200')
```

### 1.2.2 Vault Cluster - Initialize and Seal/Unseal

```
>>> client.sys.is_initialized()
False

>>> shares = 5
>>> threshold = 3
>>> result = client.sys.initialize(shares, threshold)
>>> root_token = result['root_token']
>>> keys = result['keys']
>>> client.sys.is_initialized()
```

(continues on next page)

(continued from previous page)

```
True

>>> client.token = root_token

>>> client.sys.is_sealed()
True
>>> # Unseal a Vault cluster with individual keys
>>> unseal_response1 = client.sys.submit_unseal_key(keys[0])
>>> unseal_response2 = client.sys.submit_unseal_key(keys[1])
>>> unseal_response3 = client.sys.submit_unseal_key(keys[2])
>>> client.sys.is_sealed()
False
>>> # Seal a previously unsealed Vault cluster.
>>> client.sys.seal()
<Response [204]>
>>> client.sys.is_sealed()
True

>>> # Unseal with multiple keys until threshold met
>>> unseal_response = client.sys.submit_unseal_keys(keys)

>>> client.sys.is_sealed()
False
```

### 1.2.3 Read and write to secrets engines

**Note:** Vault currently defaults the *secret/* path to the KV secrets engine *version 2* automatically when the Vault server is started in "dev" mode.

**Note:** Starting with Vault v1.1.0, _no_ KV secrets engine is mounted by default. I.e., outside of dev mode, a KV engine mounted under path *secret/* must be explicitly enabled before use.

#### KV Secrets Engine - Version 2

```
>>> # Retrieve an authenticated hvac.Client() instance
>>> client = test_utils.create_client()
>>>
>>> # Write a k/v pair under path: secret/foo
>>> create_response = client.secrets.kv.v2.create_or_update_secret(
...     path='foo',
...     secret=dict(baz='bar'),
... )
>>>
>>> # Read the data written under path: secret/foo
>>> read_response = client.secrets.kv.read_secret_version(path='foo')
>>> print('Value under path "secret/foo" / key "baz": {val}'.format(
...     val=read_response['data']['data']['baz'],
... ))
Value under path "secret/foo" / key "baz": bar
```

(continues on next page)

(continued from previous page)

```
>>>
>>> # Delete all metadata/versions for path: secret/foo
>>> client.secrets.kv.delete_metadata_and_all_versions('foo')
<Response [204]>
```

#### KV Secrets Engine - Version 1

Preferred usage:

```
>>> create_response = client.secrets.kv.v1.create_or_update_secret('foo',␣
↪secret=dict(baz='bar'))
>>> read_response = client.secrets.kv.v1.read_secret('foo')
>>> print('Value under path "secret/foo" / key "baz": {val}'.format(
...     val=read_response['data']['baz'],
... ))
Value under path "secret/foo" / key "baz": bar
>>> delete_response = client.secrets.kv.v1.delete_secret('foo')
```

### 1.2.4 Authentication

#### Basic Token Authentication

```
# Token
>>> client.token = os.environ['VAULT_TOKEN']
>>> client.is_authenticated()
True
```

#### LDAP Authentication Example

```
>>> client = hvac.Client(url='https://localhost:8200')
>>> # LDAP, getpass -> user/password, bring in LDAP3 here for teststup?
>>> login_response = client.auth.ldap.login(
...     username=os.environ['LDAP_USERNAME'],
...     password=os.environ['LDAP_PASSWORD'],
... )
>>> client.is_authenticated()
True
>>> print('The client token returned from the LDAP auth method is: {token}'.format(
...     token=login_response['auth']['client_token']
... ))
The client token returned from the LDAP auth method is: ...
```

CHAPTER

TWO

# USAGE

## 2.1 Secrets Engines

### 2.1.1 Active Directory

**Contents**



#### Configure AD Secrets Secrets Engine

Configure the AD secrets engine to either manage service accounts or service account libraries.

Source reference: `hvac.api.secrets_engines.activedirectory.configure()`

```
import hvac
client = hvac.Client()

# Authenticate to Vault using client.auth.x

# Not all these settings may apply to your setup, refer to Vault
# documentation for context of what to use here

config_response = client.secrets.activedirectory.configure(
    binddn='username@domain.fqdn', # A upn or DN can be used for this value, Vault␣
↪resolves the user to a dn silently
    bindpass='***********',
    url='ldaps://domain.fqdn',
    userdn='CN=Users,DN=domain,DN=fqdn',
```

(continues on next page)

(continued from previous page)

```
    upndomain='domain.fqdn',
    ttl=60,
    max_ttl=120
)
print(config_response)
```

### Read Config

Return the AD Secret Engine configuration.

Source reference: `hvac.api.secrets_engines.activedirectory.read_config()`

```
import hvac
client = hvac.Client()

# Authenticate to Vault using client.auth.x

config_response = client.secrets.activedirectory.read_config()
```

### Create or Update Role

Create or Update a role which allows the retrieval and rotation of an AD account. Retrieve and rotate the actual credential via generate_credentials().

Source reference: `hvac.api.secrets_engines.activedirectory.create_or_update_role()`

```
import hvac
client = hvac.Client()

# Authenticate to Vault using client.auth.x

role_response = client.secrets.activedirectory.create_or_update_role(
    name='sql-service-account',
    service_account_name='svc-sqldb-petshop@domain.fqdn',
    ttl=60)
```

### Read Role

Retrieve the role configuration which allows the retrieval and rotation of an AD account. Retrieve and rotate the actual credential via generate_credentials().

Source reference: `hvac.api.secrets_engines.activedirectory.read_role()`

```
import hvac
client = hvac.Client()

# Authenticate to Vault using client.auth.x

role_response = client.secrets.activedirectory.read_role(name='sql-service-account')
```

### List Roles

List all configured roles which allows the retrieval and rotation of an AD account. Retrieve and rotate the actual credential via generate_credentials().

Source reference: `hvac.api.secrets_engines.activedirectory.list_roles()`

```
import hvac
client = hvac.Client()

# Authenticate to Vault using client.auth.x

all_roles = client.secrets.activedirectory.list_roles()
```

### Delete Role

Remove the role configuration which allows the retrieval and rotation of an AD account.

The account is retained in Active Directory, but the password will be whatever Vault had rotated it to last. To regain control, the password will need to be reset via Active Directory.

Source reference: `hvac.api.secrets_engines.activedirectory.delete_role()`

```
import hvac
client = hvac.Client()

# Authenticate to Vault using client.auth.x

deletion_response = client.secrets.activedirectory.delete_role(name='sql-service-
↪account')
```

### Generate Credentials

Retrieve a service account password from AD. Return the previous password (if known). Vault shall rotate the password before returning it, if it has breached its configured ttl.

Source reference: `hvac.api.secrets_engines.activedirectory.generate_credentials()`

```
import hvac
client = hvac.Client()

# Authenticate to Vault using client.auth.x

gen_creds_response = client.secrets.activedirectory.generate_credentials(
    name='hvac-role',
)
print('Retrieved Service Account Password: {access} (Current) / {secret} (Old)'.
↪format(
    access=gen_creds_response['data']['current_password'],
    secret=gen_creds_response['data']['old_password'],
))
```

### 2.1.2 AWS

**Contents**



#### Configure Root IAM Credentials

Source reference: *hvac.api.secrets_engines.Aws.configure_root_iam_credentials()*

```
import os

import hvac
client = hvac.Client()

client.secrets.aws.configure_root_iam_credentials(
    access_key=os.getenv('AWS_ACCESS_KEY_ID'),
    secret_key=os.getenv('AWS_SECRET_ACCESS_KEY'),
)
```

### Rotate Root IAM Credentials

Source reference: *hvac.api.secrets_engines.Aws.rotate_root_iam_credentials()*

```
import hvac
client = hvac.Client()

client.secrets.aws.rotate_root_iam_credentials()
```

### Configure Lease

Source reference: *hvac.api.secrets_engines.Aws.configure_lease()*

```
import hvac
client = hvac.Client()

# Set the default least TTL to 300 seconds / 5 minutes
client.secrets.aws.configure_lease(
    lease='300s',
)
```

### Read Lease

Source reference: `hvac.api.secrets_engines.Aws.read_lease()`

```
import hvac
client = hvac.Client()

read_lease_response = client.secrets.aws.read_lease()
print('The current "lease_max" TTL is: {lease_max}'.format(
    lease_max=read_lease_response['data']['lease_max'],
))
```

### Create or Update Role

Source reference: *hvac.api.secrets_engines.Aws.create_or_update_role()*

```
import hvac
client = hvac.Client()

describe_ec2_policy_doc = {
    'Version': '2012-10-17',
    'Statement': [
        {
            'Resource': '*'
            'Action': 'ec2:Describe*',
            'Effect': 'Allow',
        },
    ],
}
client.secrets.aws.create_or_update_role(
    name='hvac-role',
    credential_type='assumed_role',
```

(continues on next page)

(continued from previous page)

```
    policy_document=describe_ec2_policy_doc,
    policy_arns=['arn:aws:iam::aws:policy/AmazonVPCReadOnlyAccess'],
)
```

### Legacy Parameters

**Note:** In previous versions of Vault (before version 0.11.0), this API route only supports the *policy_document* and *policy_arns* parameters (which hvac will translate to *policy* and *arn* parameters respectively in the request sent to Vault). If running these versions of Vault, the *legacy_params* parameter on this method can be set to *True*.

For older versions of Vault (any version before 0.11.0):

```
import hvac
client = hvac.Client()

describe_ec2_policy_doc = {
    'Version': '2012-10-17',
    'Statement': [
        {
            'Resource': '*'
            'Action': 'ec2:Describe*',
            'Effect': 'Allow',
        },
    ],
}

# Note: with the legacy params, the `policy_arns` parameter is translated to `arn`
# in the request sent to Vault and only one ARN is accepted. If a list is provided,
# hvac will only use the first ARN in the list.
client.secrets.aws.create_or_update_role(
    name='hvac-role',
    credential_type='assumed_role',
    policy_document=describe_ec2_policy_doc,
    policy_arns='arn:aws:iam::aws:policy/AmazonVPCReadOnlyAccess',
    legacy_params=True,
)
```

### Read Role

Source reference: *hvac.api.secrets_engines.Aws.read_role()*

```
import hvac
client = hvac.Client()

read_role_response = client.secrets.aws.read_role(
    name='hvac-role',
)
print('The credential type for role "hvac-role" is: {cred_type}'.format(
    cred_type=read_role_response['data']['credential_types'],
))
```

#### List Roles

Source reference: *hvac.api.secrets_engines.Aws.list_roles()*

```
import hvac
client = hvac.Client()

list_roles_response = client.secrets.aws.list_roles()
print('AWS secrets engine role listing: {roles}'.format(
    roles=', '.join(list_roles_response['data']['keys'])
))
```

#### Delete Role

Source reference: *hvac.api.secrets_engines.Aws.delete_role()*

```
import hvac
client = hvac.Client()

client.secrets.aws.delete_role(
    name='hvac-role',
)
```

#### Generate Credentials

Source reference: *hvac.api.secrets_engines.Aws.generate_credentials()*

```
import hvac
client = hvac.Client()

gen_creds_response = client.secrets.aws.generate_credentials(
    name='hvac-role',
)
print('Generated access / secret keys: {access} / {secret}'.format(
    access=gen_creds_response['data']['access_key'],
    secret=gen_creds_response['data']['secret_key'],
))
```

### 2.1.3 Azure

**Note:** Every method under the *Azure class* includes a *mount_point* parameter that can be used to address the Azure secret engine under a custom mount path. E.g., If enabling the Azure secret engine using Vault's CLI commands via *vault secrets enable -path=my-azure azure*", the *mount_point* parameter in *hvac.api.secrets_engines.Azure()* methods would need to be set to "my-azure".

### Configure

*hvac.api.secrets_engines.Azure.configure()*

```
import hvac
client = hvac.Client()

client.secrets.azure.configure(
    subscription_id='my-subscription-id',
    tenant_id='my-tenant-id',
)
```

### Read Config

*hvac.api.secrets_engines.Azure.read_config()*

```
import hvac
client = hvac.Client()

azure_secret_config = client.secrets.azure.read_config()
print('The Azure secret engine is configured with a subscription ID of {id}'.format(
    id=azure_secret_config['subscription_id'],
))
```

### Delete Config

*hvac.api.secrets_engines.Azure.delete_config()*

```
import hvac
client = hvac.Client()

client.secrets.azure.delete_config()
```

### Create Or Update A Role

*hvac.api.secrets_engines.Azure.create_or_update_role()*

```
import hvac
client = hvac.Client()


azure_roles = [
    {
        'role_name': "Contributor",
        'scope': "/subscriptions/95e675fa-307a-455e-8cdf-0a66aeaa35ae",
    },
]
client.secrets.azure.create_or_update_role(
    name='my-azure-secret-role',
    azure_roles=azure_roles,
)
```

#### List Roles

*hvac.api.secrets_engines.Azure.list_roles()*

```
import hvac
client = hvac.Client()

azure_secret_engine_roles = client.secrets.azure.list_roles()
print('The following Azure secret roles are configured: {roles}'.format(
    roles=','.join(roles['keys']),
))
```

#### Generate Credentials

*hvac.api.secrets_engines.Azure.generate_credentials()*

```
import hvac
from azure.common.credentials import ServicePrincipalCredentials

client = hvac.Client()
azure_creds = client.secrets.azure.secret.generate_credentials(
    name='some-azure-role-name',
)
azure_spc = ServicePrincipalCredentials(
    client_id=azure_creds['client_id'],
    secret=azure_creds['client_secret'],
    tenant=TENANT_ID,
)
```

### 2.1.4 GCP

- *Configure*
- *Read Config*
- *Create Or Update Roleset*
- *Rotate Roleset Account*
- *Rotate Roleset Account Key*
- *Read Roleset*
- *List Rolesets*
- *Delete Roleset*
- *Generate Oauth2 Access Token*
- *Generate Service Account Key*

### Configure

Gcp.**configure**(*credentials=None*, *ttl=None*, *max_ttl=None*, *mount_point='gcp'*)

Configure shared information for the Gcp secrets engine.

**Supported methods:** POST: /{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **credentials** (*str | unicode*) – JSON credentials (either file contents or '@path/to/file') See docs for alternative ways to pass in to this parameter, as well as the required permissions.
- **ttl** (*int | str*) – – Specifies default config TTL for long-lived credentials (i.e. service account keys). Accepts integer number of seconds or Go duration format string.
- **max_ttl** (*int | str*) – Specifies the maximum config TTL for long-lived credentials (i.e. service account keys). Accepts integer number of seconds or Go duration format string.**
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')


credentials = test_utils.load_config_file('example.jwt.json')
configure_response = client.secrets.gcp.configure(
    credentials=credentials,
    max_ttl=3600,
)
print(configure_response)
```

Example output:

```
<Response [204]>
```

### Read Config

Gcp.**read_config**(*mount_point='gcp'*)

Read the configured shared information for the Gcp secrets engine.

Credentials will be omitted from returned data.

**Supported methods:** GET: /{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

read_config_response = client.secrets.gcp.read_config()
print('Max TTL for GCP secrets engine set to: {max_ttl}'.format(max_ttl=read_config_
↪response['data']['max_ttl']))
```

Example output:

```
Max TTL for GCP secrets engine set to: 3600
```

### Create Or Update Roleset

Gcp.**create_or_update_roleset**(*name*, *project*, *bindings*, *secret_type=None*, *token_scopes=None*, *mount_point='gcp'*)

Create a roleset or update an existing roleset.

**See roleset docs for the GCP secrets backend to learn more about what happens when you create or update a** roleset.

**Supported methods:** POST: /{mount_point}/roleset/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role. Cannot be updated.
- **project** (*str | unicode*) – Name of the GCP project that this roleset's service account will belong to. Cannot be updated.
- **bindings** (*str | unicode*) – Bindings configuration string (expects HCL or JSON format in raw or base64-encoded string)
- **secret_type** (*str | unicode*) – Cannot be updated.
- **token_scopes** (*list[str]*) – List of OAuth scopes to assign to access_token secrets generated under this role set (access_token role sets only)
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')


bindings = """
    resource "//cloudresourcemanager.googleapis.com/project/some-gcp-project-id" {
      roles = [
        "roles/viewer"
      ],
    }
```

(continues on next page)

(continued from previous page)

```
"""
token_scopes = [
    'https://www.googleapis.com/auth/cloud-platform',
    'https://www.googleapis.com/auth/bigquery',
]

roleset_response = client.secrets.gcp.create_or_update_roleset(
    name='hvac-doctest',
    project='some-gcp-project-id',
    bindings=bindings,
    token_scopes=token_scopes,
)
```

### Rotate Roleset Account

Gcp.**rotate_roleset_account**(*name*, *mount_point='gcp'*)

Rotate the service account this roleset uses to generate secrets.

**This also replaces the key access_token roleset. This can be used to invalidate old secrets generated by the** roleset or fix issues if a roleset's service account (and/or keys) was changed outside of Vault (i.e. through GCP APIs/cloud console).

**Supported methods:** POST: /{mount_point}/roleset/{name}/rotate. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

rotate_response = client.secrets.gcp.rotate_roleset_account(name='hvac-doctest')
```

### Rotate Roleset Account Key

Gcp.**rotate_roleset_account_key**(*name*, *mount_point='gcp'*)

Rotate the service account key this roleset uses to generate access tokens.

This does not recreate the roleset service account.

**Supported methods:** POST: /{mount_point}/roleset/{name}/rotate-key. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

rotate_response = client.secrets.gcp.rotate_roleset_account_key(name='hvac-doctest')
```

### Read Roleset

Gcp.**read_roleset**(*name*, *mount_point='gcp'*)

Read a roleset.

**Supported methods:** GET: /{mount_point}/roleset/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

read_response = client.secrets.gcp.read_roleset(name='hvac-doctest')
```

### List Rolesets

Gcp.**list_rolesets**(*mount_point='gcp'*)

List configured rolesets.

**Supported methods:** LIST: /{mount_point}/rolesets. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

list_response = client.secrets.gcp.list_rolesets()
```

### Delete Roleset

Gcp.**delete_roleset**(*name*, *mount_point='gcp'*)

Delete an existing roleset by the given name.

**Supported methods:** DELETE: /{mount_point}/roleset/{name} Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

delete_response = client.secrets.gcp.delete_roleset(name='hvac-doctest')
```

### Generate Oauth2 Access Token

Gcp.**generate_oauth2_access_token**(*roleset*, *mount_point='gcp'*)

Generate an OAuth2 token with the scopes defined on the roleset.

This OAuth access token can be used in GCP API calls, e.g. curl -H "Authorization: Bearer $TOKEN" ...

**Supported methods:** GET: /{mount_point}/token/{roleset}. Produces: 200 application/json

**Parameters**

- **roleset** (*str | unicode*) – Name of an roleset with secret type access_token to generate access_token under.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

token_response = client.secrets.gcp.generate_oauth2_access_token(roleset='hvac-doctest
↪')
```

### Generate Service Account Key

Gcp.**generate_service_account_key**(*roleset*, *key_algorithm='KEY_ALG_RSA_2048'*, *key_type='TYPE_GOOGLE_CREDENTIALS_FILE'*, *method='POST'*, *mount_point='gcp'*)

Generate Secret (IAM Service Account Creds): Service Account Key

**If using GET ('read'), the optional parameters will be set to their defaults. Use POST if you want to specify** different values for these params.

**Parameters**

- **roleset** (*str | unicode*) – Name of an roleset with secret type service_account_key to generate key under.
- **key_algorithm** (*str | unicode*) – Key algorithm used to generate key. Defaults to 2k RSA key You probably should not choose other values (i.e. 1k),
- **key_type** (*str | unicode*) – Private key type to generate. Defaults to JSON credentials file.
- **method** (*str | unicode*) – Supported methods: POST: /{mount_point}/key/{roleset}. Produces: 200 application/json GET: /{mount_point}/key/{roleset}. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

key_response = client.secrets.gcp.generate_service_account_key(roleset='hvac-doctest')
```

### 2.1.5 Identity

New in version Vault: 0.9.0

**Contents**

### Entity

#### Create Or Update Entity

*hvac.api.secrets_engines.Identity.create_or_update_entity()*

```
import hvac
client = hvac.Client()

create_response = client.secrets.identity.create_or_update_entity(
                name='hvac-entity',
                metadata=dict(extra_datas='yup'),
        )
entity_id = create_response['data']['id']
print('Entity ID for "hvac-entity" is: {id}'.format(id=entity_id))
```

### Create Or Update Entity By Name

*hvac.api.secrets_engines.Identity.create_or_update_entity_by_name()*

```
import hvac
client = hvac.Client()

client.secrets.identity.create_or_update_entity_by_name(
        name='hvac-entity',
        metadata=dict(new_datas='uhuh'),
)
```

### Read Entity

*hvac.api.secrets_engines.Identity.read_entity()*

```
import hvac
client = hvac.Client()

read_response = client.secrets.identity.read_entity(
        entity_id=entity_id,
)
name = read_response['data']['name']
print('Name for entity ID {id} is: {name}'.format(id=entity_id, name=name))
```

### Read Entity By Name

New in version Vault: 0.11.2

*hvac.api.secrets_engines.Identity.read_entity_by_name()*

```
import hvac
client = hvac.Client()

read_response = client.secrets.identity.read_entity_by_name(
        name='hvac-entity',
)
entity_id = read_response['data']['id']
print('Entity ID for "hvac-entity" is: {id}'.format(id=entity_id))
```

### Update Entity

*hvac.api.secrets_engines.Identity.update_entity()*

```
import hvac
client = hvac.Client()

client.secrets.identity.update_entity(
        entity_id=entity_id,
        metadata=dict(new_metadata='yup'),
)
```

### Delete Entity

*hvac.api.secrets_engines.Identity.delete_entity()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_entity(
        entity_id=entity_id,
)
```

### Delete Entity By Name

New in version Vault: 0.11.2

*hvac.api.secrets_engines.Identity.delete_entity_by_name()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_entity_by_name(
        name='hvac-entity',
)
```

### List Entities

*hvac.api.secrets_engines.Identity.list_entities()*

```
import hvac
client = hvac.Client()

list_response = client.secrets.identity.list_entities()
entity_keys = list_response['data']['keys']
print('The following entity IDs are currently configured: {keys}'.format(keys=entity_
↪keys))
```

### List Entities By Name

New in version Vault: 0.11.2

*hvac.api.secrets_engines.Identity.list_entities_by_name()*

```
import hvac
client = hvac.Client()

list_response = client.secrets.identity.list_entities_by_name()
entity_keys = list_response['data']['keys']
print('The following entity names are currently configured: {keys}'.
↪format(keys=entity_keys))
```

### Merge Entities

*hvac.api.secrets_engines.Identity.merge_entities()*

```
import hvac
client = hvac.Client()

client.secrets.identity.merge_entities(
        from_entity_ids=from_entity_ids,
        to_entity_id=to_entity_id,
)
```

### Entity Alias

### Create Or Update Entity Alias

*hvac.api.secrets_engines.Identity.create_or_update_entity_alias()*

```
import hvac
client = hvac.Client()

create_response = client.secrets.identity.create_or_update_entity_alias(
        name='hvac-entity-alias',
        canonical_id=entity_id,
        mount_accessor='auth_approle_73c16de3',
)
alias_id = create_response['data']['id']
print('Alias ID for "hvac-entity-alias" is: {id}'.format(id=alias_id))
```

### Read Entity Alias

*hvac.api.secrets_engines.Identity.read_entity_alias()*

```
import hvac
client = hvac.Client()

read_response = client.secrets.identity.read_entity_alias(
        alias_id=alias_id,
)
name = read_response['data']['name']
print('Name for entity alias {id} is: {name}'.format(id=alias_id, name=name))
```

### Update Entity Alias

*hvac.api.secrets_engines.Identity.update_entity_alias()*

```
import hvac
client = hvac.Client()

client.secrets.identity.update_entity_alias(
        alias_id=alias_id,
        name='new-alias-name',
```

(continues on next page)

(continued from previous page)

```
        canonical_id=entity_id,
        mount_accessor='auth_approle_73c16de3',
)
```

### List Entity Aliases

*hvac.api.secrets_engines.Identity.list_entity_aliases()*

```
import hvac
client = hvac.Client()

list_response = client.secrets.identity.list_entity_aliases()
alias_keys = list_response['data']['keys']
print('The following entity alias IDs are currently configured: {keys}'.
↪format(keys=alias_keys))
```

### Delete Entity Alias

*hvac.api.secrets_engines.Identity.delete_entity_alias()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_entity_alias(
        alias_id=alias_id,
)
```

### Group

### Create Or Update Group

*hvac.api.secrets_engines.Identity.create_or_update_group()*

```
import hvac
client = hvac.Client()

create_response = client.secrets.identity.create_or_update_group(
        name='hvac-group',
        metadata=dict(extra_datas='we gots em'),
)
group_id = create_response['data']['id']
print('Group ID for "hvac-group" is: {id}'.format(id=group_id))
```

### Read Group

*hvac.api.secrets_engines.Identity.read_group()*

```
import hvac
client = hvac.Client()

read_response = client.secrets.identity.read_group(
        group_id=group_id,
)
name = read_response['data']['name']
print('Name for group ID {id} is: {name}'.format(id=group_id, name=name))
```

### Update Group

*hvac.api.secrets_engines.Identity.update_group()*

```
import hvac
client = hvac.Client()

client.secrets.identity.update_group(
        group_id=group_id,
        metadata=dict(new_metadata='yup'),
)
```

### Delete Group

*hvac.api.secrets_engines.Identity.delete_group()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_group(
        group_id=group_id,
)
```

### List Groups

*hvac.api.secrets_engines.Identity.list_groups()*

```
import hvac
client = hvac.Client()

list_response = client.secrets.identity.list_groups()
group_keys = list_entities_response['data']['keys']
print('The following group IDs are currently configured: {keys}'.format(keys=group_
↪keys))
```

### List Groups By Name

New in version Vault: 0.11.2

*hvac.api.secrets_engines.Identity.list_groups_by_name()*

```
import hvac
client = hvac.Client()

list_response = client.secrets.identity.list_groups_by_name()
group_keys = list_response['data']['keys']
print('The following group names are currently configured: {keys}'.format(keys=group_
↪keys))
```

### Create Or Update Group By Name

New in version Vault: 0.11.2

*hvac.api.secrets_engines.Identity.create_or_update_group_by_name()*

```
import hvac
client = hvac.Client()

client.secrets.identity.create_or_update_group_by_name(
        name='hvac-group',
        metadata=dict(new_datas='uhuh'),
)
```

### Read Group By Name

New in version Vault: 0.11.2

*hvac.api.secrets_engines.Identity.read_group_by_name()*

```
import hvac
client = hvac.Client()

read_response = client.secrets.identity.read_group_by_name(
        name='hvac-group',
)
group_id = read_response['data']['id']
print('Group ID for "hvac-group" is: {id}'.format(id=group_id))
```

### Delete Group By Name

New in version Vault: 0.11.2

*hvac.api.secrets_engines.Identity.delete_group_by_name()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_group_by_name(
```

(continues on next page)

(continued from previous page)

```
        name='hvac-group',
)
```

### Group Alias

#### Create Or Update Group Alias

*hvac.api.secrets_engines.Identity.create_or_update_group_alias()*

```
import hvac
client = hvac.Client()

create_response = client.secrets.identity.create_or_update_group_alias(
                name='hvac-group-alias',
                canonical_id=group_id,
                mount_accessor='auth_approle_73c16de3',
        )
alias_id = create_response['data']['id']
print('Group alias ID for "hvac-group_alias" is: {id}'.format(id=alias_id))
```

#### Update Group Alias

*hvac.api.secrets_engines.Identity.update_group_alias()*

```
import hvac
client = hvac.Client()

client.secrets.identity.update_group_alias(
        alias_id=alias_id,
        name='new-alias-name',
        canonical_id=group_id,
        mount_accessor='auth_approle_73c16de3',
)
```

#### Read Group Alias

*hvac.api.secrets_engines.Identity.read_group_alias()*

```
import hvac
client = hvac.Client()

read_response = client.secrets.identity.read_group_alias(
        alias_id=alias_id,
)
name = read_response['data']['name']
print('Name for group alias {id} is: {name}'.format(id=alias_id, name=name))
```

### Delete Group Alias

*hvac.api.secrets_engines.Identity.delete_group_alias()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_group_alias(
        alias_id=alias_id,
)
```

### List Group Aliases

*hvac.api.secrets_engines.Identity.list_group_aliases()*

```
import hvac
client = hvac.Client()

list_response = client.secrets.identity.list_group_aliases()
alias_keys = list_response['data']['keys']
print('The following group alias IDs are currently configured: {keys}'.
↪format(keys=alias_keys))
```

### Lookup

### Lookup Entity

*hvac.api.secrets_engines.Identity.lookup_entity()*

```
import hvac
client = hvac.Client()

lookup_response = client.secrets.identity.lookup_entity(
        name='hvac-entity',
)
entity_id = lookup_response['data']['id']
print('Entity ID for "hvac-entity" is: {id}'.format(id=entity_id))
```

### Lookup Group

*hvac.api.secrets_engines.Identity.lookup_group()*

```
import hvac
client = hvac.Client()

lookup_response = client.secrets.identity.lookup_group(
        name='hvac-group',
)
group_id = lookup_response['data']['id']
print('Group ID for "hvac-entity" is: {id}'.format(id=group_id))
```

### Tokens

#### Configure Tokens Backend

*hvac.api.secrets_engines.Identity.configure_tokens_backend()*

```
import hvac
client = hvac.Client()

client.secrets.identity.configure_tokens_backend(
        issuer='https://python-hvac.org:1234',
)
```

#### Read Tokens Backend Configuration

*hvac.api.secrets_engines.Identity.read_tokens_backend_configuration()*

```
import hvac
client = hvac.Client()

config = client.secrets.identity.read_tokens_backend_configuration()
print('Tokens backend issuer: {issuer}'.format(issuer=config['data']['issuer']))
```

#### Create Named Key

*hvac.api.secrets_engines.Identity.create_named_key()*

```
import hvac
client = hvac.Client()

client.secrets.identity.create_named_key(
        name='hvac',
)
```

#### Read Named Key

*hvac.api.secrets_engines.Identity.read_named_key()*

```
import hvac
client = hvac.Client()

key_response = client.secrets.identity.read_named_key(
        name='hvac',
)
print('Identity key "hvac" algorithm is: {algorithm}'.format(
        algorithm=response['data']['algorithm'],
))
```

### Delete Named Key

*hvac.api.secrets_engines.Identity.delete_named_key()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_named_key(
        name='hvac',
)
```

### List Named Keys

*hvac.api.secrets_engines.Identity.delete_named_key()*

```
import hvac
client = hvac.Client()

list_keys_resp = client.secrets.identity.list_named_keys()
print('Current token key names: {names}'.format(
        names=', '.join(response['data']['keys']),
))
```

### Rotate Named Key

*hvac.api.secrets_engines.Identity.rotate_named_key()*

```
import hvac
client = hvac.Client()

client.secrets.identity.rotate_named_key(
        name='hvac',
        verification_ttl='24h',
)
```

### Create or Update Role

*hvac.api.secrets_engines.Identity.create_or_update_role()*

```
import hvac
client = hvac.Client()

key_name = 'hvac-key'
token_client_id = 'some-client-id'
client.secrets.identity.create_named_key(
        name=key_name,
        allowed_client_ids=[token_client_id],
)
client.secrets.identity.create_or_update_role(
        name='hvac-person',
        key_name=key_name,
        client_id=token_client_id,
)
```

### Read Role

*hvac.api.secrets_engines.Identity.create_or_update_role()*

```
import hvac
client = hvac.Client()

read_resp = client.secrets.identity.read_role(
        name='hvac-person',
)
print('Identity role "hvac-person" is set to use key: {key_name}'.format(
        key_name=read_resp['data']['key'],
))
```

### Delete Role

*hvac.api.secrets_engines.Identity.delete_role()*

```
import hvac
client = hvac.Client()

client.secrets.identity.delete_role(
        name='hvac-person',
)
```

### List Roles

*hvac.api.secrets_engines.Identity.list_roles()*

```
import hvac
client = hvac.Client()

response = client.secrets.identity.list_roles()
print('Current token role names: {names}'.format(
        names=', '.join(response['data']['keys']),
))
```

### Generate Signed ID Token

*hvac.api.secrets_engines.Identity.generate_signed_id_token()*

```
import hvac
client = hvac.Client()

# Note: the token attribute on the following Client instance must have an
# identity associated with it. Otherwise the request will be reject by vault due to:
# "no entity associated with the request's token"
response = client.secrets.identity.generate_signed_id_token(
        name='hvac-person',
)
print('Generated signed id token: {token}'.format(
        token=response['data']['token'],
))
```

#### Introspect Signed ID Token

*hvac.api.secrets_engines.Identity.introspect_signed_id_token()*

```
import hvac
client = hvac.Client()

response = client.secrets.identity.introspect_signed_id_token(
        token='some-generated-signed-id-token',
)
print('Specified token is active?: {active}'.format(
        active=response['active'],
))
```

#### Read .well-known Configurations

*hvac.api.secrets_engines.Identity.read_well_known_configurations()*

```
import hvac
client = hvac.Client()

response = client.secrets.identity.read_well_known_configurations()
print('JWKS URI is: {jwks_uri}'.format(
        active=response['jwks_uri'],
))
```

#### Read Active Public Keys

*hvac.api.secrets_engines.Identity.read_active_public_keys()*

```
import hvac
client = hvac.Client()

response = client.secrets.identity.read_active_public_keys()
print('Active public keys: {keys}'.format(
        keys=response['keys'],
))
```

### 2.1.6 PKI

#### Read CA Certificate

hvac.api.secrets_engines.pki.read_ca_certificate()

```
import hvac
client = hvac.Client()

read_ca_certificate_response = client.secrets.pki.read_ca_certificate()
print('Current PKI CA Certificate: {}'.format(read_ca_certificate_response))
```

### Read CA Certificate Chain

hvac.api.secrets_engines.pki.read_ca_certificate_chain()

```
import hvac
client = hvac.Client()

read_ca_certificate_chain_response = client.secrets.pki.read_ca_certificate_chain()
print('Current PKI CA Certificate Chain: {}'.format(read_ca_certificate_chain_
↪response))
```

### Read Certificate

hvac.api.secrets_engines.pki.read_certificate()

```
import hvac
client = hvac.Client()

read_certificate_response = client.secrets.pki.read_certificate(serial='crl')
print('Current PKI CRL: {}'.format(read_certificate_response))
```

### List Certificates

hvac.api.secrets_engines.pki.list_certificates()

```
import hvac
client = hvac.Client()

list_certificate_response = client.secrets.pki.list_certificates()
print('Current certificates (serial numbers): {}'.format(list_certificate_response))
```

### Submit CA Information

hvac.api.secrets_engines.pki.submit_ca_information()

```
import hvac
client = hvac.Client()

submit_ca_information_response = client.secrets.pki.submit_ca_information(
'-----BEGIN RSA PRIVATE KEY-----\n...\n-----END CERTIFICATE-----'
)
```

### Read CRL Configuration

hvac.api.secrets_engines.pki.read_crl_configuration()

```
import hvac
client = hvac.Client()

read_crl_configuration_response = client.secrets.pki.read_crl_configuration()
print('CRL configuration: {}'.format(read_crl_configuration_response))
```

### Set CRL Configuration

hvac.api.secrets_engines.pki.set_crl_configuration()

```
import hvac
client = hvac.Client()

set_crl_configuration_response = client.secrets.pki.set_crl_configuration(
   expiry='72h',
   disable=False
)
```

### Read URLs

hvac.api.secrets_engines.pki.read_urls()

```
import hvac
client = hvac.Client()

read_urls_response = client.secrets.pki.read_urls()
print('Get PKI urls: {}'.format(read_urls_response))
```

### Set URLs

hvac.api.secrets_engines.pki.set_urls()

```
import hvac
client = hvac.Client()

set_urls_response = client.secrets.pki.set_urls(
{
  'issuing_certificates': ['http://127.0.0.1:8200/v1/pki/ca'],
  'crl_distribution_points': ['http://127.0.0.1:8200/v1/pki/crl']
}
)
```

### Read CRL

hvac.api.secrets_engines.pki.read_crl()

```
import hvac
client = hvac.Client()

read_crl_response = client.secrets.pki.read_crl()
print('Current CRL: {}'.format(read_crl_response))
```

### Rotate CRLs

hvac.api.secrets_engines.pki.rotate_crl()

```
import hvac
client = hvac.Client()

rotate_crl_response = client.secrets.pki.rotate_crl()
print('Rotate CRL: {}'.format(rotate_crl_response))
```

### Generate Intermediate

hvac.api.secrets_engines.pki.generate_intermediate()

```
import hvac
client = hvac.Client()

generate_intermediate_response = client.secrets.pki.generate_intermediate(
    type='exported',
    common_name='Vault integration tests'
)
print('Intermediate certificate: {}'.format(generate_intermediate_response))
```

### Set Signed Intermediate

hvac.api.secrets_engines.pki.set_signed_intermediate()

```
import hvac
client = hvac.Client()

set_signed_intermediate_response = client.secrets.pki.set_signed_intermediate(
    '-----BEGIN CERTIFICATE...'
)
```

### Generate Certificate

hvac.api.secrets_engines.pki.generate_certificate()

```
import hvac
client = hvac.Client()

generate_certificate_response = client.secrets.pki.generate_certificate(
   name='myrole',
   common_name='test.example.com'
)
print('Certificate: {}'.format(generate_certificate_response))
```

### Revoke Certificate

hvac.api.secrets_engines.pki.revoke_certificate()

```
import hvac
client = hvac.Client()

revoke_certificate_response = client.secrets.pki.revoke_certificate(
   serial_number='39:dd:2e...'
)
print('Certificate: {}'.format(revoke_certificate_response))
```

### Create/Update Role

hvac.api.secrets_engines.pki.create_or_update_role()

```
import hvac
client = hvac.Client()

create_or_update_role_response = client.secrets.pki.create_or_update_role(
   'mynewrole',
   {
      'ttl': '72h',
      'allow_localhost': 'false'
   }
)
print('New role: {}'.format(create_or_update_role_response))
```

### Read Role

hvac.api.secrets_engines.pki.read_role()

```
import hvac
client = hvac.Client()

read_role_response = client.secrets.pki.read_role('myrole')
print('Role definition: {}'.format(read_role_response))
```

### List Roles

hvac.api.secrets_engines.pki.list_roles()

```
import hvac
client = hvac.Client()

list_roles_response = client.secrets.pki.list_roles()
print('List of available roles: {}'.format(list_roles_response))
```

### Delete Role

hvac.api.secrets_engines.pki.delete_role()

```
import hvac
client = hvac.Client()

delete_role_response = client.secrets.pki.delete_role('role2delete')
```

### Generate Root

hvac.api.secrets_engines.pki.generate_root()

```
import hvac
client = hvac.Client()

generate_root_response = client.secrets.pki.generate_root(
   type='exported',
   common_name='New root CA'
)
print('New root CA: {}'.format(generate_root_response))
```

### Delete Root

hvac.api.secrets_engines.pki.delete_root()

```
import hvac
client = hvac.Client()

delete_root_response = client.secrets.pki.delete_root()
```

### Sign Intermediate

hvac.api.secrets_engines.pki.sign_intermediate()

```
import hvac
client = hvac.Client()

sign_intermediate_response = client.secrets.pki.sign_intermediate(
    csr='....',
    common_name='example.com',
)
print('Signed certificate: {}'.format(sign_intermediate_response))
```

### Sign Self-Issued

hvac.api.secrets_engines.pki.sign_self_issued()

```
import hvac
client = hvac.Client()

sign_self_issued_response = client.secrets.pki.sign_self_issued(
   certificate='...'
)
print('Signed certificate: {}'.format(sign_self_issued_response))
```

### Sign Certificate

hvac.api.secrets_engines.pki.sign_certificate()

```
import hvac
client = hvac.Client()

sign_certificate_response = client.secrets.pki.sign_certificate(
   name='myrole',
   csr='...',
   common_name='example.com'
)
print('Signed certificate: {}'.format(sign_certificate_response))
```

### Sign Verbatim

hvac.api.secrets_engines.pki.sign_verbatim()

```
import hvac
client = hvac.Client()

sign_verbatim_response = client.secrets.pki.sign_verbatim(
   name='myrole',
   csr='...'
)
print('Signed certificate: {}'.format(sign_verbatim_response))
```

### Tidy

hvac.api.secrets_engines.pki.tidy()

```
import hvac
client = hvac.Client()

tidy_response = client.secrets.pki.tidy()
```

### 2.1.7 KV Secrets Engines

The *hvac.api.secrets_engines.Kv* instance under the `Client class's kv attribute` is a wrapper to expose either version 1 (*KvV1*) or version 2 of the key/value secrets engines' API methods (*KvV2*). At present, this class defaults to version 2 when accessing methods on the instance.

#### Setting the Default KV Version

*hvac.api.secrets_engines.KvV1.read_secret()*

```
import hvac
client = hvac.Client()

client.kv.default_kv_version = 1
client.kv.read_secret(path='hvac')  # => calls hvac.api.secrets_engines.KvV1.read_
↪secret
```

#### Explicitly Calling a KV Version Method

*hvac.api.secrets_engines.KvV1.list_secrets()*

```
import hvac
client = hvac.Client()

client.kv.v1.read_secret(path='hvac')
client.kv.v2.read_secret_version(path='hvac')
```

#### Specific KV Version Usage

#### KV - Version 1

---

**Note:** Every method under the *Kv class's v1 attribute* includes a *mount_point* parameter that can be used to address the KvV1 secret engine under a custom mount path. E.g., If enabling the KvV1 secret engine using Vault's CLI commands via *vault secrets enable -path=my-kvv1 -version=1 kv*", the *mount_point* parameter in *hvac.api.secrets_engines.KvV1()* methods would be set to "my-kvv1".

---

#### Read a Secret

*hvac.api.secrets_engines.KvV1.read_secret()*

```
import hvac
client = hvac.Client()

# The following path corresponds, when combined with the mount point, to a full Vault␣
↪API route of "v1/secretz/hvac"
mount_point = 'secretz'
secret_path = 'hvac'

read_secret_result = client.secrets.kv.v1.read_secret(
```

(continues on next page)

(continued from previous page)

```
    path=secret_path,
    mount_point=mount_point,
)
print('The "psst" key under the secret path ("/v1/secret/hvac") is: {psst}'.format(
    psst=read_secret_result['data']['psst'],
))
```

### List Secrets

*hvac.api.secrets_engines.KvV1.list_secrets()*

```
import hvac
client = hvac.Client()

list_secrets_result = client.secrets.kv.v1.list_secrets(path='hvac')

print('The following keys found under the selected path ("/v1/secret/hvac"): {keys}'.
↪format(
    keys=','.join(list_secrets_result['data']['keys']),
))
```

### Create or Update a Secret

*hvac.api.secrets_engines.KvV1.create_or_update_secret()*

```
import hvac
client = hvac.Client()
hvac_secret = {
    'psst': 'this is so secret yall',
}

client.secrets.kv.v1.create_or_update_secret(
    path='hvac',
    secret=hvac_secret,
)

read_secret_result = client.secrets.kv.v1.read_secret(
    path='hvac',
)
print('The "psst" key under the secret path ("/v1/secret/hvac") is: {psst}'.format(
    psst=read_secret_result['data']['psst'],
))
```

### Delete a Secret

*hvac.api.secrets_engines.KvV1.delete_secret()*

```
import hvac
client = hvac.Client()

client.secrets.kv.v1.delete_secret(
    path='hvac',
)

# The following will raise a :py:class:`hvac.exceptions.InvalidPath` exception.
read_secret_result = client.secrets.kv.v1.read_secret(
    path='hvac',
)
```

### KV - Version 2

**Note:** Every method under the *Kv class's v2 attribute* includes a *mount_point* parameter that can be used to address the KvV2 secret engine under a custom mount path. E.g., If enabling the KvV2 secret engine using Vault's CLI commands via *vault secrets enable -path=my-kvv2 -version=2 kv*", the *mount_point* parameter in *hvac.api.secrets_engines.KvV2()* methods would be set to "my-kvv2".

### Configuration

*hvac.api.secrets_engines.KvV2.configure()*

Setting the default *max_versions* for a key/value engine version 2 under a path of *kv*:

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.configure(
    max_versions=20,
    mount_point='kv',
)
```

Setting the default *cas_required* (check-and-set required) flag under the implicit default path of *secret*:

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.configure(
    cas_required=True,
)
```

### Read Configuration

*hvac.api.secrets_engines.KvV2.configure()*

Reading the configuration of a KV version 2 engine mounted under a path of *kv*:

```
import hvac
client = hvac.Client()

kv_configuration = client.secrets.kv.v2.read_configuration(
    mount_point='kv',
)
print('Config under path "kv": max_versions set to "{max_ver}"'.format(
    max_ver=kv_configuration['data']['max_versions'],
))
print('Config under path "kv": check-and-set require flag set to {cas}'.format(
    cas=kv_configuration['data']['cas_required'],
))
```

### Read Secret Versions

*hvac.api.secrets_engines.KvV2.read_secret_version()*

Read the latest version of a given secret/path ("hvac"):

```
import hvac
client = hvac.Client()

secret_version_response = client.secrets.kv.v2.read_secret_version(
    path='hvac',
)
print('Latest version of secret under path "hvac" contains the following keys: {data}
↪'.format(
    data=secret_version_response['data']['data'].keys(),
))
print('Latest version of secret under path "hvac" created at: {date}'.format(
    date=secret_version_response['data']['metadata']['created_time'],
))
print('Latest version of secret under path "hvac" is version #{ver}'.format(
    ver=secret_version_response['data']['metadata']['version'],
))
```

Read specific version (1) of a given secret/path ("hvac"):

```
import hvac
client = hvac.Client()

secret_version_response = client.secrets.kv.v2.read_secret_version(
    path='hvac',
    version=1,
)
print('Version 1 of secret under path "hvac" contains the following keys: {data}'.
↪format(
    data=secret_version_response['data']['data'].keys(),
))
print('Version 1 of secret under path "hvac" created at: {date}'.format(
```

(continues on next page)

(continued from previous page)

```
    date=secret_version_response['data']['metadata']['created_time'],
))
```

### Create/Update Secret

*hvac.api.secrets_engines.KvV2.create_or_update_secret()*

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.create_or_update_secret(
    path='hvac',
    secret=dict(pssst='this is secret'),
)
```

*cas* parameter with an argument that doesn't match the current version:

```
import hvac
client = hvac.Client()

# Assuming a current version of "6" for the path "hvac" =>
client.secrets.kv.v2.create_or_update_secret(
    path='hvac',
    secret=dict(pssst='this is secret'),
    cas=5,
)  # Raises hvac.exceptions.InvalidRequest
```

*cas* parameter set to *0* will only succeed if the path hasn't already been written.

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.create_or_update_secret(
    path='hvac',
    secret=dict(pssst='this is secret #1'),
    cas=0,
)

client.secrets.kv.v2.create_or_update_secret(
    path='hvac',
    secret=dict(pssst='this is secret #2'),
    cas=0,
)  # => Raises hvac.exceptions.InvalidRequest
```

### Patch Existing Secret

Method (similar to the Vault CLI command *vault kv patch*) to update an existing path. Either to add a new key/value to the secret and/or update the value for an existing key. Raises an *hvac.exceptions.InvalidRequest* if the path hasn't been written to previously.

*hvac.api.secrets_engines.KvV2.patch()*

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.patch(
    path='hvac',
    secret=dict(pssst='this is a patched secret'),
)
```

### Delete Latest Version of Secret

*hvac.api.secrets_engines.KvV2.delete_latest_version_of_secret()*

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.delete_latest_version_of_secret(
    path=hvac,
)
```

### Delete Secret Versions

*hvac.api.secrets_engines.KvV2.delete_secret_versions()*

Marking the first 3 versions of a secret deleted under path "hvac":

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.delete_secret_versions(
    path='hvac',
    versions=[1, 2, 3],
)
```

### Undelete Secret Versions

*hvac.api.secrets_engines.KvV2.undelete_secret_versions()*

Marking the last 3 versions of a secret deleted under path "hvac" as "undeleted":

```
import hvac
client = hvac.Client()

hvac_path_metadata = client.secrets.kv.v2.read_secret_metadata(
    path='hvac',
)
```

(continues on next page)

(continued from previous page)

```
oldest_version = hvac_path_metadata['data']['oldest_version']
current_version = hvac_path_metadata['data']['current_version']
versions_to_undelete = range(max(oldest_version, current_version - 2), current_
↪version + 1)

client.secrets.kv.v2.undelete_secret_versions(
    path='hvac',
    versions=versions_to_undelete,
)
```

### Destroy Secret Versions

*hvac.api.secrets_engines.KvV2.destroy_secret_versions()*

Destroying the first three versions of a secret under path 'hvac':

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.destroy_secret_versions(
    path='hvac',
    versions=[1, 2, 3],
)
```

### List Secrets

*hvac.api.secrets_engines.KvV2.list_secrets()*

Listing secrets under the 'hvac' path prefix:

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.create_or_update_secret(
    path='hvac/big-ole-secret',
    secret=dict(pssst='this is a large secret'),
)

client.secrets.kv.v2.create_or_update_secret(
    path='hvac/lil-secret',
    secret=dict(pssst='this secret... not so big'),
)

list_response = client.secrets.kv.v2.list_secrets(
    path='hvac',
)

print('The following paths are available under "hvac" prefix: {keys}'.format(
    keys=','.join(list_response['data']['keys']),
))
```

### Read Secret Metadata

*hvac.api.secrets_engines.KvV2.read_secret_metadata()*

```
import hvac
client = hvac.Client()

hvac_path_metadata = client.secrets.kv.v2.read_secret_metadata(
    path='hvac',
)

print('Secret under path hvac is on version {cur_ver}, with an oldest version of {old_
↪ver}'.format(
    cur_ver=hvac_path_metadata['data']['oldest_version'],
    old_ver=hvac_path_metadata['data']['current_version'],
))
```

### Update Metadata

*hvac.api.secrets_engines.KvV2.update_metadata()*

Set max versions for a given path ("hvac") to 3:

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.update_metadata(
    path='hvac',
    max_versions=3,
)
```

Set cas (check-and-set) parameter as required for a given path ("hvac"):

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.update_metadata(
    path='hvac',
    cas_required=True,
)
```

Set "delete_version_after" value to 30 minutes for all new versions written to the "hvac" path / key:

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.update_metadata(
    path='hvac',
    delete_version_after="30m",
)
```

#### Delete Metadata and All Versions

*hvac.api.secrets_engines.KvV2.delete_metadata_and_all_versions()*

Delete all versions and metadata for a given path:

```
import hvac
client = hvac.Client()

client.secrets.kv.v2.delete_metadata_and_all_versions(
    path='hvac',
)
```

### 2.1.8 Transform

- *Encode/Decode Example*
- *Create/Update Role*
- *Read Role*
- *List Roles*
- *Delete Role*
- *Create/Update Transformation*
- *Read Transformation*
- *List Transformations*
- *Delete Transformation*
- *Create/Update Template*
- *Read Template*
- *List Templates*
- *Delete Template*
- *Create/Update Alphabet*
- *Read Alphabet*
- *List Alphabets*
- *Delete Alphabet*
- *Create Or Update FPE Transformation*
- *Create Or Update Masking Transformation*
- *Create Or Update Tokenization Transformation*
- *Create Or Update Tokenization Store*
- *Encode*
- *Validate Token*
- *Check Tokenization*

- *Retrieve Token Metadata*
- *Snapshot Tokenization State*
- *Restore Tokenization State*
- *Export Decoded Tokenization State*
- *Rotate Tokenization Key*
- *Update Tokenization Key Config*
- *List Tokenization Key Configuration*
- *Read Tokenization Key Configuration*
- *Trim Tokenization Key Version*

### Encode/Decode Example

*hvac.api.secrets_engines.Transform.encode()* *hvac.api.secrets_engines.Transform.decode()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

input_value = '1111-1111-1111-1111'

role_name = 'hvac-role'
transformation_name = 'hvac-fpe-credit-card'
transformations = [transformation_name]

# Create a role and a transformation
client.secrets.transform.create_or_update_role(
    name=role_name,
    transformations=transformations,
)
client.secrets.transform.create_or_update_transformation(
    name=transformation_name,
    transform_type='fpe',
    template='builtin/creditcardnumber',
    tweak_source='internal',
    allowed_roles=[role_name],
)

# Use the role/transformation combination to encode a value
encode_response = client.secrets.transform.encode(
    role_name=role_name,
    value=input_value,
    transformation=transformation_name,
)
print('The encoded value is: %s' % encode_response['data']['encoded_value'])

# Use the role/transformation combination to decode a value
decode_response = client.secrets.transform.decode(
    role_name=role_name,
    value=encode_response['data']['encoded_value'],
    transformation=transformation_name,
```

(continues on next page)

(continued from previous page)

```
)
print('The decoded value is: %s' % decode_response['data']['decoded_value'])
```

```
The encoded value is: ...
The decoded value is: 1111-1111-1111-1111
```

### Create/Update Role

*hvac.api.secrets_engines.Transform.create_or_update_role()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.secrets.transform.create_or_update_role(
    name='hvac-role',
    transformations=[
        'hvac-fpe-credit-card',
    ],
)
```

### Read Role

*hvac.api.secrets_engines.Transform.read_role()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

role_name = 'hvac-role'
client.secrets.transform.create_or_update_role(
    name=role_name,
    transformations=[
        'hvac-fpe-credit-card',
    ],
)
read_response = client.secrets.transform.read_role(
    name=role_name,
)
print('Role "{}" has the following transformations configured: {}'.format(
    role_name,
    ', '.join(read_response['data']['transformations']),
))
```

```
Role "hvac-role" has the following transformations configured: hvac-fpe-credit-card
```

### List Roles

*hvac.api.secrets_engines.Transform.list_roles()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.secrets.transform.create_or_update_role(
    name='hvac-role',
    transformations=[
        'hvac-fpe-credit-card',
    ],
)
list_response = client.secrets.transform.list_roles()
print('List of transform role names: {}'.format(
    ', '.join(list_response['data']['keys']),
))
```

```
List of transform role names: hvac-role
```

### Delete Role

*hvac.api.secrets_engines.Transform.delete_role()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

role_name = 'hvac-role'

# Create a role
client.secrets.transform.create_or_update_role(
    name=role_name,
    transformations=[
        'hvac-fpe-credit-card',
    ],
)

# Subsequently delete it...
client.secrets.transform.delete_role(
    name=role_name,
)
```

### Create/Update Transformation

*hvac.api.secrets_engines.Transform.create_or_update_transformation()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

transformation_name = 'hvac-fpe-credit-card'
template = 'builtin/creditcardnumber'
client.secrets.transform.create_or_update_transformation(
    name=transformation_name,
    transform_type='fpe',
```

(continues on next page)

(continued from previous page)

```
    template=template,
    tweak_source='internal',
    allowed_roles=[
        'test-role'
    ],
)
```

### Read Transformation

*hvac.api.secrets_engines.Transform.read_transformation()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

transformation_name = 'hvac-fpe-credit-card'
template = 'builtin/creditcardnumber'
client.secrets.transform.create_or_update_transformation(
    name=transformation_name,
    transform_type='fpe',
    template=template,
    tweak_source='internal',
    allowed_roles=[
        'hvac-role'
    ],
)
read_response = client.secrets.transform.read_transformation(
    name=transformation_name,
)
print('Transformation "{}" has the following type configured: {}'.format(
    transformation_name,
    read_response['data']['type'],
))
```

```
Transformation "hvac-fpe-credit-card" has the following type configured: fpe
```

### List Transformations

*hvac.api.secrets_engines.Transform.list_transformations()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

transformation_name = 'hvac-fpe-credit-card'
template = 'builtin/creditcardnumber'
client.secrets.transform.create_or_update_transformation(
    name=transformation_name,
    transform_type='fpe',
    template=template,
    tweak_source='internal',
    allowed_roles=[
        'hvac-role'
    ],
)
```

(continues on next page)

(continued from previous page)

```
list_response = client.secrets.transform.list_transformations()
print('List of transformations: {}'.format(
    ', '.join(list_response['data']['keys']),
))
```

```
List of transformations: hvac-fpe-credit-card
```

### Delete Transformation

*hvac.api.secrets_engines.Transform.delete_transformation()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

transformation_name = 'hvac-fpe-credit-card'
template = 'builtin/creditcardnumber'

# Create a transformation
client.secrets.transform.create_or_update_transformation(
    name=transformation_name,
    transform_type='fpe',
    template=template,
    tweak_source='internal',
    allowed_roles=[
        'hvac-role'
    ],
)

# Subsequently delete it...
client.secrets.transform.delete_role(
    name=role_name,
)
```

### Create/Update Template

*hvac.api.secrets_engines.Transform.create_or_update_template()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

template_name = 'hvac-template'
create_response = client.secrets.transform.create_or_update_template(
    name=template_name,
    template_type='regex',
    pattern='(\\d{9})',
    alphabet='builtin/numeric',
)
```

### Read Template

*hvac.api.secrets_engines.Transform.read_template()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

template_name = 'hvac-template'
client.secrets.transform.create_or_update_template(
    name=template_name,
    template_type='regex',
    pattern='(\\d{9})',
    alphabet='builtin/numeric',
)
read_response = client.secrets.transform.read_template(
    name=template_name,
)
print('Template "{}" has the following type configured: {}'.format(
    template_name,
    read_response['data']['type'],
))
```

```
Template "hvac-template" has the following type configured: regex
```

### List Templates

*hvac.api.secrets_engines.Transform.list_templates()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

template_name = 'hvac-template'
client.secrets.transform.create_or_update_template(
    name=template_name,
    template_type='regex',
    pattern='(\\d{9})',
    alphabet='builtin/numeric',
)
list_response = client.secrets.transform.list_templates()
print('List of templates: {}'.format(
    ', '.join(list_response['data']['keys']),
))
```

```
List of templates: builtin/creditcardnumber, builtin/socialsecuritynumber, hvac-
↪template
```

### Delete Template

*hvac.api.secrets_engines.Transform.delete_template()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

template_name = 'hvac-template'
client.secrets.transform.create_or_update_template(
    name=template_name,
    template_type='regex',
    pattern='(\\d{9})',
    alphabet='builtin/numeric',
)

# Subsequently delete it...
client.secrets.transform.delete_template(
    name=template_name,
)
```

### Create/Update Alphabet

*hvac.api.secrets_engines.Transform.create_or_update_alphabet()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

alphabet_name = 'hvac-alphabet'
alphabet_value = 'abc'
client.secrets.transform.create_or_update_alphabet(
    name=alphabet_name,
    alphabet=alphabet_value,
)
```

### Read Alphabet

*hvac.api.secrets_engines.Transform.read_alphabet()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

alphabet_name = 'hvac-alphabet'
alphabet_value = 'abc'
client.secrets.transform.create_or_update_alphabet(
    name=alphabet_name,
    alphabet=alphabet_value,
)
read_response = client.secrets.transform.read_alphabet(
    name=alphabet_name,
)
print('Alphabet "{}" has this jazz: {}'.format(
    alphabet_name,
    read_response['data']['alphabet'],
))
```

```
Alphabet "hvac-alphabet" has this jazz: abc
```

### List Alphabets

*hvac.api.secrets_engines.Transform.list_alphabets()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

alphabet_name = 'hvac-alphabet'
alphabet_value = 'abc'
client.secrets.transform.create_or_update_alphabet(
    name=alphabet_name,
    alphabet=alphabet_value,
)
list_response = client.secrets.transform.list_alphabets()
print('List of alphabets: {}'.format(
    ', '.join(list_response['data']['keys']),
))
```

```
List of alphabets: builtin/alphalower, ..., hvac-alphabet
```

### Delete Alphabet

*hvac.api.secrets_engines.Transform.delete_alphabet()*

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

alphabet_name = 'hvac-alphabet'
alphabet_value = 'abc'

# Create an alphabet
client.secrets.transform.create_or_update_alphabet(
    name=alphabet_name,
    alphabet=alphabet_value,
)

# Subsequently delete it...
client.secrets.transform.delete_alphabet(
    name=alphabet_name,
)
```

### Create Or Update FPE Transformation

*hvac.api.secrets_engines.Transform.create_or_update_fpe_transformation()*

Creates or update an FPE transformation with the given name.

If a transformation with the name does not exist, it will be created. If the transformation exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/transformations/fpe/:name.

**param name** The name of the transformation to create or update. This is part of the request URL.

**type name** str

**param template** The template name to use for matching value on encode and decode operations when using this transformation.

**type template** str

**param tweak_source** Specifies the source of where the tweak value comes from. Valid sources are: supplied, generated, and internal.

**type tweak_source** str

**param allowed_roles** A list of allowed roles that this transformation can be assigned to. A role using this transformation must exist in this list in order for encode and decode operations to properly function.

**type allowed_roles** list

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the create_or_update_fpe_transformation request.

**rtype** requests.Response

### Create Or Update Masking Transformation

*hvac.api.secrets_engines.Transform.create_or_update_masking_transformation()*

Creates or update a masking transformation with the given name. If a transformation with the name does not exist, it will be created. If the transformation exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/transformations/masking/:name.

**param name** The name of the transformation to create or update. This is part of the request URL.

**type name** str

**param template** The template name to use for matching value on encode and decode operations when using this transformation.

**type template** str

**param masking_character** The character to use for masking. If multiple characters are provided, only the first one is used and the rest is ignored. Only used when the type is masking.

**type masking_character** str

**param allowed_roles** A list of allowed roles that this transformation can be assigned to. A role using this transformation must exist in this list in order for encode and decode operations to properly function.

**type allowed_roles** list

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the create_or_update_masking_transformation request.

**rtype** requests.Response

### Create Or Update Tokenization Transformation

*hvac.api.secrets_engines.Transform.create_or_update_tokenization_transformation()*

This endpoint creates or updates a tokenization transformation with the given name. If a transformation with the name does not exist, it will be created. If the transformation exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/:name.

**param max_ttl** The maximum TTL of a token. If 0 or unspecified, tokens may have no expiration.

**type max_ttl** str

**param mapping_mode** Specifies the mapping mode for stored tokenization values.

- *default* is strongly recommended for highest security
- *exportable* exportable allows for all plaintexts to be decoded via the export-decoded endpoint in an emergency.

**type mapping_mode** str

**param allowed_roles** aAlist of allowed roles that this transformation can be assigned to. A role using this transformation must exist in this list in order for encode and decode operations to properly function.

**type allowed_roles** list

**param stores** list of tokenization stores to use for tokenization state. Vault's internal storage is used by default.

**type stores** list

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the create_or_update_tokenization_transformation request.

**rtype** requests.Response

### Create Or Update Tokenization Store

*hvac.api.secrets_engines.Transform.create_or_update_tokenization_store()*

Create or update a storage configuration for use with tokenization. The database user configured here should only have permission to SELECT, INSERT, and UPDATE rows in the tables.

**Supported methods:** POST: /{mount_point}/store/:name.

**param name** The name of the store to create or update.

**type name** str

**param type** Specifies the type of store. Currently only *sql* is supported.

**type type** str

**param driver** Specifies the database driver to use, and thus which SQL database type. Currently the supported options are *postgres* or *mysql*

**type driver** str

**param supported_transformations** The types of transformations this store can host. Currently only *tokenization* is supported.

**type supported_transformations** list(str)

**param connection_string** database connection string with template slots for username and password that Vault will use for locating and connecting to a database. Each database driver type has a different syntax for its connection strings.

**type connection_string** str

**param username** username value to use when connecting to the database.

**type username** str

**param password** password value to use when connecting to the database.

**type password** str

**param schema** schema within the database to expect tokenization state tables.

**type schema** str

**param max_open_connections** maximum number of connections to the database at any given time.

**type max_open_connections** int

**param max_idle_connections** maximum number of idle connections to the database at any given time.

**type max_idle_connections** int

**param max_connection_lifetime** means no limit.

**type max_connection_lifetime** duration

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the create_or_update_tokenization_store request.

**rtype** requests.Response

### Encode

*hvac.api.secrets_engines.Transform.encode()*

Encode the provided value using a named role.

**Supported methods:** POST: /{mount_point}/encode/:role_name.

**param role_name** the role name to use for this operation. This is specified as part of the URL.

**type role_name** str | unicode

**param value** the value to be encoded.

**type value** str | unicode

**param transformation** the transformation within the role that should be used for this encode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.

**type transformation** str | unicode

**param tweak** the tweak source.

**type tweak** str | unicode

**param batch_input** a list of items to be encoded in a single batch. When this parameter is set, the 'value', 'transformation' and 'tweak' parameters are ignored. Instead, the aforementioned parameters should be provided within each object in the list.

**type batch_input** list

**param mount_point** The "path" the secrets engine was mounted on.

**type mount_point** str | unicode

**return** The response of the encode request.

**rtype** requests.Response

### Validate Token

*hvac.api.secrets_engines.Transform.validate_token()*

Determine if a provided tokenized value is valid and unexpired. Only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/validate/:role_name.

**param role_name** the role name to use for this operation. This is specified as part of the URL.

**type role_name** str

**param value** the token for which to check validity.

**type value** str

**param transformation** the transformation within the role that should be used for this decode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.

**type transformation** str

**param batch_input** a list of items to be decoded in a single batch. When this parameter is set, the 'value' parameter is ignored. Instead, the aforementioned parameters should be provided within each object in the list.

**type batch_input** list

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the validate_token request.

**rtype** requests.Response

### Check Tokenization

*hvac.api.secrets_engines.Transform.check_tokenization()*

Determine if a provided plaintext value has an valid, unexpired tokenized value. Note that this cannot return the token, just confirm that a tokenized value exists. This endpoint is only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/tokenized/:role_name.

**param role_name** the role name to use for this operation. This is specified as part of the URL.

**type role_name** str

**param value** the token to test for whether it has a valid tokenization.

**type value** str

**param transformation** the transformation within the role that should be used for this decode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.

**type transformation** str

**param batch_input** a list of items to be decoded in a single batch. When this parameter is set, the 'value' parameter is ignored. Instead, the aforementioned parameters should be provided within each object in the list.

**type batch_input** list

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the check_tokenization request.

**rtype** requests.Response

### Retrieve Token Metadata

*hvac.api.secrets_engines.Transform.retrieve_token_metadata()*

This endpoint retrieves metadata for a tokenized value using a named role. Only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/metadata/:role_name.

**param role_name** the role name to use for this operation. This is specified as part of the URL.

**type role_name** str

**param value** the token for which to retrieve metadata.

**type value** str

**param transformation** the transformation within the role that should be used for this decode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.

**type transformation** str

**param batch_input** a list of items to be decoded in a single batch. When this parameter is set, the 'value' parameter is ignored. Instead, the aforementioned parameters should be provided within each object in the list.

**type batch_input** list

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the retrieve_token_metadata request.

**rtype** requests.Response

### Snapshot Tokenization State

*hvac.api.secrets_engines.Transform.snapshot_tokenization_state()*

This endpoint starts or continues retrieving a snapshot of the stored state of a tokenization transform. This state is protected as it is in the underlying store, and so is safe for storage or transport. Snapshots may be used for backup purposes or to migrate from one store to another. If more than one store is configured for a tokenization transform, the snapshot data contains the contents of the first store.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/snapshot/:name.

**param name** the name of the transformation to snapshot.

**type name** str

**param limit** maximum number of tokenized value states to return on this call.

**type limit** int

**param continuation** absent or empty, a new snapshot is started. If present, the snapshot should continue at the next available value.

**type continuation** str

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the snapshot_tokenization_state request.

**rtype** requests.Response

### Restore Tokenization State

*hvac.api.secrets_engines.Transform.restore_tokenization_state()*

This endpoint restores previously snapshotted tokenization state values to the underlying store(s) of a tokenization transform. Calls to this endpoint are idempotent, so multiple outputs from a snapshot run can be applied via restore in any order and duplicates will not cause a problem.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/restore/:name.

**param name** the name of the transformation to restore.

**type name** str

**param values** number of tokenization state values from a previous snapshot call.

**type values** str

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the restore_tokenization_state request.

**rtype** requests.Response

### Export Decoded Tokenization State

*hvac.api.secrets_engines.Transform.export_decoded_tokenization_state()*

Start or continue retrieving an export of tokenization state, including the tokens and their decoded values. This call is only supported on tokenization stores configured with the exportable mapping mode. Refer to the Tokenization documentation for when to use the exportable mapping mode. Decoded values are in Base64 representation.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/export-decoded/:name.

**param name** the name of the transformation to export.

**type name** str

**param limit** maximum number of tokenized value states to return on this call.

**type limit** int

**param continuation** absent or empty, a new export is started. If present, the export should continue at the next available value.

**type continuation** str

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the export_decoded_tokenization_state request.

**rtype** requests.Response

### Rotate Tokenization Key

*hvac.api.secrets_engines.Transform.rotate_tokenization_key()*

Rotate the version of the named key. After rotation, new requests will be encoded with the new version of the key.

**Supported methods:** POST: /{mount_point}/tokenization/keys/{transform_name}/rotate.

**param transform_name** the transform name to use for this operation. This is specified as part of the URL.

**type transform_name** str

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the rotate_tokenization_key request.

**rtype** requests.Response

### Update Tokenization Key Config

*hvac.api.secrets_engines.Transform.update_tokenization_key_config()*

Allow the minimum key version to be set for decode operations. Only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/tokenization/keys/{transform_name}/config.

**param transform_name** the transform name to use for this operation. This is specified as part of the URL.

**type transform_name** str

**param min_decryption_version** the minimum key version that vault can use to decode values for the corresponding transform.

**type min_decryption_version** int

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the update_tokenization_key_config request.

**rtype** requests.Response

### List Tokenization Key Configuration

*hvac.api.secrets_engines.Transform.list_tokenization_key_configuration()*

List all tokenization keys. Only valid for tokenization transformations.

**Supported methods:** LIST: /{mount_point}/tokenization/keys/.

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the list_tokenization_key_configuration request.

**rtype** requests.Response

### Read Tokenization Key Configuration

*hvac.api.secrets_engines.Transform.read_tokenization_key_configuration()*

Read tokenization key configuration for a particular transform. Only valid for tokenization transformations.

**Supported methods:** GET: /{mount_point}/tokenization/keys/:{mount_point}_name.

**param transform_name** the transform name to use for this operation. This is specified as part of the URL.

**type transform_name** str

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the read_tokenization_key_configuration request.

**rtype** requests.Response

### Trim Tokenization Key Version

*hvac.api.secrets_engines.Transform.trim_tokenization_key_version()*

Trim older key versions setting a minimum version for the keyring. Once trimmed, previous versions of the key cannot be recovered.

**Supported methods:** POST: /{mount_point}/tokenization/keys/{transform_name}/trim.

**param transform_name** the transform name to use for this operation. This is specified as part of the URL.

**type transform_name** str

**param min_available_version**

**type min_available_version** int

**param mount_point** The "path" the method/backend was mounted on.

**type mount_point** str

**return** The response of the trim_tokenization_key_version request.

**rtype** requests.Response

### 2.1.9 Transit



**Note:** The following helper method is used various of the examples included here.

```
import sys


def base64ify(bytes_or_str):
    """Helper method to perform base64 encoding across Python 2.7 and Python 3.X"""
    if sys.version_info[0] >= 3 and isinstance(bytes_or_str, str):
        input_bytes = bytes_or_str.encode('utf8')
```

(continues on next page)

(continued from previous page)

```
    else:
        input_bytes = bytes_or_str

    output_bytes = base64.urlsafe_b64encode(input_bytes)
    if sys.version_info[0] >= 3:
        return output_bytes.decode('ascii')
    else:
        return output_bytes
```

### Create Key

Transit.**create_key**(*name*, *convergent_encryption=None*, *derived=None*, *exportable=None*, *allow_plaintext_backup=None*, *key_type=None*, *mount_point='transit'*)

Create a new named encryption key of the specified type.

The values set here cannot be changed after key creation.

**Supported methods:** POST: /{mount_point}/keys/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to create. This is specified as part of the URL.
- **convergent_encryption** (*bool*) – If enabled, the key will support convergent encryption, where the same plaintext creates the same ciphertext. This requires derived to be set to true. When enabled, each encryption(/decryption/rewrap/datakey) operation will derive a nonce value rather than randomly generate it.
- **derived** (*bool*) – Specifies if key derivation is to be used. If enabled, all encrypt/decrypt requests to this named key must provide a context which is used for key derivation.
- **exportable** (*bool*) – Enables keys to be exportable. This allows for all the valid keys in the key ring to be exported. Once set, this cannot be disabled.
- **allow_plaintext_backup** (*bool*) – If set, enables taking backup of named key in the plaintext format. Once set, this cannot be disabled.
- **key_type** (*str | unicode*) – Specifies the type of key to create. The currently-supported types are:
  - **aes256-gcm96**: AES-256 wrapped with GCM using a 96-bit nonce size AEAD
  - **chacha20-poly1305**: ChaCha20-Poly1305 AEAD (symmetric, supports derivation and convergent encryption)
  - **ed25519**: ED25519 (asymmetric, supports derivation).
  - **ecdsa-p256**: ECDSA using the P-256 elliptic curve (asymmetric)
  - **ecdsa-p384**: ECDSA using the P-384 elliptic curve (asymmetric)
  - **ecdsa-p521**: ECDSA using the P-521 elliptic curve (asymmetric)
  - **rsa-2048**: RSA with bit size of 2048 (asymmetric)
  - **rsa-3072**: RSA with bit size of 3072 (asymmetric)
  - **rsa-4096**: RSA with bit size of 4096 (asymmetric)
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.secrets.transit.create_key(name='hvac-key')
```

### Read Key

Transit.**read_key**(*name*, *mount_point='transit'*)

Read information about a named encryption key.

The keys object shows the creation time of each key version; the values are not the keys themselves. Depending on the type of key, different information may be returned, e.g. an asymmetric key will return its public key in a standard format for the type.

**Supported methods:** GET: /{mount_point}/keys/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to read. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_key request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

read_key_response = client.secrets.transit.read_key(name='hvac-key')
latest_version = read_key_response['data']['latest_version']
print('Latest version for key "hvac-key" is: {ver}'.format(ver=latest_version))
```

Example output:

```
Latest version for key "hvac-key" is: 1
```

### List Keys

Transit.**list_keys**(*mount_point='transit'*)

List keys (if there are any).

Only the key names are returned (not the actual keys themselves).

An exception is thrown if there are no keys.

**Supported methods:** LIST: /{mount_point}/keys. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

list_keys_response = client.secrets.transit.read_key(name='hvac-key')
keys = list_keys_response['data']['keys']
print('Currently configured keys: {keys}'.format(keys=keys))
```

Example output:

```
Currently configured keys: {'1': ...}
```

### Delete Key

Transit.**delete_key**(*name*, *mount_point='transit'*)

Delete a named encryption key.

It will no longer be possible to decrypt any data encrypted with the named key. Because this is a potentially catastrophic operation, the deletion_allowed tunable must be set in the key's /config endpoint.

**Supported methods:** DELETE: /{mount_point}/keys/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to delete. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

key_name = 'gonna-delete-this-key'

client.secrets.transit.create_key(
    name=key_name,
)

# Update key subsequently to allow deletion...
client.secrets.transit.update_key_configuration(
    name=key_name,
    deletion_allowed=True,
)

# Finally, delete the key
client.secrets.transit.delete_key(name=key_name)
```

### Update Key Configuration

Transit.**update_key_configuration**(*name*, *min_decryption_version=None*, *min_encryption_version=None*, *deletion_allowed=None*, *exportable=None*, *allow_plaintext_backup=None*, *mount_point='transit'*)

Tune configuration values for a given key.

These values are returned during a read operation on the named key.

**Supported methods:** POST: /{mount_point}/keys/{name}/config. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to update configuration for.
- **min_decryption_version** (*int*) – Specifies the minimum version of ciphertext allowed to be decrypted. Adjusting this as part of a key rotation policy can prevent old copies of ciphertext from being decrypted, should they fall into the wrong hands. For signatures, this value controls the minimum version of signature that can be verified against. For HMACs, this controls the minimum version of a key allowed to be used as the key for verification.
- **min_encryption_version** (*int*) – Specifies the minimum version of the key that can be used to encrypt plaintext, sign payloads, or generate HMACs. Must be 0 (which will use the latest version) or a value greater or equal to min_decryption_version.
- **deletion_allowed** (*bool*) – Specifies if the key is allowed to be deleted.
- **exportable** (*bool*) – Enables keys to be exportable. This allows for all the valid keys in the key ring to be exported. Once set, this cannot be disabled.
- **allow_plaintext_backup** (*bool*) – If set, enables taking backup of named key in the plaintext format. Once set, this cannot be disabled.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

# allow key "hvac-key" to be exported in subsequent requests
client.secrets.transit.update_key_configuration(
    name='hvac-key',
    exportable=True,
)
```

### Rotate Key

Transit.**rotate_key**(*name*, *mount_point='transit'*)

Rotate the version of the named key.

After rotation, new plaintext requests will be encrypted with the new version of the key. To upgrade ciphertext to be encrypted with the latest version of the key, use the rewrap endpoint. This is only supported with keys that support encryption and decryption operations.

**Supported methods:** POST: /{mount_point}/keys/{name}/rotate. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to read information about. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')
client.secrets.transit.rotate_key(name='hvac-key')
```

### Export Key

Transit.**export_key**(*name*, *key_type*, *version=None*, *mount_point='transit'*)

Return the named key.

The keys object shows the value of the key for each version. If version is specified, the specific version will be returned. If latest is provided as the version, the current key will be provided. Depending on the type of key, different information may be returned. The key must be exportable to support this operation and the version must still be valid.

**Supported methods:** GET: /{mount_point}/export/{key_type}/{name}(/{version}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to read information about. This is specified as part of the URL.
- **key_type** (*str | unicode*) – Specifies the type of the key to export. This is specified as part of the URL. Valid values are: encryption-key signing-key hmac-key
- **version** (*str | unicode*) – Specifies the version of the key to read. If omitted, all versions of the key will be returned. If the version is set to latest, the current key will be returned.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')
export_key_response = client.secrets.transit.export_key(
    name='hvac-key',
    key_type='hmac-key',
)

print('Exported keys: %s' % export_key_response['data']['keys'])
```

Example output:

```
Exported keys: {...}
```

### Encrypt Data

Transit.**encrypt_data**(*name*, *plaintext*, *context=None*, *key_version=None*, *nonce=None*, *batch_input=None*, *type=None*, *convergent_encryption=None*, *mount_point='transit'*)

Encrypt the provided plaintext using the named key.

This path supports the create and update policy capabilities as follows: if the user has the create capability for this endpoint in their policies, and the key does not exist, it will be upserted with default values (whether the key requires derivation depends on whether the context parameter is empty or not). If the user only has update capability and the key does not exist, an error will be returned.

**Supported methods:** POST: /{mount_point}/encrypt/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to encrypt against. This is specified as part of the URL.
- **plaintext** (*str | unicode*) – Specifies base64 encoded plaintext to be encoded.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled for this key.

- **key_version** (*int*) – Specifies the version of the key to use for encryption. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **nonce** (*str | unicode*) – Specifies the base64 encoded nonce value. This must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+. The value must be exactly 96 bits (12 bytes) long and the user must ensure that for any given context (and thus, any given encryption key) this nonce value is never reused.
- **batch_input** (*List[dict]*) – Specifies a list of items to be encrypted in a single batch. When this parameter is set, if the parameters 'plaintext', 'context' and 'nonce' are also set, they will be ignored. The format for the input is: [dict(context="b64_context", plaintext="b64_plaintext"), ...]
- **type** (*str | unicode*) – This parameter is required when encryption key is expected to be created. When performing an upsert operation, the type of key to create.
- **convergent_encryption** (*str | unicode*) – This parameter will only be used when a key is expected to be created. Whether to support convergent encryption. This is only supported when using a key with key derivation enabled and will require all requests to carry both a context and 96-bit (12-byte) nonce. The given nonce will be used in place of a randomly generated nonce. As a result, when the same context and nonce are supplied, the same ciphertext is generated. It is very important when using this mode that you ensure that all nonces are unique for a given context. Failing to do so will severely impact the ciphertext's security.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import base64
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

encrypt_data_response = client.secrets.transit.encrypt_data(
    name='hvac-key',
    plaintext=base64ify('hi its me hvac'.encode()),
)
ciphertext = encrypt_data_response['data']['ciphertext']
print('Encrypted plaintext ciphertext is: {cipher}'.format(cipher=ciphertext))
```

Example output:

```
Encrypted plaintext ciphertext is: vault:...
```

### Decrypt Data

Transit.**decrypt_data**(*name*, *ciphertext*, *context=None*, *nonce=None*, *batch_input=None*, *mount_point='transit'*)

Decrypt the provided ciphertext using the named key.

**Supported methods:** POST: /{mount_point}/decrypt/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to decrypt against. This is specified as part of the URL.
- **ciphertext** (*str | unicode*) – the ciphertext to decrypt.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled.
- **nonce** (*str | unicode*) – Specifies a base64 encoded nonce value used during encryption. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+.
- **batch_input** (*List[dict]*) – Specifies a list of items to be decrypted in a single batch. When this parameter is set, if the parameters 'ciphertext', 'context' and 'nonce' are also set, they will be ignored. Format for the input goes like this: [dict(context="b64_context", ciphertext="b64_plaintext"), ...]
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

decrypt_data_response = client.secrets.transit.decrypt_data(
    name='hvac-key',
    ciphertext=ciphertext,
)
plaintext = decrypt_data_response['data']['plaintext']
print('Decrypted plaintext is: {text}'.format(text=plaintext))
```

Example output:

```
Decrypted plaintext is: ...
```

### Rewrap Data

Transit.**rewrap_data**(*name*, *ciphertext*, *context=None*, *key_version=None*, *nonce=None*, *batch_input=None*, *mount_point='transit'*)

Rewrap the provided ciphertext using the latest version of the named key.

Because this never returns plaintext, it is possible to delegate this functionality to untrusted users or scripts.

**Supported methods:** POST: /{mount_point}/rewrap/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to re-encrypt against. This is specified as part of the URL.
- **ciphertext** (*str | unicode*) – Specifies the ciphertext to re-encrypt.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled.
- **key_version** (*int*) – Specifies the version of the key to use for the operation. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **nonce** (*str | unicode*) – Specifies a base64 encoded nonce value used during encryption. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+.
- **batch_input** (*List[dict]*) – Specifies a list of items to be decrypted in a single batch. When this parameter is set, if the parameters 'ciphertext', 'context' and 'nonce' are also set, they will be ignored. Format for the input goes like this: [dict(context="b64_context", ciphertext="b64_plaintext"), ...]
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

encrypt_data_response = client.secrets.transit.rewrap_data(
    name='hvac-key',
    ciphertext=ciphertext,
)
rewrapped_ciphertext = encrypt_data_response['data']['ciphertext']
print('Rewrapped ciphertext is: {cipher}'.format(cipher=rewrapped_ciphertext))
```

Example output:

```
Rewrapped ciphertext is: vault:...
```

### Generate Data Key

Transit.**generate_data_key**(*name*, *key_type*, *context=None*, *nonce=None*, *bits=None*, *mount_point='transit'*)

Generates a new high-entropy key and the value encrypted with the named key.

Optionally return the plaintext of the key as well. Whether plaintext is returned depends on the path; as a result, you can use Vault ACL policies to control whether a user is allowed to retrieve the plaintext value of a key. This is useful if you want an untrusted user or operation to generate keys that are then made available to trusted users.

**Supported methods:** POST: /{mount_point}/datakey/{key_type}/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to use to encrypt the datakey. This is specified as part of the URL.
- **key_type** (*str | unicode*) – Specifies the type of key to generate. If plaintext, the plaintext key will be returned along with the ciphertext. If wrapped, only the ciphertext value will be returned. This is specified as part of the URL.
- **context** (*str | unicode*) – Specifies the key derivation context, provided as a base64-encoded string. This must be provided if derivation is enabled.
- **nonce** (*str | unicode*) – Specifies a nonce value, provided as base64 encoded. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+. The value must be exactly 96 bits (12 bytes) long and the user must ensure that for any given context (and thus, any given encryption key) this nonce value is never reused.
- **bits** (*int*) – Specifies the number of bits in the desired key. Can be 128, 256, or 512.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')
gen_key_response = client.secrets.transit.generate_data_key(
    name='hvac-key',
    key_type='plaintext',
)
ciphertext = gen_key_response['data']['ciphertext']
print('Generated data key ciphertext is: {cipher}'.format(cipher=ciphertext))
```

Example output:

```
Generated data key ciphertext is: vault:...
```

### Generate Random Bytes

Transit.**generate_random_bytes**(*n_bytes=None*, *output_format=None*, *mount_point='transit'*)
Return high-quality random bytes of the specified length.

**Supported methods:** POST: /{mount_point}/random(/{bytes}). Produces: 200 application/json

**Parameters**

- **n_bytes** (*int*) – Specifies the number of bytes to return. This value can be specified either in the request body, or as a part of the URL.
- **output_format** (*str | unicode*) – Specifies the output encoding. Valid options are hex or base64.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

gen_bytes_response = client.secrets.transit.generate_random_bytes(n_bytes=32)
random_bytes = gen_bytes_response['data']['random_bytes']
print('Here are some random bytes: {bytes}'.format(bytes=random_bytes))
```

Example output:

```
Here are some random bytes: ...
```

### Hash Data

Transit.**hash_data**(*hash_input*, *algorithm=None*, *output_format=None*, *mount_point='transit'*)
Return the cryptographic hash of given data using the specified algorithm.

**Supported methods:** POST: /{mount_point}/hash(/{algorithm}). Produces: 200 application/json

**Parameters**

- **hash_input** (*str | unicode*) – Specifies the base64 encoded input data.
- **algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **output_format** (*str | unicode*) – Specifies the output encoding. This can be either hex or base64.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

hash_data_response = client.secrets.transit.hash_data(
    hash_input=base64ify('hi its me hvac'),
    algorithm='sha2-256',
)
sum = hash_data_response['data']['sum']
print('Hashed data is: {sum}'.format(sum=sum))
```

Example output:

```
Hashed data is: ...
```

### Generate Hmac

Transit.**generate_hmac**(*name*, *hash_input*, *key_version=None*, *algorithm=None*, *mount_point='transit'*)

Return the digest of given data using the specified hash algorithm and the named key.

The key can be of any type supported by transit; the raw key will be marshaled into bytes to be used for the HMAC function. If the key is of a type that supports rotation, the latest (current) version will be used.

**Supported methods:** POST: /{mount_point}/hmac/{name}(/{algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to generate hmac against. This is specified as part of the URL.
- **hash_input** – Specifies the base64 encoded input data.
- **key_version** (*int*) – Specifies the version of the key to use for the operation. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

generate_hmac_response = client.secrets.transit.generate_hmac(
    name='hvac-key',
    hash_input=base64ify('hi its me hvac'),
    algorithm='sha2-256',
)
hmac = generate_hmac_response['data']
print("HMAC'd data is: {hmac}".format(hmac=hmac))
```

Example output:

```
HMAC'd data is: {'hmac': 'vault:...'}
```

### Sign Data

Transit.**sign_data**(*name*, *hash_input*, *key_version=None*, *hash_algorithm=None*, *context=None*, *prehashed=None*, *signature_algorithm=None*, *marshaling_algorithm=None*, *mount_point='transit'*)

Return the cryptographic signature of the given data using the named key and the specified hash algorithm.

The key must be of a type that supports signing.

**Supported methods:** POST: /{mount_point}/sign/{name}(/{hash_algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to use for signing. This is specified as part of the URL.
- **hash_input** (*str | unicode*) – Specifies the base64 encoded input data.
- **key_version** (*int*) – Specifies the version of the key to use for signing. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **hash_algorithm** (*str | unicode*) – Specifies the hash algorithm to use for supporting key types (notably, not including ed25519 which specifies its own hash algorithm). This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **context** (*str | unicode*) – Base64 encoded context for key derivation. Required if key derivation is enabled; currently only available with ed25519 keys.
- **prehashed** (*bool*) – Set to true when the input is already hashed. If the key type is rsa-2048 or rsa-4096, then the algorithm used to hash the input should be indicated by the hash_algorithm parameter. Just as the value to sign should be the base64-encoded representation of the exact binary data you want signed, when set, input is expected to be base64-encoded binary hashed data, not hex-formatted. (As an example, on the command line, you could generate a suitable input via openssl dgst -sha256 -binary | base64.)
- **signature_algorithm** (*str | unicode*) – When using a RSA key, specifies the RSA signature algorithm to use for signing. Supported signature types are: pss, pkcs1v15

- **marshaling_algorithm** (*str | unicode*) – Specifies the way in which the signature should be marshaled. This currently only applies to ECDSA keys. Supported types are: asn1, jws
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

key_name = 'hvac-signing-key'

# Note: some key types do no support signing...
# E.g., "key type aes256-gcm96 does not support verification"
client.secrets.transit.create_key(
    name=key_name,
    key_type='ed25519',
)

sign_data_response = client.secrets.transit.sign_data(
    name=key_name,
    hash_input=base64ify('hi its me hvac'),
)
signature = sign_data_response['data']['signature']
print('Signature is: {signature}'.format(signature=signature))
```

Example output:

```
Signature is: vault:...
```

### Verify Signed Data

Transit.**verify_signed_data**(*name*, *hash_input*, *signature=None*, *hmac=None*, *hash_algorithm=None*, *context=None*, *prehashed=None*, *signature_algorithm=None*, *marshaling_algorithm=None*, *mount_point='transit'*)

Return whether the provided signature is valid for the given data.

**Supported methods:** POST: /{mount_point}/verify/{name}(/{hash_algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key that was used to generate the signature or HMAC.
- **hash_input** – Specifies the base64 encoded input data.
- **signature** (*str | unicode*) – Specifies the signature output from the /transit/sign function. Either this must be supplied or hmac must be supplied.
- **hmac** (*str | unicode*) – Specifies the signature output from the /transit/hmac function. Either this must be supplied or signature must be supplied.

- **hash_algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **context** (*str | unicode*) – Base64 encoded context for key derivation. Required if key derivation is enabled; currently only available with ed25519 keys.
- **prehashed** (*bool*) – Set to true when the input is already hashed. If the key type is rsa-2048 or rsa-4096, then the algorithm used to hash the input should be indicated by the hash_algorithm parameter.
- **signature_algorithm** (*str | unicode*) – When using a RSA key, specifies the RSA signature algorithm to use for signature verification. Supported signature types are: pss, pkcs1v15
- **marshaling_algorithm** (*str | unicode*) – Specifies the way in which the signature should be marshaled. This currently only applies to ECDSA keys. Supported types are: asn1, jws
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

verify_signed_data_response = client.secrets.transit.verify_signed_data(
    name='hvac-signing-key',
    hash_input=base64ify('hi its me hvac'),
    signature=signature,
)
valid = verify_signed_data_response['data']['valid']
print('Signature is valid?: {valid}'.format(valid=valid))
```

Example output:

```
Signature is valid?: True
```

### Backup Key

Transit.**backup_key**(*name*, *mount_point='transit'*)

Return a plaintext backup of a named key.

The backup contains all the configuration data and keys of all the versions along with the HMAC key. The response from this endpoint can be used with the /restore endpoint to restore the key.

**Supported methods:** GET: /{mount_point}/backup/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the key.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

key_name = 'hvac-key'

# Update the key configuration to allow exporting
client.secrets.transit.update_key_configuration(
    name=key_name,
    exportable=True,
    allow_plaintext_backup=True,
)

backup_key_response = client.secrets.transit.backup_key(
    name=key_name,
)

backed_up_key = backup_key_response['data']['backup']
print('Backed up key: %s' % backed_up_key)
```

Example output:

```
Backed up key: ...
```

### Restore Key

Transit.**restore_key**(*backup*, *name=None*, *force=None*, *mount_point='transit'*)

Restore the backup as a named key.

This will restore the key configurations and all the versions of the named key along with HMAC keys. The input to this endpoint should be the output of /backup endpoint. For safety, by default the backend will refuse to restore to an existing key. If you want to reuse a key name, it is recommended you delete the key before restoring. It is a good idea to attempt restoring to a different key name first to verify that the operation successfully completes.

**Supported methods:** POST: /{mount_point}/restore(/name). Produces: 204 (empty body)

**Parameters**

- **backup** (*str | unicode*) – Backed up key data to be restored. This should be the output from the /backup endpoint.
- **name** (*str | unicode*) – If set, this will be the name of the restored key.
- **force** (*bool*) – If set, force the restore to proceed even if a key by this name already exists.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')

client.secrets.transit.update_key_configuration(
    name=key_name,
    deletion_allowed=True,
)
delete_resp = client.secrets.transit.delete_key(name=key_name)

# Restore a key after deletion
client.secrets.transit.restore_key(backup=backed_up_key)
```

### Trim Key

Transit.**trim_key**(*name*, *min_version*, *mount_point='transit'*)

Trims older key versions setting a minimum version for the keyring.

Once trimmed, previous versions of the key cannot be recovered.

**Supported methods:** POST: /{mount_point}/keys/{name}/trim. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to be trimmed.
- **min_version** (*int*) – The minimum version for the key ring. All versions before this version will be permanently deleted. This value can at most be equal to the lesser of min_decryption_version and min_encryption_version. This is not allowed to be set when either min_encryption_version or min_decryption_version is set to zero.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** dict

### Examples

**Note:** Transit key trimming was added for Vault versions >=0.11.4.

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

key_name = 'hvac-key'

for _ in range(0, 10):
    # Rotate the key a bunch...
    client.secrets.transit.rotate_key(
        name=key_name,
    )
```

(continues on next page)

(continued from previous page)

```
# Set a minimum encryption version
client.secrets.transit.update_key_configuration(
    name=key_name,
    min_decryption_version=3,
    min_encryption_version=5,
)

# Trim any unneeded versions remaining of the key...
client.secrets.transit.trim_key(
    name='hvac-key',
    min_version=3,
)
```

## 2.2 Auth Methods

### 2.2.1 Approle

**Contents**



#### Enabling

*hvac.api.system_backend.Auth.enable_auth_method()*

```
import hvac
client = hvac.Client()

client.sys.enable_auth_method(
    method_type='approle',
)

# Mount approle auth method under a different path:
client.sys.enable_auth_method(
    method_type='approle',
    path='my-approle',
)
```

### Authentication

*hvac.api.auth_methods.AppRole.login()*

```
import hvac
client = hvac.Client()


client.auth.approle.login(
    role_id='<some_role_id>',
    secret_id='<some_secret_id>',
)
```

### Create or Update AppRole

*hvac.api.auth_methods.AppRole.create_or_update_approle()*

```
import hvac
client = hvac.Client()

client.auth.approle.create_or_update_approle(
    role_name='some-role',
    token_policies=['some-policy'],
    token_type='service,
)
```

### Read Role ID

*hvac.api.auth_methods.AppRole.read_role_id()*

```
import hvac
client = hvac.Client()

resp = client.auth.approle.read_role_id(
    role_name='some-role',
)
print(f'AppRole role ID for some-role: {resp["data"]["role_id"]}')
```

### Generate Secret ID

*hvac.api.auth_methods.AppRole.generate_secret_id()*

```
import hvac
client = hvac.Client()

resp = client.auth.approle.generate_secret_id(
    role_name='some-role',
    cidr_list=['127.0.0.1/32'],
)
print(f'AppRole secret ID for some-role: {resp["data"]["secret_id"]}')
```

### 2.2.2 AWS

**Contents**

- *Delete Role*
- *Create Role Tags*
- *IAM Login*
- *EC2 Login*
- *Place Role Tags In Blacklist*
- *Read Role Tag Blacklist*
- *List Blacklist Tags*
- *Delete Blacklist Tags*
- *Tidy Blacklist Tags*
- *Read Identity Whitelist*
- *List Identity Whitelist*
- *Delete Identity Whitelist Entries*
- *Tidy Identity Whitelist Entries*

### IAM Authentication

Source reference: *hvac.api.auth_methods.Aws.iam_login()*

### Static Access Key Strings

Various examples of authenticating with static access key strings:

```
import hvac

client = hvac.Client()

client.auth.aws.iam_login('MY_AWS_ACCESS_KEY_ID', 'MY_AWS_SECRET_ACCESS_KEY')
client.auth.aws.iam_login('MY_AWS_ACCESS_KEY_ID', 'MY_AWS_SECRET_ACCESS_KEY', 'MY_AWS_
↪SESSION_TOKEN')
client.auth.aws.iam_login('MY_AWS_ACCESS_KEY_ID', 'MY_AWS_SECRET_ACCESS_KEY', role=
↪'MY_ROLE')
```

### Boto3 Session

Retrieving credentials from a boto3 Session object:

```
import boto3
import hvac

session = boto3.Session()
credentials = session.get_credentials()

client = hvac.Client()
client.auth.aws.iam_login(credentials.access_key, credentials.secret_key, credentials.
↪token)
```

### EC2 Metadata Service

Retrieving static instance role credentials within an EC2 instnace using the EC2 metadata service (the EC2 auth method is probably a better fit for this case, which is outlined below under *EC2 Authentication*):

```
import logging
import requests
from requests.exceptions import RequestException
import hvac

logger = logging.getLogger(__name__)

EC2_METADATA_URL_BASE = 'http://169.254.169.254'

def load_aws_ec2_role_iam_credentials(role_name, metadata_url_base=EC2_METADATA_URL_
↪BASE):
    """
    Requests an ec2 instance's IAM security credentials from the EC2 metadata service.
    :param role_name: Name of the instance's role.
    :param metadata_url_base: IP address for the EC2 metadata service.
    :return: dict, unmarshalled JSON response of the instance's security credentials
    """
    metadata_pkcs7_url = '{base}/latest/meta-data/iam/security-credentials/{role}'.
↪format(
        base=metadata_url_base,
        role=role_name,
    )
    logger.debug("load_aws_ec2_role_iam_credentials connecting to %s" % metadata_
↪pkcs7_url)
    response = requests.get(url=metadata_pkcs7_url)
    response.raise_for_status()
    security_credentials = response.json()
    return security_credentials

credentials = load_aws_ec2_role_iam_credentials('some-instance-role')

client = hvac.Client()
client.auth.aws.iam_login(credentials['AccessKeyId'], credentials['SecretAccessKey'],␣
↪credentials['Token'])
```

### Lambda and/or EC2 Instance

```
import os
import hvac


def infer_credentials_from_iam_role(iam_role):
    on_lambda = 'AWS_LAMBDA_FUNCTION_NAME' in os.environ
    if on_lambda:
        return os.environ['AWS_ACCESS_KEY_ID'], os.environ['AWS_SECRET_ACCESS_KEY'],␣
↪os.environ['AWS_SESSION_TOKEN']
    else:
        security_credentials = load_aws_ec2_role_iam_credentials(iam_role)
        return security_credentials['AccessKeyId'], security_credentials[
↪'SecretAccessKey']
```

(continues on next page)

(continued from previous page)

```
access_key_id, secret_access_key, session_token = infer_credentials_from_iam_role(
↪'some-role')

client = hvac.Client()
client.auth.aws.iam_login(access_key_id, secret_access_key, session_token)
```

### Caveats For Non-Default AWS Regions

I.e., calling *hvac.api.auth_methods.Aws.iam_login()* with a *region* argument other than its default of "**us-east-1**". For additional background / context on this matter, see the comments at hvac#251 and/or vault-ruby#161.

The following code snippets are for authenticating hosts in the **us-west-1** region:

---

**Note:** In order to authenticate to various regions, the AWS auth method configuration needs to be set up with an "endpoint URL" corresponding to the region in question. E.g.: "**https://sts.us-west-1.amazonaws.com**" in the case of this example. Vault defaults to an endpoint of "**https://sts.amazonaws.com**" if not configured with a different endpoint URL.

---

```
import boto3
import hvac

VAULT_ADDR = os.environ["VAULT_ADDR"]
VAULT_HEADER_VALUE = os.environ["VAULT_HEADER_VALUE"]

client = hvac.Client(url=VAULT_ADDR)

# One-time setup of the credentials / configuration for the Vault server to use.
# Note the explicit region subdomain bit included in the endpoint argument.
client.auth.aws.configure(
    access_key='SOME_ACCESS_KEY_FOR_VAULTS_USE',
    secret_key='SOME_ACCESS_KEY_FOR_VAULTS_USE',
    endpoint='https://sts.us-west-1.amazonaws.com',
)

session = boto3.Session()
creds = session.get_credentials().get_frozen_credentials()
client.auth.aws.iam_login((
    access_key=creds.access_key,
    secret_key=creds.secret_key,
    session_token=creds.token,
    header_value=VAULT_HEADER_VALUE,
    role='some-role,
    use_token=True,
    region='us-west-1',
)
```

### EC2 Authentication

Source reference: *hvac.api.auth_methods.Aws.ec2_login()*

### EC2 Metadata Service

Authentication using EC2 instance role credentials and the EC2 metadata service

```
#!/usr/bin/env python
import logging.handlers
import os

import hvac
import requests
from requests.exceptions import RequestException


logger = logging.getLogger(__name__)

VAULT_URL = os.getenv('VAULT_ADDR', 'https://127.0.0.1:8200')
VAULT_CERTS = ('/etc/vault.d/ssl/bundle.crt', '/etc/vault.d/ssl/vault.key')
TOKEN_NONCE_PATH = os.getenv('WP_VAULT_TOKEN_NONCE_PATH', '/root/.vault-token-meta-
↪nonce')
EC2_METADATA_URL_BASE = 'http://169.254.169.254'


def load_aws_ec2_pkcs7_string(metadata_url_base=EC2_METADATA_URL_BASE):
    """
    Requests an ec2 instance's pkcs7-encoded identity document from the EC2 metadata␣
↪service.
    :param metadata_url_base: IP address for the EC2 metadata service.
    :return: string, pkcs7-encoded identity document from the EC2 metadata service
    """
    metadata_pkcs7_url = '{base}/latest/dynamic/instance-identity/pkcs7'.
↪format(base=metadata_url_base)
    logger.debug("load_aws_ec2_pkcs7_string connecting to %s" % metadata_pkcs7_url)

    response = requests.get(url=metadata_pkcs7_url)
    response.raise_for_status()

    pcks7 = response.text.replace('\n', '')

    return pcks7


def load_aws_ec2_nonce_from_disk(token_nonce_path=TOKEN_NONCE_PATH):
    """
    Helper method to load a previously stored "token_meta_nonce" returned in the
    initial authorization AWS EC2 request from the current instance to our Vault␣
↪service.
    :param token_nonce_path: string, the full filesystem path to a file containing␣
↪the instance's
        token meta nonce.
    :return: string, a previously stored "token_meta_nonce"
    """
    logger.debug("Attempting to load vault token meta nonce from path: %s" % token_
↪nonce_path)
```

(continues on next page)

(continued from previous page)

```
    try:
        with open(token_nonce_path, 'rb') as nonce_file:
            nonce = nonce_file.readline()
    except IOError:
        logger.warning("Unable to load vault token meta nonce at path: %s" % token_
↪nonce_path)
        nonce = None

    logger.debug("Nonce loaded: %s" % nonce)
    return nonce


def write_aws_ec2_nonce_to_disk(token_meta_nonce, token_nonce_path=TOKEN_NONCE_PATH):
    """
    Helper method to store the current "token_meta_nonce" returned from authorization␣
↪AWS EC2 request
    from the current instance to our Vault service.
    :return: string, a previously stored "token_meta_nonce"
    :param token_meta_nonce: string, the actual nonce
    :param token_nonce_path: string, the full filesystem path to a file containing␣
↪the instance's
        token meta nonce.
    :return: None
    """
    logger.debug('Writing nonce "{0}" to file "{1}".'.format(token_meta_nonce, token_
↪nonce_path))
    with open(token_nonce_path, 'w') as nonce_file:
        nonce_file.write(token_meta_nonce)


def auth_ec2(vault_client, pkcs7=None, nonce=None, role=None, mount_point='aws',␣
↪store_nonce=True):
    """
    Helper method to authenticate to vault using the "auth_ec2" backend.
    :param vault_client: hvac.Client
    :param pkcs7: pkcs7-encoded identity document from the EC2 metadata service
    :param nonce: string, the nonce retruned from the initial AWS EC2 auth request␣
↪(if applicable)
    :param role: string, the role/policy to request. Defaults to the current instance
↪'s AMI ID if not provided.
    :param mount_point: string, the path underwhich the AWS EC2 auth backend is␣
↪provided
    :param store_nonce: bool, if True, store the nonce received in the auth_ec2␣
↪response on disk for later use.
        Especially useful for automated secure introduction.
    :param kwargs: dict, remaining arguments blindly passed through by this lookup␣
↪module class
    :return: None
    """
    if pkcs7 is None:
        logger.debug('No pkcs7 argument provided to auth_ec2 backend.')
        logger.debug('Attempting to retrieve information from EC2 metadata service.')
        pkcs7 = load_aws_ec2_pkcs7_string()

    if nonce is None:
        logger.debug('No nonce argument provided to auth_ec2 backend.'
        logger.debug('Attempting to retrieve information from disk.')
```

(continues on next page)

(continued from previous page)

```
        nonce = load_aws_ec2_nonce_from_disk()

    auth_ec2_resp = vault_client.auth.aws.ec2_login(
        pkcs7=pkcs7,
        nonce=nonce,
        role=role,
        use_token=False,
        mount_point=mount_point
    )

    if store_nonce and 'metadata' in auth_ec2_resp.get('auth', dict()):
        token_meta_nonce = auth_ec2_resp['auth']['metadata'].get('nonce')
        if token_meta_nonce is not None:
            logger.debug('token_meta_nonce received back from auth_ec2 call: %s' %␣
↪token_meta_nonce)
            write_aws_ec2_nonce_to_disk(token_meta_nonce)
        else:
            logger.warning('No token meta nonce returned in auth response.')

    return auth_ec2_resp


def get_vault_client(vault_url=VAULT_URL, certs=VAULT_CERTS, verify_certs=True, ec2_
↪role=None):
    """
    Instantiates a hvac / vault client.
    :param vault_url: string, protocol + address + port for the vault service
    :param certs: tuple, Optional tuple of self-signed certs to use for verification␣
↪with hvac's requests
    :param verify_certs: bool, if True use the provided certs tuple for verification␣
↪with hvac's requests.
        If False, don't verify SSL with hvac's requests (typically used with local␣
↪development).
    :param ec2_role: str, Name of the Vault AWS auth backend role to use when␣
↪retrieving a token (if applicable)
    :return: hvac.Client
    """
    logger.debug('Retrieving a vault (hvac) client...')
    if verify_certs:
        # We use a self-signed certificate for the vault service itself, so we need␣
↪to include our
        # local ca bundle here for the underlying requests module.
        os.environ['REQUESTS_CA_BUNDLE'] = '/etc/ssl/certs/ca-certificates.crt'
        vault_client = hvac.Client(
            url=vault_url,
            cert=certs,
        )
    else:
        vault_client = hvac.Client(
            url=vault_url,
            verify=False,
        )

    vault_client.token = load_vault_token(vault_client, ec2_role=ec2_role)

    if not vault_client.is_authenticated():
        raise hvac.exceptions.Unauthorized('Unable to authenticate to the Vault␣
↪service')
```

(continues on next page)

(continued from previous page)

```
    return vault_client


authenticated_vault_client = get_vault_client()
```

### Methods

#### Configure

Source reference: *hvac.api.auth_methods.Aws.configure()*

#### Read Config

Source reference: *hvac.api.auth_methods.Aws.read_config()*

#### Delete Config

Source reference: *hvac.api.auth_methods.Aws.delete_config()*

#### Configure Identity Integration

Source reference: *hvac.api.auth_methods.Aws.configure_identity_integration()*

#### Read Identity Integration

Source reference: *hvac.api.auth_methods.Aws.read_identity_integration()*

#### Create Certificate Configuration

Source reference: *hvac.api.auth_methods.Aws.create_certificate_configuration()*

#### Read Certificate Configuration

Source reference: *hvac.api.auth_methods.Aws.read_certificate_configuration()*

#### Delete Certificate Configuration

Source reference: *hvac.api.auth_methods.Aws.delete_certificate_configuration()*

#### List Certificate Configurations

Source reference: *hvac.api.auth_methods.Aws.list_certificate_configurations()*

#### Create Sts Role

Source reference: *hvac.api.auth_methods.Aws.create_sts_role()*

#### Read Sts Role

Source reference: *hvac.api.auth_methods.Aws.read_sts_role()*

#### List Sts Roles

Source reference: *hvac.api.auth_methods.Aws.list_sts_roles()*

#### Delete Sts Role

Source reference: *hvac.api.auth_methods.Aws.delete_sts_role()*

#### Configure Identity Whitelist Tidy

Source reference: *hvac.api.auth_methods.Aws.configure_identity_whitelist_tidy()*

#### Read Identity Whitelist Tidy

Source reference: *hvac.api.auth_methods.Aws.read_identity_whitelist_tidy()*

#### Delete Identity Whitelist Tidy

Source reference: *hvac.api.auth_methods.Aws.delete_identity_whitelist_tidy()*

#### Configure Role Tag Blacklist Tidy

Source reference: *hvac.api.auth_methods.Aws.configure_role_tag_blacklist_tidy()*

#### Read Role Tag Blacklist Tidy

Source reference: *hvac.api.auth_methods.Aws.read_role_tag_blacklist_tidy()*

#### Delete Role Tag Blacklist Tidy

Source reference: *hvac.api.auth_methods.Aws.delete_role_tag_blacklist_tidy()*

#### Create Role

Source reference: *hvac.api.auth_methods.Aws.create_role()*

#### Read Role

Source reference: *hvac.api.auth_methods.Aws.read_role()*

#### List Roles

Source reference: *hvac.api.auth_methods.Aws.list_roles()*

#### Delete Role

Source reference: *hvac.api.auth_methods.Aws.delete_role()*

#### Create Role Tags

Source reference: *hvac.api.auth_methods.Aws.create_role_tags()*

#### IAM Login

Source reference: *hvac.api.auth_methods.Aws.iam_login()*

#### EC2 Login

Source reference: *hvac.api.auth_methods.Aws.ec2_login()*

#### Place Role Tags In Blacklist

Source reference: *hvac.api.auth_methods.Aws.place_role_tags_in_blacklist()*

#### Read Role Tag Blacklist

Source reference: *hvac.api.auth_methods.Aws.read_role_tag_blacklist()*

#### List Blacklist Tags

Source reference: *hvac.api.auth_methods.Aws.list_blacklist_tags()*

#### Delete Blacklist Tags

Source reference: *hvac.api.auth_methods.Aws.delete_blacklist_tags()*

#### Tidy Blacklist Tags

Source reference: *hvac.api.auth_methods.Aws.tidy_blacklist_tags()*

#### Read Identity Whitelist

Source reference: *hvac.api.auth_methods.Aws.read_identity_whitelist()*

#### List Identity Whitelist

Source reference: *hvac.api.auth_methods.Aws.list_identity_whitelist()*

#### Delete Identity Whitelist Entries

Source reference: *hvac.api.auth_methods.Aws.delete_identity_whitelist_entries()*

#### Tidy Identity Whitelist Entries

Source reference: *hvac.api.auth_methods.Aws.tidy_identity_whitelist_entries()*

### 2.2.3 Azure

**Note:** Every method under the `Client class's azure attribute` includes a *mount_point* parameter that can be used to address the Azure auth method under a custom mount path. E.g., If enabling the Azure auth method using Vault's CLI commands via *vault auth enable -path=my-azure azure*", the *mount_point* parameter in *hvac.api.auth_methods.Azure()* methods would be set to "my-azure".

#### Enabling the Auth Method

*hvac.v1.Client.enable_auth_backend()*

```
import hvac
client = hvac.Client()

azure_auth_path = 'company-azure'
description = 'Auth method for use by team members in our company's Azure organization
↪'

if '%s/' % azure_auth_path not in vault_client.list_auth_backends():
    print('Enabling the azure auth backend at mount_point: {path}'.format(
        path=azure_auth_path,
    ))
    client.enable_auth_backend(
        backend_type='azure',
        description=description,
        mount_point=azure_auth_path,
    )
```

#### Configure

*hvac.api.auth_methods.Azure.configure()*

```
import os
import hvac
client = hvac.Client()

client.auth.azure.configure(
    tenant_id='my-tenant-id'
    resource='my-resource',
    client_id=os.environ.get('AZURE_CLIENT_ID'),
    client_secret=os.environ.get('AZURE_CLIENT_SECRET'),
)
```

### Read Config

*hvac.api.auth_methods.Azure.read_config()*

```
import hvac
client = hvac.Client()

read_config = client.auth.azure.read_config()
print('The configured tenant_id is: {id}'.format(id=read_config['tenant_id']))
```

### Delete Config

*hvac.api.auth_methods.Azure.delete_config()*

```
import hvac
client = hvac.Client()

client.auth.azure.delete_config()
```

### Create a Role

*hvac.api.auth_methods.Azure.create_role()*

```
import hvac
client = hvac.Client()

client.auth.azure.create_role(
    name='my-role',
    policies=policies,
    bound_service_principal_ids=bound_service_principal_ids,
)
```

### Read A Role

*hvac.api.auth_methods.Azure.read_role()*

```
import hvac
client = hvac.Client()

role_name = 'my-role'
read_role_response = client.auth.azure.read_role(
    name=role_name,
)
print('Policies for role "{name}": {policies}'.format(
    name='my-role',
    policies=','.join(read_role_response['policies']),
))
```

#### List Roles

*hvac.api.auth_methods.Azure.list_roles()*

```
import hvac
client = hvac.Client()

roles = client.auth.azure.list_roles()
print('The following Azure auth roles are configured: {roles}'.format(
    roles=','.join(roles['keys']),
))
```

#### Delete A Role

*hvac.api.auth_methods.Azure.delete_role()*

```
import hvac
client = hvac.Client()

client.auth.azure.delete_role(
    name='my-role',
)
```

#### Login

*hvac.api.auth_methods.Azure.login()*

```
import hvac
client = hvac.Client()

client.auth.azure.login(
    role=role_name,
    jwt='Some MST JWT...',
)
client.is_authenticated  # ==> returns True
```

### 2.2.4 GCP

**Note:** Every method under the `Client class's gcp.auth attribute` includes a *mount_point* parameter that can be used to address the GCP auth method under a custom mount path. E.g., If enabling the GCP auth method using Vault's CLI commands via *vault auth enable -path=my-gcp gcp*", the *mount_point* parameter in `hvac.api.auth.Gcp()` methods would be set to "my-gcp".

### Enabling the Auth Method

Source reference: *hvac.v1.Client.enable_auth_backend()*

```
import hvac
client = hvac.Client()

gcp_auth_path = 'company-gcp'
description = 'Auth method for use by team members in our company's Gcp organization'

if '%s/' % gcp_auth_path not in vault_client.list_auth_backends():
    print('Enabling the gcp auth backend at mount_point: {path}'.format(
        path=gcp_auth_path,
    ))
    client.enable_auth_backend(
        backend_type='gcp',
        description=description,
        mount_point=gcp_auth_path,
    )
```

### Configure

Source reference: `hvac.api.auth.Gcp.configure()`

```
import hvac
client = hvac.Client()

client.auth.gcp.configure(
    credentials='some signed JSON web token for the Vault server...'
)
```

### Read Config

Source reference: `hvac.api.auth.Gcp.read_config()`

```
import hvac
client = hvac.Client()

read_config = client.auth.gcp.read_config()
print('The configured project_id is: {id}'.format(id=read_config['project_id']))
```

### Delete Config

Source reference: `hvac.api.auth.Gcp.delete_config()`

```
import hvac
client = hvac.Client()

client.auth.gcp.delete_config()
```

### Create Role

Source reference: `hvac.api.auth.Gcp.create_role()`

```
import hvac
client = hvac.Client()

    client.auth.gcp.create_role(
            name='some-gcp-role-name',
            role_type='iam',
            project_id='some-gcp-project-id',
            bound_service_accounts=['*'],
    )
```

### Edit Service Accounts On IAM Role

Source reference: `hvac.api.auth.Gcp.edit_service_accounts_on_iam_role()`

```
import hvac
client = hvac.Client()

client.gcp.edit_service_accounts_on_iam_role(
            name='some-gcp-role-name',
    add=['hvac@appspot.gserviceaccount.com'],
)

client.gcp.edit_service_accounts_on_iam_role(
            name='some-gcp-role-name',
    remove=['disallowed-service-account@appspot.gserviceaccount.com'],
)
```

### Edit Labels On GCE Role

Source reference: `hvac.api.auth.Gcp.edit_labels_on_gce_role()`

```
import hvac
client = hvac.Client()

client.gcp.edit_labels_on_gce_role(
            name='some-gcp-role-name',
    add=['some-key:some-value'],
)

client.gcp.edit_labels_on_gce_role(
            name='some-gcp-role-name',
    remove=['some-bad-key:some-bad-value'],
)
```

### Read A Role

Source reference: `hvac.api.auth.Gcp.read_role()`

```
import hvac
client = hvac.Client()

read_role_response = client.gcp.read_role(
    name=role_name,
)

print('Policies for role "{name}": {policies}'.format(
    name='my-role',
    policies=','.join(read_role_response['policies']),
))
```

### List Roles

Source reference: `hvac.api.auth.Gcp.list_roles()`

```
import hvac
client = hvac.Client()

roles = client.auth.gcp.list_roles()
print('The following GCP auth roles are configured: {roles}'.format(
    roles=','.join(roles['keys']),
))
```

### Delete A Role

Source reference: `hvac.api.auth.Gcp.delete_role()`

```
import hvac
client = hvac.Client()

client.gcp.delete_role(
)
```

### Login

Source reference: `hvac.api.auth.Gcp.login()`

```
import hvac
client = hvac.Client()

client.gcp.login(
    role=role_name,
    jwt='some signed JSON web token...',
)
client.is_authenticated  # ==> returns True
```

#### Example with google-api-python-client Usage

```
import time

import googleapiclient.discovery # pip install google-api-python-client
from google.oauth2 import service_account # pip install google-auth
import hvac # pip install hvac

# First load some previously generated GCP service account key
path_to_sa_json = 'some-service-account-path.json'
credentials = service_account.Credentials.from_service_account_file(path_to_sa_json)
with open(path_to_sa_json, 'r') as f:
    creds = json.load(f)
    project = creds['project_id']
    service_account = creds['client_email']

# Generate a payload for subsequent "signJwt()" call
# Reference: https://google-auth.readthedocs.io/en/latest/reference/google.auth.jwt.
↪html#google.auth.jwt.Credentials
now = int(time.time())
expires = now + 900  # 15 mins in seconds, can't be longer.
payload = {
    'iat': now,
    'exp': expires,
    'sub': service_account,
    'aud': 'vault/my-role'
}
body = {'payload': json.dumps(payload)}
name = f'projects/{project}/serviceAccounts/{service_account}'

# Perform the GCP API call
iam = googleapiclient.discovery.build('iam', 'v1', credentials=credentials)
request = iam.projects().serviceAccounts().signJwt(name=name, body=body)
resp = request.execute()
jwt = resp['signedJwt']

# Perform hvac call to configured GCP auth method
client.auth.gcp.login(
    role='my-role',
    jwt=jwt,
)
```

### 2.2.5 GitHub

**Note:** Every method under the `Client class's github attribute` includes a *mount_point* parameter that can be used to address the Github auth method under a custom mount path. E.g., If enabling the Github auth method using Vault's CLI commands via *vault auth enable -path=my-github github*", the *mount_point* parameter in *hvac.api.auth_methods.Github()* methods would be set to "my-github".

### Enabling the Auth Method

*hvac.v1.Client.enable_auth_backend()*

```
import hvac
client = hvac.Client()

github_auth_path = 'company-github'
description = 'Auth method for use by team members in our company's Github␣
↪organization'

if '%s/' % github_auth_path not in vault_client.list_auth_backends():
    print('Enabling the github auth backend at mount_point: {path}'.format(
        path=github_auth_path,
    ))
    client.enable_auth_backend(
        backend_type='github',
        description=description,
        mount_point=github_auth_path,
    )
```

### Configure Connection Parameters

*hvac.api.auth_methods.Github.configure()*

```
import hvac
client = hvac.Client()

client.auth.github.configure(
    organization='our-lovely-company',
    max_ttl='48h',  # i.e., A given token can only be renewed for up to 48 hours
)
```

### Reading Configuration

*hvac.api.auth_methods.Github.read_configuration()*

```
import hvac
client = hvac.Client()

github_config = client.auth.github.read_configuration()
print('The Github auth method is configured with a ttl of: {ttl}'.format(
    ttl=github_config['data']['ttl']
)
```

### Mapping Teams to Policies

*hvac.api.auth_methods.Github.map_team()*

```
import hvac
client = hvac.Client()

teams = [
    dict(name='some-dev-team', policies=['dev-team']),
    dict(name='admin-team', policies=['administrator']),
]
for team in teams:
    client.auth.github.map_team(
        team_name=team['name'],
        policies=team['policies'],
    )
```

### Reading Team Mappings

*hvac.api.auth_methods.Github.read_team_mapping()*

```
import hvac
client = hvac.Client()

team_name = 'my-super-cool-team'
github_config = client.auth.github.read_team_mapping(
    team_name=team_name,
)
print('The Github team {team} is mapped to the following policies: {policies}'.format(
    team=team_name,
    policies=github_config['data']['value'],
)
```

### Mapping Users to Policies

*hvac.api.auth_methods.Github.map_user()*

```
import hvac
client = hvac.Client()

users = [
    dict(name='some-dev-user', policies=['dev-team']),
    dict(name='some-admin-user', policies=['administrator']),
]
for user in users:
    client.auth.github.map_user(
        user_name=user['name'],
        policies=user['policies'],
    )
```

#### Reading User Mappings

*hvac.api.auth_methods.Github.read_user_mapping()*

```
import hvac
client = hvac.Client()

user_name = 'some-dev-user'
github_config = client.auth.github.read_user_mapping(
    user_name=user_name,
)
print('The Github user "{user}" is mapped to the following policies: {policies}'.
↪format(
    user=user_name,
    policies=github_config['data']['value'],
)
```

#### Authentication / Login

*hvac.api.auth_methods.Github.login()*

Log in and automatically update the underlying "token" attribute on the *hvac.adapters.Adapter()* instance:

```
import hvac
client = hvac.Client()
login_response = client.auth.github.login(token='some personal github token')
```

### 2.2.6 JWT/OIDC

**Contents**



**Note:** The *hvac.api.auth_methods.JWT* and *hvac.api.auth_methods.OIDC* share all the same methods. They only differ in the default path their methods will use. I.e., *v1/auth/jwt* versus *v1/auth/oidc*.

### Enabling

```
import hvac
client = hvac.Client()

# For JWT
client.sys.enable_auth_method(
    method_type='jwt',
)

# For OIDC
client.sys.enable_auth_method(
    method_type='oidc',
)
```

### Configure

*hvac.api.auth_methods.JWT.configure()*

```
import hvac
client = hvac.Client()

client.auth.jwt.configure(
    oidc_discovery_url=oidc_discovery_url,
    oidc_discovery_ca_pem=some_ca_file_contents,
)

# or

client.auth.oidc.configure(
    oidc_discovery_url=oidc_discovery_url,
    oidc_discovery_ca_pem=some_ca_file_contents,
)
```

### Read Config

*hvac.api.auth_methods.JWT.read_config()*

```
import hvac
client = hvac.Client()

read_response = client.auth.jwt.read_config()
# or
read_response = client.auth.oidc.read_config()

discovery_url = read_response['data']['oidc_discovery_url']
print('Current OIDC discovery URL is set to: %s' % discovery_url)
```

### Create Role

*hvac.api.auth_methods.JWT.create_role()*

```
import hvac
client = hvac.Client()

role_name = 'hvac'
allowed_redirect_uris = ['https://localhost:8200/jwt-test/callback']
user_claim = 'https://vault/user'

# JWT
client.auth.jwt.create_role(
    name=role_name,
    role_type='jwt',
    allowed_redirect_uris=allowed_redirect_uris,
    user_claim='sub',
    bound_audiences=['12345'],
)

# OIDC
client.auth.oidc.create_role(
    name=role_name,
    allowed_redirect_uris=allowed_redirect_uris,
    user_claim=user_claim,
)
```

### Read Role

*hvac.api.auth_methods.JWT.read_role()*

```
import hvac
client = hvac.Client()

response = client.auth.jwt.read_role(
    name='hvac',
)
print('hvac role has a user_claim setting of: %s' % response['data']['user_claim'])
```

### List Roles

*hvac.api.auth_methods.JWT.list_roles()*

```
import hvac
client = hvac.Client()

list_resp = client.auth.jwt.list_roles()
print('Configured roles: %s' % ', '.join(list_resp['data']['keys']))
```

### Delete Role

*hvac.api.auth_methods.JWT.delete_role()*

```
import hvac
client = hvac.Client()

client.auth.jwt.delete_role(
    name='hvac',
)
```

### OIDC Authorization URL Request

*hvac.api.auth_methods.JWT.oidc_authorization_url_request()*

```
import webbrowser
import http.server
import hvac
client = hvac.Client()

auth_url_response = client.auth.oidc.oidc_authorization_url_request(
    role='hvac',
    redirect_uri='http://localhost:8250/oidc/callback'
)
auth_url = auth_url_response['data']['auth_url']
if auth_url == '':
    return None

params = parse.parse_qs(auth_url.split('?')[1])
auth_url_nonce = params['nonce'][0]
auth_url_state = params['state'][0]

webbrowser.open(auth_url)
token = login_odic_get_token()

auth_result = client.auth.oidc.oidc_callback(
    code=token, path='oidc', nonce=auth_url_nonce, state=auth_url_state
)

print('Client token returned: %s' % auth_result['auth']['client_token'])

# handles the callback
def login_odic_get_token(self):
    from http.server import BaseHTTPRequestHandler, HTTPServer

    class HttpServ(HTTPServer):
        def __init__(self, *args, **kwargs):
            HTTPServer.__init__(self, *args, **kwargs)
            self.token = None

    class AuthHandler(BaseHTTPRequestHandler):
        token = ''

        def do_GET(self):
            params = parse.parse_qs(self.path.split('?')[1])
            self.server.token = params['code'][0]
```

(continues on next page)

(continued from previous page)

```
            self.send_response(200)
            self.end_headers()
            self.wfile.write(str.encode('<div>Authentication successful, you can␣
↪close the browser now.</div>'))

    server_address = ('', 8250)
    httpd = HttpServ(server_address, AuthHandler)
    httpd.handle_request()
    return httpd.token
```

#### JWT Login

*hvac.api.auth_methods.JWT.jwt_login()*

```
import hvac
client = hvac.Client()

response = client.auth.jwt.jwt_login(
    role=role_name,
    jwt=generate_token_response['data']['token'],
)
print('Client token returned: %s' % response['auth']['client_token'])
```

### 2.2.7 Kubernetes

#### Authentication

```
# Kubernetes (from k8s pod)
f = open('/var/run/secrets/kubernetes.io/serviceaccount/token')
jwt = f.read()
client.auth_kubernetes("example", jwt)
```

### 2.2.8 LDAP

**Note:** Every method under the `Client class's ldap attribute` includes a *mount_point* parameter that can be used to address the LDAP auth method under a custom mount path. E.g., If enabling the LDAP auth method using Vault's CLI commands via *vault auth enable -path=my-ldap ldap*", the *mount_point* parameter in *hvac.api.auth_methods.Ldap()* methods would be set to "my-ldap".

### Enabling the LDAP Auth Method

*hvac.v1.Client.enable_auth_backend()*

```
import hvac
client = hvac.Client()

ldap_auth_path = 'company-ldap'
description = "Auth method for use by team members in our company's LDAP organization"

if '%s/' % ldap_auth_path not in vault_client.sys.list_auth_methods():
    print('Enabling the ldap auth backend at mount_point: {path}'.format(
        path=ldap_auth_path,
    ))
    client.enable_auth_backend(
        backend_type='ldap',
        description=description,
        mount_point=ldap_auth_path,
    )
```

### Configure LDAP Auth Method Settings

*hvac.api.auth_methods.Ldap.configure()*

```
import hvac
client = hvac.Client()

client.auth.ldap.configure(
    user_dn='dc=users,dc=hvac,dc=network',
    group_dn='ou=groups,dc=hvac,dc=network',
    url='ldaps://ldap.hvac.network:12345',
    bind_dn='cn=admin,dc=hvac,dc=network',
    bind_pass='ourverygoodadminpassword'
    user_attr='uid',
    group_attr='cn',
)
```

### Reading the LDAP Auth Method Configuration

*hvac.api.auth_methods.Ldap.read_configuration()*

```
import hvac
client = hvac.Client()

ldap_configuration = client.auth.ldap.read_configuration()
print('The LDAP auth method is configured with a LDAP server URL of: {url}'.format(
    url=ldap_configuration['data']['url']
)
```

### Create or Update a LDAP Group Mapping

*hvac.api.auth_methods.Ldap.create_or_update_group()*

```
import hvac
client = hvac.Client()

client.auth.ldap.create_or_update_group(
    name='some-dudes',
    policies=['policy-for-some-dudes'],
)
```

### List LDAP Group Mappings

*hvac.api.auth_methods.Ldap.list_groups()*

```
import hvac
client = hvac.Client()

ldap_groups = client.auth.ldap.list_groups()
print('The following groups are configured in the LDAP auth method: {groups}'.format(
    groups=','.join(ldap_groups['data']['keys'])
)
```

### Read LDAP Group Mapping

*hvac.api.auth_methods.Ldap.read_group()*

```
import hvac
client = hvac.Client()

some_dudes_ldap_group = client.auth.ldap.read_group(
    name='somedudes',
)
print('The "somedudes" group in the LDAP auth method are mapped to the following␣
↪policies: {policies}'.format(
    policies=','.join(some_dudes_ldap_group['data']['policies'])
)
```

### Deleting a LDAP Group Mapping

*hvac.api.auth_methods.Ldap.delete_group()*

```
import hvac
client = hvac.Client()

client.auth.ldap.delete_group(
    name='some-group',
)
```

### Creating or Updating a LDAP User Mapping

*hvac.api.auth_methods.Ldap.create_or_update_user()*

```
import hvac
client = hvac.Client()

client.auth.ldap.create_or_update_user(
    username='somedude',
    policies=['policy-for-some-dudes'],
)
```

### Listing LDAP User Mappings

*hvac.api.auth_methods.Ldap.list_users()*

```
import hvac
client = hvac.Client()

ldap_users = client.auth.ldap.list_users()
print('The following users are configured in the LDAP auth method: {users}'.format(
    users=','.join(ldap_users['data']['keys'])
)
```

### Reading a LDAP User Mapping

*hvac.api.auth_methods.Ldap.read_user()*

```
import hvac
client = hvac.Client()

some_dude_ldap_user = client.auth.ldap.read_user(
    username='somedude'
)
print('The "somedude" user in the LDAP auth method is mapped to the following␣
↪policies: {policies}'.format(
    policies=','.join(some_dude_ldap_user['data']['policies'])
)
```

### Deleting a Configured User Mapping

*hvac.api.auth_methods.Ldap.delete_user()*

```
import hvac
client = hvac.Client()

client.auth.ldap.delete_user(
    username='somedude',
)
```

#### Authentication / Login

`hvac.api.auth_methods.Ldap.login_with_user()`

For a LDAP backend mounted under a non-default (ldap) path. E.g., via Vault CLI with *vault auth enable -path=prod-ldap ldap*

```
from getpass import getpass

import hvac

service_account_username = 'someuser'
password_prompt = 'Please enter your password for the LDAP authentication backend: '
service_account_password = getpass(prompt=password_prompt)

client = hvac.Client()

# Here the mount_point parameter corresponds to the path provided when enabling the␣
↪backend
client.auth.ldap.login(
    username=service_account_username,
    password=service_account_password,
    mount_point='prod-ldap'
)
print(client.is_authenticated())  # => True
```

### 2.2.9 MFA

#### Configure MFA Auth Method Settings

*hvac.api.auth_methods.Mfa.configure()*

**Note:** The legacy/unsupported MFA auth method covered by this class's configuration API route only supports integration with a subset of Vault auth methods. See the list of supported auth methods in this module's `"SUPPORTED_AUTH_METHODS" attribute` and/or the associated Vault MFA documentation for additional information.

```
import hvac
client = hvac.Client()

userpass_auth_path = 'some-userpass'

if '%s/' % userpass_auth_path not in vault_client.list_auth_backends():
    print('Enabling the userpass auth backend at mount_point: {path}'.format(
        path=userpass_auth_path,
    ))
    client.enable_auth_backend(
        backend_type='userpass',
        mount_point=userpass_auth_path,
    )

client.auth.mfa.configure(
    mount_point=userpass_auth_path,
)
```

### Reading the MFA Auth Method Configuration

*hvac.api.auth_methods.Mfa.read_configuration()*

```
import hvac
client = hvac.Client()

mfa_configuration = client.auth.mfa.read_configuration()
print('The MFA auth method is configured with a MFA type of: {mfa_type}'.format(
    mfa_type=mfa_configuration['data']['type']
)
```

### Configure Duo MFA Type Access Credentials

*hvac.api.auth_methods.Mfa.configure_duo_access()*

```
from getpass import getpass

import hvac
client = hvac.Client()

secret_key_prompt = 'Please enter the Duo access secret key to configure: '
duo_access_secret_key = getpass(prompt=secret_key_prompt)

client.auth.mfa.configure_duo_access(
    mount_point=userpass_auth_path,
    host='api-1234abcd.duosecurity.com',
    integration_key='SOME_DUO_IKEY',
    secret_key=duo_access_secret_key,
)
```

### Configure Duo MFA Type Behavior

*hvac.api.auth_methods.Mfa.configure_duo_behavior()*

```
import hvac
client = hvac.Client()

client.auth.mfa.configure_duo_behavior(
    mount_point=userpass_auth_path,
    username_format='%s@hvac.network',
)
```

### Read Duo MFA Type Behavior

*hvac.api.auth_methods.Mfa.read_duo_behavior_configuration()*

```
import hvac
client = hvac.Client()

duo_behavior_config = client.auth.mfa.read_duo_behavior_configuration(
    mount_point=userpass_auth_path,
)
```

(continues on next page)

(continued from previous page)

```
print('The Duo MFA behvaior is configured with a username_format of: {username_format}
↪'.format(
    username_format=duo_behavior_config['data']['username_format'],
)
```

#### Authentication / Login

```
from getpass import getpass

import hvac

login_username = 'someuser'
password_prompt = 'Please enter your password for the userpass (with MFA)␣
↪authentication backend: '
login_password = getpass(prompt=password_prompt)
passcode_prompt = 'Please enter your OTP for the userpass (with MFA) authentication␣
↪backend: '
userpass_mfa_passcode = getpass(prompt=passcode_prompt)

client = hvac.Client()

# Here the mount_point parameter corresponds to the path provided when enabling the␣
↪backend
client.auth.mfa.auth_userpass(
    username=login_username,
    password=login_password,
    mount_point=userpass_auth_path,
    passcode=userpass_mfa_passcode,
)
print(client.is_authenticated)  # => True
```

### 2.2.10 Okta

**Note:** Every method under the `Client class's okta attribute` includes a *mount_point* parameter that can be used to address the Okta auth method under a custom mount path. E.g., If enabling the Okta auth method using Vault's CLI commands via *vault secret enable -path=my-okta okta*", the *mount_point* parameter in Source reference: *hvac.api.auth_methods.Okta()* methods would be set to "my-okta".

#### Enabling the Auth Method

Source reference: *hvac.v1.Client.enable_secret_backend()*

```
import hvac
client = hvac.Client()

okta_path = 'company-okta'
description = 'Auth method for use by team members in our company's Okta organization'

if '%s/' % okta_path not in vault_client.sys.list_auth_methods():
```

(continues on next page)

(continued from previous page)

```
    print('Enabling the okta secret backend at mount_point: {path}'.format(
        path=okta_secret_path,
    ))
    client.enable_secret_backend(
        backend_type='okta',
        description=description,
        mount_point=okta_secret_path,
    )
```

### Configure

Source reference: *hvac.api.auth_methods.Okta.configure()*

```
import hvac
client = hvac.Client()

client.auth.okta.configure(
    org_name='hvac-project'
)
```

### Read Config

Source reference: *hvac.api.auth_methods.Okta.read_config()*

```
import hvac
client = hvac.Client()

okta_config = client.auth.okta.read_config()
print('The Okta auth method at path /okta has a configured organization name of:
↪{name}'.format(
    name=okta_config['data']['org_name'],
))
```

### List Users

Source reference: *hvac.api.auth_methods.Okta.list_users()*

```
import hvac
client = hvac.Client()

users = client.auth.okta.list_users()
print('The following Okta users are registered: {users}'.format(
    users=','.join(users['data']['keys']),
))
```

### Register User

Source reference: *hvac.api.auth_methods.Okta.register_user()*

```
import hvac
client = hvac.Client()

client.auth.okta.register_user(
    username='hvac-person',
    policies=['hvac-admin'],
)
```

### Read User

Source reference: *hvac.api.auth_methods.Okta.read_user()*

```
import hvac
client = hvac.Client()

read_user = client.auth.okta.read_user(
    username='hvac-person',
)
print('Okta user "{name}" has the following attached policies: {policies}'.format(
    name='hvac-person',
    policies=', '.join(read_user['data']['policies'],
))
```

### Delete User

Source reference: *hvac.api.auth_methods.Okta.delete_user()*

```
import hvac
client = hvac.Client()

client.auth.okta.delete_user(
    username='hvac-person'
)
```

### List Groups

Source reference: *hvac.api.auth_methods.Okta.list_groups()*

```
import hvac
client = hvac.Client()

groups = client.auth.okta.list_groups()
print('The following Okta groups are registered: {groups}'.format(
    groups=','.join(groups['data']['keys']),
))
```

### Register Group

Source reference: *hvac.api.auth_methods.Okta.register_group()*

```
import hvac
client = hvac.Client()

client.auth.okta.register_group(
    name='hvac-group',
    policies=['hvac-group-members'],
)
```

### Read Group

Source reference: *hvac.api.auth_methods.Okta.read_group()*

```
import hvac
client = hvac.Client()

read_group = client.auth.okta.read_group(
    name='hvac-group',
)
print('Okta group "{name}" has the following attached policies: {policies}'.format(
    name='hvac-group',
    policies=', '.join(read_group['data']['policies'],
))
```

### Delete Group

Source reference: *hvac.api.auth_methods.Okta.delete_group()*

```
import hvac
client = hvac.Client()

client.auth.okta.delete_group(
    name='hvac-group',
)
```

### Login

Source reference: *hvac.api.auth_methods.Okta.login()*

```
from getpass import getpass

import hvac
client = hvac.Client()


password_prompt = 'Please enter your password for the Okta authentication backend: '
okta_password = getpass(prompt=password_prompt)

client.auth.okta.login(
    username='hvac-person',
```

(continues on next page)

(continued from previous page)

```
    password=okta_password,
)
```

### 2.2.11 Token

#### Authentication

```
# Token
client.token = 'MY_TOKEN'
assert client.is_authenticated() # => True
```

#### Token Management

Token creation and revocation:

```
token = client.create_token(policies=['root'], lease='1h')

current_token = client.lookup_token()
some_other_token = client.lookup_token('xxx')

client.revoke_token('xxx')
client.revoke_token('yyy', orphan=True)

client.revoke_token_prefix('zzz')

client.renew_token('aaa')
```

Lookup and revoke tokens via a token accessor:

```
token = client.create_token(policies=['root'], lease='1h')
token_accessor = token['auth']['accessor']

same_token = client.lookup_token(token_accessor, accessor=True)
client.revoke_token(token_accessor, accessor=True)
```

Wrapping/unwrapping a token:

```
wrap = client.create_token(policies=['root'], lease='1h', wrap_ttl='1m')
result = self.client.unwrap(wrap['wrap_info']['token'])
```

Login with a wrapped token:

```
wrap = client.create_token(policies=['root'], lease='1h', wrap_ttl='1m')
new_client = hvac.Client()
new_client.auth_cubbyhole(wrap['wrap_info']['token'])
assert new_client.token != wrapped_token['wrap_info']['token']
```

### 2.2.12 Authenticate to different auth backends

```
# App ID
client.auth_app_id('MY_APP_ID', 'MY_USER_ID')

# GitHub
client.auth_github('MY_GITHUB_TOKEN')

# TLS
client = Client(cert=('path/to/cert.pem', 'path/to/key.pem'))
client.auth_tls()

# Non-default mount point (available on all auth types)
client.auth_userpass('MY_USERNAME', 'MY_PASSWORD', mount_point='CUSTOM_MOUNT_POINT')

# Authenticating without changing to new token (available on all auth types)
result = client.auth_github('MY_GITHUB_TOKEN', use_token=False)
print(result['auth']['client_token']) # => u'NEW_TOKEN'

# Custom or unsupported auth type
params = {
    'username': 'MY_USERNAME',
    'password': 'MY_PASSWORD',
    'custom_param': 'MY_CUSTOM_PARAM',
}

result = client.login('/v1/auth/CUSTOM_AUTH/login', json=params)

# Logout
client.logout()
```

## 2.3 System Backend

### 2.3.1 Audit

### Examples

```
audit_devices = client.sys.list_enabled_audit_devices()

options = {
    'path': '/tmp/vault.log',
    'log_raw': True,
}

client.sys.enable_audit_device('file', options=options, path='somefile')
client.sys.disable_audit_device('oldfile')
```

### List Enabled Audit Devices

Audit.**list_enabled_audit_devices**()

List enabled audit devices.

It does not list all available audit devices. This endpoint requires sudo capability in addition to any path-specific capabilities.

**Supported methods:** GET: /sys/audit. Produces: 200 application/json

**Returns** JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

enabled_audit_devices = client.sys.list_enabled_audit_devices()
print('The following audit devices are enabled: {audit_devices_list}'.format(
    audit_devices_list=', '.join(enabled_audit_devices['data'].keys()),
))
```

Example output:

```
The following audit devices are enabled: somefile/
```

### Enable Audit Device

Audit.**enable_audit_device**(*device_type*, *description=None*, *options=None*, *path=None*, *local=None*)

Enable a new audit device at the supplied path.

The path can be a single word name or a more complex, nested path.

**Supported methods:** PUT: /sys/audit/{path}. Produces: 204 (empty body)

**Parameters**

- **device_type** (*str | unicode*) – Specifies the type of the audit device.
- **description** (*str | unicode*) – Human-friendly description of the audit device.

- **options** (*str | unicode*) – Configuration options to pass to the audit device itself. This is dependent on the audit device type.
- **path** (*str | unicode*) – Specifies the path in which to enable the audit device. This is part of the request URL.
- **local** (*bool*) – Specifies if the audit device is a local only.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

options = {
    'path': '/tmp/vault.audit.log'
}

client.sys.enable_audit_device(
    device_type='file',
    options=options,
    path='tmp-file-audit',
)
```

### Disable Audit Device

Audit.**disable_audit_device**(*path*)

Disable the audit device at the given path.

**Supported methods:** DELETE: /sys/audit/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – The path of the audit device to delete. This is part of the request URL.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.disable_audit_device(
    path='tmp-file-audit',
)
```

### Calculate Hash

Audit.**calculate_hash**(*path*, *input_to_hash*)

Hash the given input data with the specified audit device's hash function and salt.

This endpoint can be used to discover whether a given plaintext string (the input parameter) appears in the audit log in obfuscated form.

**Supported methods:** POST: /sys/audit-hash/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – The path of the audit device to generate hashes for. This is part of the request URL.
- **input_to_hash** (*str | unicode*) – The input string to hash.

**Returns** The JSON response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

input_to_hash = 'some sort of string thinger'

audit_hash = client.sys.calculate_hash(
    path='tmp-file-audit',
    input_to_hash=input_to_hash,
)

print('The hash for the provided input is: %s' % audit_hash['data']['hash'])
```

Example output:

```
The hash for the provided input is: hmac-sha256:...
```

## 2.3.2 Auth

### Examples

```
methods = client.sys.list_auth_methods()

client.sys.enable_auth_method('userpass', path='customuserpass')
client.sys.disable_auth_method('github')
```

### List Auth Methods

Auth.**list_auth_methods**()
List all enabled auth methods.

**Supported methods:** GET: /sys/auth. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

auth_methods = client.sys.list_auth_methods()
print('The following auth methods are enabled: {auth_methods_list}'.format(
    auth_methods_list=', '.join(auth_methods['data'].keys()),
))
```

Example output:

```
The following auth methods are enabled: token/
```

### Enable Auth Method

Auth.**enable_auth_method**(*method_type*, *description=None*, *config=None*, *plugin_name=None*, *local=False*, *path=None*, ***kwargs*)
Enable a new auth method.

After enabling, the auth method can be accessed and configured via the auth path specified as part of the URL. This auth path will be nested under the auth prefix.

**Supported methods:** POST: /sys/auth/{path}. Produces: 204 (empty body)

**Parameters**

- **method_type** (*str | unicode*) – The name of the authentication method type, such as "github" or "token".
- **description** (*str | unicode*) – A human-friendly description of the auth method.
- **config** (*dict*) – Configuration options for this auth method. These are the possible values:
  - **default_lease_ttl**: The default lease duration, specified as a string duration like "5s" or "30m".

- **max_lease_ttl**: The maximum lease duration, specified as a string duration like "5s" or "30m".
- **audit_non_hmac_request_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
- **audit_non_hmac_response_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
- **listing_visibility**: Speficies whether to show this mount in the UI-specific listing endpoint.
- **passthrough_request_headers**: Comma-separated list of headers to whitelist and pass from the request to the backend.

- **plugin_name** (*str | unicode*) – The name of the auth plugin to use based from the name in the plugin catalog. Applies only to plugin methods.
- **local** (*bool*) – <Vault enterprise only> Specifies if the auth method is a local only. Local auth methods are not replicated nor (if a secondary) removed by replication.
- **path** (*str | unicode*) – The path to mount the method on. If not provided, defaults to the value of the "method_type" argument.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.enable_auth_method(
    method_type='github',
    path='github-hvac',
)
```

### Disable Auth Method

Auth.**disable_auth_method**(*path*)

Disable the auth method at the given auth path.

**Supported methods:** DELETE: /sys/auth/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.disable_auth_method(
    path='github-hvac',
)
```

### Read Auth Method Tuning

Auth.**read_auth_method_tuning**(*path*)

Read the given auth path's configuration.

This endpoint requires sudo capability on the final path, but the same functionality can be achieved without sudo via sys/mounts/auth/[auth-path]/tune.

**Supported methods:** GET: /sys/auth/{path}/tune. Produces: 200 application/json

**Parameters** **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')
response = client.sys.read_auth_method_tuning(
    path='github-hvac',
)

print('The max lease TTL for the auth method under path "github-hvac" is: {max_ttl}'.
↪format(
    max_ttl=response['data']['max_lease_ttl'],
))
```

Example output:

```
The max lease TTL for the auth method under path "github-hvac" is: 2764800
```

### Tune Auth Method

Auth.**tune_auth_method**(*path*, *default_lease_ttl=None*, *max_lease_ttl=None*, *description=None*, *audit_non_hmac_request_keys=None*, *audit_non_hmac_response_keys=None*, *listing_visibility=None*, *passthrough_request_headers=None*, ***kwargs*)

Tune configuration parameters for a given auth path.

This endpoint requires sudo capability on the final path, but the same functionality can be achieved without sudo via sys/mounts/auth/[auth-path]/tune.

**Supported methods:** POST: /sys/auth/{path}/tune. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.
- **default_lease_ttl** (*int*) – Specifies the default time-to-live. If set on a specific auth path, this overrides the global default.
- **max_lease_ttl** (*int*) – The maximum time-to-live. If set on a specific auth path, this overrides the global default.
- **description** (*str | unicode*) – Specifies the description of the mount. This overrides the current stored value, if any.
- **audit_non_hmac_request_keys** (*array*) – Specifies the list of keys that will not be HMAC'd by audit devices in the request data object.
- **audit_non_hmac_response_keys** (*list*) – Specifies the list of keys that will not be HMAC'd by audit devices in the response data object.
- **listing_visibility** (*list*) – Specifies whether to show this mount in the UI-specific listing endpoint. Valid values are "unauth" or "".
- **passthrough_request_headers** (*list*) – List of headers to whitelist and pass from the request to the backend.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.tune_auth_method(
    path='github-hvac',
    description='The Github auth method for hvac users',
)
```

## 2.3.3 Health

- *Read Status*

### Read Status

Health.**read_health_status**(*standby_ok=None*, *active_code=None*, *standby_code=None*, *dr_secondary_code=None*, *performance_standby_code=None*, *sealed_code=None*, *uninit_code=None*, *method='HEAD'*)

Read the health status of Vault.

This matches the semantics of a Consul HTTP health check and provides a simple way to monitor the health of a Vault instance.

**Parameters**

- **standby_ok** (*bool*) – Specifies if being a standby should still return the active status code instead of the standby status code. This is useful when Vault is behind a non-configurable load balance that just wants a 200-level response.
- **active_code** (*int*) – The status code that should be returned for an active node.
- **standby_code** (*int*) – Specifies the status code that should be returned for a standby node.
- **dr_secondary_code** (*int*) – Specifies the status code that should be returned for a DR secondary node.
- **performance_standby_code** (*int*) – Specifies the status code that should be returned for a performance standby node.
- **sealed_code** (*int*) – Specifies the status code that should be returned for a sealed node.
- **uninit_code** (*int*) – Specifies the status code that should be returned for a uninitialized node.
- **method** (*str | unicode*) – Supported methods: HEAD: /sys/health. Produces: 000 (empty body) GET: /sys/health. Produces: 000 application/json

**Returns** The JSON response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

status = client.sys.read_health_status(method='GET')
print('Vault initialization status is: %s' % status['initialized'])
```

Example output:

```
Vault initialization status is: True
```

### 2.3.4 Init



#### Read Status

Init.**read_init_status**()
Read the initialization status of Vault.

**Supported methods:** GET: /sys/init. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

#### Examples

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')

read_response = client.sys.read_init_status()
print('Vault initialize status: %s' % read_response['initialized'])
```

Example output:

```
Vault initialize status: True
```

#### Is Initialized

Init.**is_initialized**()
Determine is Vault is initialized or not.

**Returns** True if Vault is initialized, False otherwise.

**Return type** bool

#### Examples

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')

print('Vault initialize status: %s' % client.sys.is_initialized())
```

Example output:

```
Vault initialize status: True
```

### Initialize

Init.**initialize**(*secret_shares=5*, *secret_threshold=3*, *pgp_keys=None*, *root_token_pgp_key=None*, *stored_shares=None*, *recovery_shares=None*, *recovery_threshold=None*, *recovery_pgp_keys=None*)

Initialize a new Vault.

The Vault must not have been previously initialized. The recovery options, as well as the stored shares option, are only available when using Vault HSM.

**Supported methods:** PUT: /sys/init. Produces: 200 application/json

**Parameters**

- **secret_shares** (*int*) – The number of shares to split the master key into.
- **secret_threshold** (*int*) – Specifies the number of shares required to reconstruct the master key. This must be less than or equal secret_shares. If using Vault HSM with auto-unsealing, this value must be the same as secret_shares.
- **pgp_keys** (*list*) – List of PGP public keys used to encrypt the output unseal keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as secret_shares.
- **root_token_pgp_key** (*str | unicode*) – Specifies a PGP public key used to encrypt the initial root token. The key must be base64-encoded from its original binary representation.
- **stored_shares** (*int*) – <enterprise only> Specifies the number of shares that should be encrypted by the HSM and stored for auto-unsealing. Currently must be the same as secret_shares.
- **recovery_shares** (*int*) – <enterprise only> Specifies the number of shares to split the recovery key into.
- **recovery_threshold** (*int*) – <enterprise only> Specifies the number of shares required to reconstruct the recovery key. This must be less than or equal to recovery_shares.
- **recovery_pgp_keys** (*list*) – <enterprise only> Specifies an array of PGP public keys used to encrypt the output recovery keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as recovery_shares.

**Returns** The JSON response of the request.

**Return type** dict

#### Examples

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')

init_result = client.sys.initialize()

root_token = init_result['root_token']
unseal_keys = init_result['keys']
```

When called for a previously initialized Vault cluster, an exception is raised:

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')

init_result = client.sys.initialize()
```

Example output:

```
Traceback (most recent call last):
  ...
hvac.exceptions.InvalidRequest: Vault is already initialized
```

### 2.3.5 Key

- *Read Root Generation Progress*
- *Start Root Token Generation*
- *Cancel Root Generation*
- *Generate Root*
- *Get Encryption Key Status*
- *Rotate Encryption Key*
- *Read Rekey Progress*
- *Start Rekey*
- *Cancel Rekey*
- *Rekey*
- *Rekey Multi*
- *Read Rekey Verify Progress*
- *Cancel Rekey Verify*
- *Rekey Verify*
- *Rekey Verify Multi*
- *Read Backup Keys*

### Read Root Generation Progress

Key.**read_root_generation_progress**()
Read the configuration and process of the current root generation attempt.

**Supported methods:** GET: /sys/generate-root/attempt. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

root_gen_progress = client.sys.read_root_generation_progress()
print('Root generation "started" status: %s' % root_gen_progress['started'])
```

Example output:

```
Root generation "started" status: ...
```

### Start Root Token Generation

Key.**start_root_token_generation**(*otp=None*, *pgp_key=None*)
Initialize a new root generation attempt.

Only a single root generation attempt can take place at a time. One (and only one) of otp or pgp_key are required.

**Supported methods:** PUT: /sys/generate-root/attempt. Produces: 200 application/json

**Parameters**

- **otp** (*str | unicode*) – Specifies a base64-encoded 16-byte value. The raw bytes of the token will be XOR'd with this value before being returned to the final unseal key provider.
- **pgp_key** (*str | unicode*) – Specifies a base64-encoded PGP public key. The raw bytes of the token will be encrypted with this value before being returned to the final unseal key provider.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
from tests.utils import get_generate_root_otp

client = hvac.Client(url='https://127.0.0.1:8200')

new_otp = get_generate_root_otp()
start_generate_root_response = client.sys.start_root_token_generation(
    otp=new_otp,
```

(continues on next page)

(continued from previous page)

```
)
nonce = start_generate_root_response['nonce']
print('Nonce for root generation is: %s' % nonce)
```

Example output:

```
Nonce for root generation is: ...
```

### Cancel Root Generation

Key.**cancel_root_generation**()
Cancel any in-progress root generation attempt.

This clears any progress made. This must be called to change the OTP or PGP key being used.

**Supported methods:** DELETE: /sys/generate-root/attempt. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** request.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.cancel_root_generation()
```

### Generate Root

Key.**generate_root**(*key*, *nonce*)
Enter a single master key share to progress the root generation attempt.

If the threshold number of master key shares is reached, Vault will complete the root generation and issue the new token. Otherwise, this API must be called multiple times until that threshold is met. The attempt nonce must be provided with each call.

**Supported methods:** PUT: /sys/generate-root/update. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single master key share.
- **nonce** (*str | unicode*) – The nonce of the attempt.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.generate_root(
    key=key,
    nonce=nonce,
)
```

### Get Encryption Key Status

Client.**key_status**
GET /sys/key-status

**Returns** Information about the current encryption key used by Vault.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

print('Encryption key term is: %s' % client.key_status['term'])
```

Example output:

```
Encryption key term is: 1
```

### Rotate Encryption Key

Key.**rotate_encryption_key**()
Trigger a rotation of the backend encryption key.

This is the key that is used to encrypt data written to the storage backend, and is not provided to operators. This operation is done online. Future values are encrypted with the new key, while old values are decrypted with previous encryption keys.

This path requires sudo capability in addition to update.

**Supported methods:** PUT: /sys/rorate. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.rotate_encryption_key()
```

### Read Rekey Progress

Key.**read_rekey_progress**(*recovery_key=False*)

Read the configuration and progress of the current rekey attempt.

**Supported methods:** GET: /sys/rekey-recovery-key/init. Produces: 200 application/json GET: /sys/rekey/init. Produces: 200 application/json

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

print('Rekey "started" status is: %s' % client.sys.read_rekey_progress()['started'])
```

Example output:

```
Rekey "started" status is: False
```

### Start Rekey

Key.**start_rekey**(*secret_shares=5*, *secret_threshold=3*, *pgp_keys=None*, *backup=False*, *require_verification=False*, *recovery_key=False*)

Initializes a new rekey attempt.

Only a single recovery key rekeyattempt can take place at a time, and changing the parameters of a rekey requires canceling and starting a new rekey, which will also provide a new nonce.

**Supported methods:** PUT: /sys/rekey/init. Produces: 204 (empty body) PUT: /sys/rekey-recovery-key/init. Produces: 204 (empty body)

**Parameters**

- **secret_shares** (*int*) – Specifies the number of shares to split the master key into.
- **secret_threshold** (*int*) – Specifies the number of shares required to reconstruct the master key. This must be less than or equal to secret_shares.
- **pgp_keys** (*list*) – Specifies an array of PGP public keys used to encrypt the output unseal keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as secret_shares.

- **backup** (*bool*) – Specifies if using PGP-encrypted keys, whether Vault should also store a plaintext backup of the PGP-encrypted keys at core/unseal-keys-backup in the physical storage backend. These can then be retrieved and removed via the sys/rekey/backup endpoint.
- **require_verification** (*bool*) – This turns on verification functionality. When verification is turned on, after successful authorization with the current unseal keys, the new unseal keys are returned but the master key is not actually rotated. The new keys must be provided to authorize the actual rotation of the master key. This ensures that the new keys have been successfully saved and protects against a risk of the keys being lost after rotation but before they can be persisted. This can be used with without pgp_keys, and when used with it, it allows ensuring that the returned keys can be successfully decrypted before committing to the new shares, which the backup functionality does not provide.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON dict of the response.

**Return type** dict | request.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

rekey_response = client.sys.start_rekey()
nonce = rekey_response['nonce']
print('Nonce for rekey is: %s' % nonce)
```

Example output:

```
Nonce for rekey is: ...
```

### Cancel Rekey

Key.**cancel_rekey**(*recovery_key=False*)

Cancel any in-progress rekey.

This clears the rekey settings as well as any progress made. This must be called to change the parameters of the rekey.

Note: Verification is still a part of a rekey. If rekeying is canceled during the verification flow, the current unseal keys remain valid.

**Supported methods:** DELETE: /sys/rekey/init. Produces: 204 (empty body) DELETE: /sys/rekey-recovery-key/init. Produces: 204 (empty body)

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.cancel_rekey()
```

### Rekey

Key.**rekey**(*key*, *nonce=None*, *recovery_key=False*)

Enter a single recovery key share to progress the rekey of the Vault.

If the threshold number of recovery key shares is reached, Vault will complete the rekey. Otherwise, this API must be called multiple times until that threshold is met. The rekey nonce operation must be provided with each call.

**Supported methods:** PUT: /sys/rekey/update. Produces: 200 application/json PUT: /sys/rekey-recovery-key/update. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single recovery share key.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey operation.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.rekey(
    key=key,
    nonce=nonce,
)
```

### Rekey Multi

Key.**rekey_multi**(*keys*, *nonce=None*, *recovery_key=False*)

Enter multiple recovery key shares to progress the rekey of the Vault.

If the threshold number of recovery key shares is reached, Vault will complete the rekey.

**Parameters**

- **keys** (*list*) – Specifies multiple recovery share keys.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey operation.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The last response of the rekey request.

**Return type** response.Request

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.rekey_multi(
    keys,
    nonce=nonce,
)
```

### Read Rekey Verify Progress

Key.**read_rekey_verify_progress**()

Read the configuration and progress of the current rekey verify attempt.

**Supported methods:** GET: /sys/rekey/verify. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

response = client.sys.read_rekey_verify_progress()

print(
    'Rekey verify progress is %d out of %d' % (
        response['progress'],
        response['t'],
    )
)
```

Example output:

```
Rekey verify progress is 0 out of 3
```

### Cancel Rekey Verify

Key.**cancel_rekey_verify**()

Cancel any in-progress rekey verification. This clears any progress made and resets the nonce. Unlike cancel_rekey, this only resets the current verification operation, not the entire rekey atttempt. The return value is the same as GET along with the new nonce.

**Supported methods:** DELETE: /sys/rekey/verify. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.cancel_rekey_verify()
```

### Rekey Verify

Key.**rekey_verify**(*key*, *nonce*)

Enter a single new recovery key share to progress the rekey verification of the Vault. If the threshold number of new recovery key shares is reached, Vault will complete the rekey. Otherwise, this API must be called multiple times until that threshold is met. The rekey verification nonce must be provided with each call.

**Supported methods:** PUT: /sys/rekey/verify. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies multiple recovery share keys.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey verify operation.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.rekey_verify(
    key,
    nonce=verify_nonce,
)
```

### Rekey Verify Multi

Key.**rekey_verify_multi**(*keys*, *nonce*)

Enter multiple new recovery key shares to progress the rekey verification of the Vault. If the threshold number of new recovery key shares is reached, Vault will complete the rekey. Otherwise, this API must be called multiple times until that threshold is met. The rekey verification nonce must be provided with each call.

**Supported methods:** PUT: /sys/rekey/verify. Produces: 200 application/json

**Parameters**

- **keys** (*list*) – Specifies multiple recovery share keys.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey verify operation.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.rekey_verify_multi(
    keys,
    nonce=verify_nonce,
)
```

### Read Backup Keys

Key.**read_backup_keys**(*recovery_key=False*)

Retrieve the backup copy of PGP-encrypted unseal keys.

The returned value is the nonce of the rekey operation and a map of PGP key fingerprint to hex-encoded PGP-encrypted key.

**Supported methods:** PUT: /sys/rekey/backup. Produces: 200 application/json PUT: /sys/rekey-recovery-key/backup. Produces: 200 application/json

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** dict

#### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')
rekey_response = client.sys.start_rekey(
    secret_shares=1,
    secret_threshold=1,
    pgp_keys=pgp_keys,
    backup=True,
)
nonce = rekey_response['nonce']

client.sys.rekey_multi(
    keys,
    nonce=nonce,
)

print('Backup keys are: %s' % client.sys.read_backup_keys()['data']['keys'])
```

Example output:

```
Backup keys are: {'...': [...]}
```

### 2.3.6 Leader



#### Read Leader Status

Leader.**read_leader_status**()

Read the high availability status and current leader instance of Vault.

**Supported methods:** GET: /sys/leader. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

#### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

status = client.sys.read_leader_status()
print('HA status is: %s' % status['ha_enabled'])
```

Example output:

```
HA status is: False
```

#### Step Down

Leader.**step_down**()

Force the node to give up active status.

If the node does not have active status, this endpoint does nothing. Note that the node will sleep for ten seconds before attempting to grab the active lock again, but if no standby nodes grab the active lock in the interim, the same node may become the active node again. Requires a token with root policy or sudo capability on the path.

**Returns** The JSON response of the request.

**Return type** dict

#### Examples

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')
client.sys.step_down()
```

### 2.3.7 Lease



#### View and Manage Leases

Read a lease:

```
>>> read_lease_response = client.sys.read_lease(lease_id=lease_id)
>>> print('Expire time for lease ID {id} is: {expire_time}'.format(
...     id=lease_id,
...     expire_time=read_lease_response['data']['expire_time'],
... ))
Expire time for lease ID pki/issue/my-role/... is: 20...
```

Renewing a lease:

```
>>> renew_lease_resp = client.sys.renew_lease(lease_id=lease_id)
>>> print('Lease ID: "{id}" renewed, lease duration: "{duration}"'.format(
...     id=renew_lease_resp['lease_id'],
...     duration=renew_lease_resp['lease_duration'],
... ))
Lease ID: "pki/issue/my-role/d05138a2-edeb-889d-db98-2057ecd5138f" renewed, lease␣
↪duration: "2764790"
```

Revoking a lease:

```
>>> client.sys.revoke_lease(lease_id=lease_id)
<Response [204]>
```

### Read Lease

Lease.**read_lease**(*lease_id*)

Retrieve lease metadata.

**Supported methods:** PUT: /sys/leases/lookup. Produces: 200 application/json

**Parameters** **lease_id** (*str | unicode*) – the ID of the lease to lookup.

**Returns** Parsed JSON response from the leases PUT request

**Return type** dict.

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

read_lease_resp = client.sys.read_lease(
    lease_id=lease_id,
)

# expire_time in the form of something like: 2019-02-25T07:41:30.000038-06:00
print('Current expire time for lease ID {id} is: {expires}'.format(
    id=lease_id,
    expires=read_lease_resp['data']['expire_time'],
))
```

Example output:

```
Current expire time for lease ID pki/issue/my-role/... is: ...
```

### List Leases

Lease.**list_leases**(*prefix*)
Retrieve a list of lease ids.

**Supported methods:** LIST: /sys/leases/lookup/{prefix}. Produces: 200 application/json

**Parameters** **prefix** (*str | unicode*) – Lease prefix to filter list by.

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

list_leases_response = client.sys.list_leases(
    prefix='pki',
)
print('The follow lease keys are active under the "pki" prefix: %s' % list_leases_
↪response['data']['keys'])
```

Example output:

```
The follow lease keys are active under the "pki" prefix: ['issue/']
```

### Renew Lease

Lease.**renew_lease**(*lease_id*, *increment=None*)
Renew a lease, requesting to extend the lease.

**Supported methods:** PUT: /sys/leases/renew. Produces: 200 application/json

**Parameters**

- **lease_id** (*str | unicode*) – The ID of the lease to extend.
- **increment** (*int*) – The requested amount of time (in seconds) to extend the lease.

**Returns** The JSON response of the request

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.renew_lease(
    lease_id=lease_id,
    increment=500,
)
```

### Revoke Lease

Lease.**revoke_lease**(*lease_id*)
Revoke a lease immediately.

**Supported methods:** PUT: /sys/leases/revoke. Produces: 204 (empty body)

**Parameters** **lease_id** (*str | unicode*) – Specifies the ID of the lease to revoke.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.revoke_lease(
    lease_id=lease_id,
)
```

### Revoke Prefix

Lease.**revoke_prefix**(*prefix*)
Revoke all secrets (via a lease ID prefix) or tokens (via the tokens' path property) generated under a given prefix immediately.

This requires sudo capability and access to it should be tightly controlled as it can be used to revoke very large numbers of secrets/tokens at once.

**Supported methods:** PUT: /sys/leases/revoke-prefix/{prefix}. Produces: 204 (empty body)

**Parameters** **prefix** (*str | unicode*) – The prefix to revoke.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.revoke_prefix(
    prefix='pki',
)
```

### Revoke Force

Lease.**revoke_force**(*prefix*)

Revoke all secrets or tokens generated under a given prefix immediately.

Unlike revoke_prefix, this path ignores backend errors encountered during revocation. This is potentially very dangerous and should only be used in specific emergency situations where errors in the backend or the connected backend service prevent normal revocation.

**Supported methods:** PUT: /sys/leases/revoke-force/{prefix}. Produces: 204 (empty body)

**Parameters** **prefix** (*str | unicode*) – The prefix to revoke.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.revoke_force(
    prefix='pki',
)
```

## 2.3.8 Mount



### Manipulate secret backends

```
backends = client.sys.list_mounted_secrets_engines()['data']

client.sys.enable_secrets_engine('aws', path='aws-us-east-1')
client.sys.disable_secrets_engine('mysql')

client.sys.tune_mount_configuration(path='test', default_lease_ttl='3600s', max_lease_
↪ttl='8600s')
client.sys.read_mount_configuration(path='test')

client.sys.move_backend('aws-us-east-1', 'aws-east')
```

### List Mounted Secrets Engines

Mount.**list_mounted_secrets_engines**()
Lists all the mounted secrets engines.

**Supported methods:** POST: /sys/mounts. Produces: 200 application/json

**Returns** JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

secrets_engines_list = client.sys.list_mounted_secrets_engines()['data']
print('The following secrets engines are mounted: %s' % ', '.join(sorted(secrets_
↪engines_list.keys())))
```

Example output:

```
The following secrets engines are mounted: cubbyhole/, identity/, secret/, sys/
```

### Enable Secrets Engine

Mount.**enable_secrets_engine**(*backend_type*, *path=None*, *description=None*, *config=None*, *plugin_name=None*, *options=None*, *local=False*, *seal_wrap=False*, ***kwargs*)
Enable a new secrets engine at the given path.

**Supported methods:** POST: /sys/mounts/{path}. Produces: 204 (empty body)

**Parameters**

- **backend_type** (*str | unicode*) – The name of the backend type, such as "github" or "token".
- **path** (*str | unicode*) – The path to mount the method on. If not provided, defaults to the value of the "backend_type" argument.
- **description** (*str | unicode*) – A human-friendly description of the mount.
- **config** (*dict*) – Configuration options for this mount. These are the possible values:
  - **default_lease_ttl**: The default lease duration, specified as a string duration like "5s" or "30m".
  - **max_lease_ttl**: The maximum lease duration, specified as a string duration like "5s" or "30m".
  - **force_no_cache**: Disable caching.
  - **plugin_name**: The name of the plugin in the plugin catalog to use.
  - **audit_non_hmac_request_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.

  - **audit_non_hmac_response_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
  - **listing_visibility**: Specifies whether to show this mount in the UI-specific listing endpoint. ("unauth" or "hidden")
  - **passthrough_request_headers**: Comma-separated list of headers to whitelist and pass from the request to the backend.
- **options** (*dict*) – Specifies mount type specific options that are passed to the backend.
  - **version**: <KV> The version of the KV to mount. Set to "2" for mount KV v2.
- **plugin_name** (*str | unicode*) – Specifies the name of the plugin to use based from the name in the plugin catalog. Applies only to plugin backends.
- **local** (*bool*) – <Vault enterprise only> Specifies if the auth method is a local only. Local auth methods are not replicated nor (if a secondary) removed by replication.
- **seal_wrap** (*bool*) – <Vault enterprise only> Enable seal wrapping for the mount.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.enable_secrets_engine(
    backend_type='kv',
    path='hvac-kv',
)
```

### Disable Secrets Engine

Mount.**disable_secrets_engine**(*path*)

Disable the mount point specified by the provided path.

**Supported methods:** DELETE: /sys/mounts/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.disable_secrets_engine(
    path='hvac-kv',
)
```

### Read Mount Configuration

Mount.**read_mount_configuration**(*path*)

Read the given mount's configuration.

Unlike the mounts endpoint, this will return the current time in seconds for each TTL, which may be the system default or a mount-specific value.

**Supported methods:** GET: /sys/mounts/{path}/tune. Produces: 200 application/json

**Parameters** **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.

**Returns** The JSON response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

secret_backend_tuning = client.sys.read_mount_configuration(path='hvac-kv')
print('The max lease TTL for the "hvac-kv" backend is: {max_lease_ttl}'.format(
    max_lease_ttl=secret_backend_tuning['data']['max_lease_ttl'],
 ))
```

Example output:

```
The max lease TTL for the "hvac-kv" backend is: 2764800
```

### Tune Mount Configuration

Mount.**tune_mount_configuration**(*path*, *default_lease_ttl=None*, *max_lease_ttl=None*, *description=None*, *audit_non_hmac_request_keys=None*, *audit_non_hmac_response_keys=None*, *listing_visibility=None*, *passthrough_request_headers=None*, *options=None*, *force_no_cache=None*, ***kwargs*)

Tune configuration parameters for a given mount point.

**Supported methods:** POST: /sys/mounts/{path}/tune. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.
- **mount_point** (*str*) – The path the associated secret backend is mounted
- **description** (*str*) – Specifies the description of the mount. This overrides the current stored value, if any.
- **default_lease_ttl** (*int*) – Default time-to-live. This overrides the global default. A value of 0 is equivalent to the system default TTL
- **max_lease_ttl** (*int*) – Maximum time-to-live. This overrides the global default. A value of 0 are equivalent and set to the system max TTL.
- **audit_non_hmac_request_keys** (*list*) – Specifies the comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
- **audit_non_hmac_response_keys** (*list*) – Specifies the comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
- **listing_visibility** (*str*) – Speficies whether to show this mount in the UI-specific listing endpoint. Valid values are "unauth" or "".
- **passthrough_request_headers** (*str*) – Comma-separated list of headers to whitelist and pass from the request to the backend.
- **options** (*dict*) – Specifies mount type specific options that are passed to the backend.
  - **version**: <KV> The version of the KV to mount. Set to "2" for mount KV v2.
- **force_no_cache** (*bool*) – Disable caching.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response from the request.

**Return type** request.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.tune_mount_configuration(
    path='hvac-kv',
    default_lease_ttl='3600s',
    max_lease_ttl='8600s',
)
```

#### Move Backend

Mount.**move_backend**(*from_path*, *to_path*)
Move an already-mounted backend to a new mount point.

**Supported methods:** POST: /sys/remount. Produces: 204 (empty body)

**Parameters**

- **from_path** (*str | unicode*) – Specifies the previous mount point.
- **to_path** (*str | unicode*) – Specifies the new destination mount point.

**Returns** The response of the request.

**Return type** requests.Response

#### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.move_backend(
    from_path='hvac-kv',
    to_path='kv-hvac',
)
```

### 2.3.9 Namespace



#### Create Namespace

Namespace.**create_namespace**(*path*)
Create a namespace at the given path.

**Supported methods:** POST: /sys/namespaces/{path}. Produces: 200 application/json

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

# Create namespace team1 where team1 is a child of root
client.sys.create_namespace(path="team1")

# Create namespace team1/app1 where app1 is a child of team1
client2 = hvac.Client(url='https://127.0.0.1:8200', namespace="team1")
client2.sys.create_namespace(path="app1")
```

Example output:

print(client.sys.create_namespace(path="team1")) {"request_id":"<redacted>","lease_id":"","renewable":false,"lease_duration":0

print(client2.sys.create_namespace(path="app1")) {"request_id":"<redacted>","lease_id":"","renewable":false,"lease_duration":0

### List Namespaces

Namespace.**list_namespaces**()

Lists all the namespaces.

**Supported methods:** LIST: /sys/namespaces. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')
client.sys.create_namespace(path='testns')

client.sys.list_namespaces()
```

Example output:

print(client.sys.list_namespaces()) {"request_id":"<redacted>","lease_id":"","renewable":false,"lease_duration":0,"data":{"key_i

### Delete Namespace

Namespace.**delete_namespace**(*path*)

Delete a namespaces. You cannot delete a namespace with existing child namespaces.

**Supported methods:** DELETE: /sys/namespaces. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

#### Examples

```
import hvac

# Delete namespace app1 where app1 is a child of team1
client2 = hvac.Client(url='https://127.0.0.1:8200', namespace="team1")
client2.sys.delete_namespace(path="app1")

# Delete namespace team1
client = hvac.Client(url='https://127.0.0.1:8200')
client.sys.delete_namespace(path="team1")
```

### 2.3.10 Policy



#### Manipulate policies

```
policies = client.sys.list_policies()['data']['policies'] # => ['root']

policy = """
path "sys" {
  capabilities = ["deny"]
}

path "secret/*" {
  capabilities = ["read", "list"]
}

path "secret/foo" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
"""

client.sys.create_or_update_policy(
    name='secret-writer',
    policy=policy,
)

client.sys.delete_policy('oldthing')

# The get_policy method offers some additional features and is available in the␣
↪Client class.
policy = client.get_policy('mypolicy')
```

(continues on next page)

(continued from previous page)

```
# Requires pyhcl to automatically parse HCL into a Python dictionary
policy = client.get_policy('mypolicy', parse=True)
```

### Using Python Variable(s) In Policy Rules

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')

key = 'some-key-string'

policy_body = """
path "transit/encrypt/%s" {
    capabilities = ["update"]
}
""" % key
client.sys.create_or_update_policy(
    name='my-policy-name',
    policy=policy_body,
)
```

### List Policies

Policy.**list_policies**()

List all configured policies.

**Supported methods:** GET: /sys/policy. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

list_policies_resp = client.sys.list_policies()['data']['policies']
print('List of currently configured policies: %s' % ', '.join(list_policies_resp))
```

Example output:

```
List of currently configured policies: default, my-policy-name, secret-writer, root
```

### Read Policy

Policy.**read_policy**(*name*)

Retrieve the policy body for the named policy.

**Supported methods:** GET: /sys/policy/{name}. Produces: 200 application/json

**Parameters** **name** (*str | unicode*) – The name of the policy to retrieve.

**Returns** The response of the request

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

hvac_policy_rules = client.sys.read_policy(name='secret-writer')['data']['rules']
print('secret-writer policy rules:\n%s' % hvac_policy_rules)
```

Example output:

```
secret-writer policy rules:

path "sys" {
  capabilities = ["deny"]
}

path "secret/*" {
  capabilities = ["read", "list"]
}

path "secret/foo" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
...
```

### Create Or Update Policy

Policy.**create_or_update_policy**(*name*, *policy*, *pretty_print=True*)

Add a new or update an existing policy.

Once a policy is updated, it takes effect immediately to all associated users.

**Supported methods:** PUT: /sys/policy/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the policy to create.
- **policy** (*str | unicode | dict*) – Specifies the policy document.
- **pretty_print** (*bool*) – If True, and provided a dict for the policy argument, send the policy JSON to Vault with "pretty" formatting.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

policy = '''
    path "sys" {
        capabilities = ["deny"]
    }
    path "secret" {
        capabilities = ["create", "read", "update", "delete", "list"]
    }
'''
client.sys.create_or_update_policy(
    name='secret-writer',
    policy=policy,
)
```

### Delete Policy

Policy.**delete_policy**(*name*)

Delete the policy with the given name.

This will immediately affect all users associated with this policy.

**Supported methods:** DELETE: /sys/policy/{name}. Produces: 204 (empty body)

**Parameters** **name** (*str | unicode*) – Specifies the name of the policy to delete.

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.delete_policy(
    name='secret-writer',
)
```

### 2.3.11 Raft

*hvac.api.system_backend.Raft()*



#### Join Raft Cluster

*hvac.api.system_backend.Raft.join_raft_cluster()*

```
import hvac
client = hvac.Client()

client.sys.join_raft_cluster(
    leader_api_addr='https://some-vault-node',
)
```

#### Read Raft Configuration

*hvac.api.system_backend.Raft.read_raft_config()*

```
import hvac
client = hvac.Client()

raft_config = c.sys.read_raft_config()
num_servers_in_cluster = len(raft_config['data']['config']['servers'])
```

#### Remove Raft Node

*hvac.api.system_backend.Raft.remove_raft_node()*

```
import hvac
client = hvac.Client()

client.sys.remove_raft_node(
    server_id='i-somenodeid',
)
```

### 2.3.12 Seal



#### Seal Status

`Client.seal_status`
Read the seal status of the Vault.

This is an unauthenticated endpoint.

**Supported methods:** GET: /sys/seal-status. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

#### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

print('Is Vault sealed: %s' % client.seal_status['sealed'])
```

Example output:

```
Is Vault sealed: False
```

#### Is Sealed

`Seal.is_sealed()`
Determine if Vault is sealed.

**Returns** True if Vault is seal, False otherwise.

**Return type** bool

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

print('Is Vault sealed: %s' % client.sys.is_sealed())
```

Example output:

```
Is Vault sealed: False
```

### Read Seal Status

Seal.**read_seal_status**()
Read the seal status of the Vault.

This is an unauthenticated endpoint.

**Supported methods:** GET: /sys/seal-status. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

print('Is Vault sealed: %s' % client.sys.read_seal_status()['sealed'])
```

Example output:

```
Is Vault sealed: False
```

### Seal

Seal.**seal**()
Seal the Vault.

In HA mode, only an active node can be sealed. Standby nodes should be restarted to get the same effect. Requires a token with root policy or sudo capability on the path.

**Supported methods:** PUT: /sys/seal. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.seal()
```

### Submit Unseal Key

Seal.**submit_unseal_key**(*key=None*, *reset=False*, *migrate=False*)

Enter a single master key share to progress the unsealing of the Vault.

If the threshold number of master key shares is reached, Vault will attempt to unseal the Vault. Otherwise, this API must be called multiple times until that threshold is met.

Either the key or reset parameter must be provided; if both are provided, reset takes precedence.

**Supported methods:** PUT: /sys/unseal. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single master key share. This is required unless reset is true.
- **reset** (*bool*) – Specifies if previously-provided unseal keys are discarded and the unseal process is reset.
- **migrate** – Available in 1.0 Beta - Used to migrate the seal from shamir to autoseal or autoseal to shamir. Must be provided on all unseal key calls.

**Type** migrate: bool

**Returns** The JSON response of the request.

**Return type** dict

### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.submit_unseal_key(key=key)
```

### Submit Unseal Keys

Seal.**submit_unseal_keys**(*keys*, *migrate=False*)

Enter multiple master key share to progress the unsealing of the Vault.

**Parameters**

- **keys** (*List[str]*) – List of master key shares.
- **migrate** – Available in 1.0 Beta - Used to migrate the seal from shamir to autoseal or autoseal to shamir. Must be provided on all unseal key calls.

**Type** migrate: bool

**Returns** The JSON response of the last unseal request.

**Return type** dict

#### Examples

```
import hvac
client = hvac.Client(url='https://127.0.0.1:8200')

client.sys.submit_unseal_keys(keys=keys)
```

### 2.3.13 Wrapping

- *Unwrap*
- *Is Sealed*

#### Unwrap

#### Is Sealed

Seal.**is_sealed**()
Determine if Vault is sealed.

**Returns** True if Vault is seal, False otherwise.

**Return type** bool

#### Examples

```
import hvac

client = hvac.Client(url='https://127.0.0.1:8200')
client.write(
    path="auth/approle-test/role/testrole",
)
result = client.write(
    path='auth/approle-test/role/testrole/secret-id',
    wrap_ttl="10s",
)

unwrap_response = client.sys.unwrap(
    token=result['wrap_info']['token'],
)
print('Unwrapped approle role token secret id accessor: "%s"' % unwrap_response['data
↪']['secret_id_accessor'])
```

Example output:

```
Unwrapped approle role token secret id accessor: "..."
```

## 2.4 Initialize and seal/unseal

```
print(client.sys.is_initialized()) # => False

shares = 5
threshold = 3

result = client.sys.initialize(shares, threshold)

root_token = result['root_token']
keys = result['keys']

print(client.sys.is_initialized()) # => True

print(client.sys.is_sealed()) # => True

# unseal with individual keys
client.sys.unseal(keys[0])
client.sys.unseal(keys[1])
client.sys.unseal(keys[2])

# unseal with multiple keys until threshold met
client.sys.unseal_multi(keys)

print(client.sys.is_sealed()) # => False

client.sys.seal()

print(client.sys.is_sealed()) # => True
```

CHAPTER

THREE

# ADVANCED USAGE



## 3.1 Making Use of Private CA

There is a not uncommon use case of people deploying Hashicorp Vault with a private certificate authority. Unfortunately the *requests* module does not make use of the system CA certificates. Instead of disabling SSL verification you can make use of the requests' *verify* parameter.

As documented in the advanced usage section for requests this variable can point to a file that is comprised of all CA certificates you may wish to use. This can be a single private CA, or an existing list of root certificates with the private appended to the end. The following example shows how to achieve this:

```
$ cp "$(python -c 'import certifi;print certifi.where();')" /tmp/bundle.pem
$ cat /path/to/custom.pem >> /tmp/bundle.pem
```

You then use hvac's Client.session and requests.Session() to pass the new CA bundle to hvac.

```
import os

import hvac
import requests


def get_vault_client(vault_url=VAULT_URL, certs=VAULT_CERTS):
        """
        Instantiates a hvac / vault client.
        :param vault_url: string, protocol + address + port for the vault service
        :param certs: tuple, Optional tuple of self-signed certs to use for␣
↪verification
                with hvac's requests adapater.
        :return: hvac.Client
        """
        logger.debug('Retrieving a vault (hvac) client...')
        vault_client = hvac.Client(
                url=vault_url,
                cert=certs,
```

(continues on next page)

(continued from previous page)

```
        )
        if certs:
        # When use a self-signed certificate for the vault service itself, we need to
        # include our local ca bundle here for the underlying requests module.
                rs = requests.Session()
                vault_client.session = rs
                rs.verify = certs

        vault_client.token = load_vault_token(vault_client)

        if not vault_client.is_authenticated():
                error_msg = 'Unable to authenticate to the Vault service'
                raise hvac.exceptions.Unauthorized(error_msg)

        return vault_client
```

## 3.2 Custom Requests / HTTP Adapter

New in version 0.6.2.

Calls to the requests module. (which provides the methods hvac utilizes to send HTTP/HTTPS request to Vault instances) were extracted from the *Client* class and moved to a newly added *hvac.adapters()* module. The *Client* class itself defaults to an instance of the *JSONAdapter* class for its _adapter private attribute attribute if no adapter argument is provided to its *constructor*. This attribute provides an avenue for modifying the manner in which hvac completes request. To enable this type of customization, implement a class of type *hvac.adapters.Adapter()*, override its abstract methods, and pass this custom class to the adapter argument of the *Client constructor*

## 3.3 Vault Agent Unix Socket Listener

hvac does not currently offer direct support of requests to a Vault agent process configured with a unix socket listener. However this use case can be handled with the help of the requests_unixsocket module. To accomplish this, first ensure the module is available (e.g. *pip install requests_unixsocket*), and then instantiate the *Client* class in the following manner:

```
import urllib.parse

import requests_unixsocket
import hvac

vault_agent_socket_path = '/var/run/vault/agent.sock'
socket_url = 'http+unix://{encoded_path}'.format(
        encoded_path=urllib.parse.quote(vault_agent_socket_path, safe='')
)
socket_session = requests_unixsocket.Session()
client = hvac.Client(
        url=socket_url,
        session=socket_session,
)
print(client.secrets.kv.read_secret_version(path='some-secret'))
```

CHAPTER

FOUR

# SOURCE REFERENCE

## 4.1 hvac.v1

**Classes**

| | |
|---|---|
| *Client*([url, token, cert, verify, timeout, ...]) | The hvac Client class for HashiCorp's Vault. |

**class** hvac.v1.**Client**(*url=None*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *adapter=<class 'hvac.adapters.JSONAdapter'>*, *namespace=None*, ***kwargs*)

Bases: object

The hvac Client class for HashiCorp's Vault. **Methods**

| | |
|---|---|
| *__init__*([url, token, cert, verify, ...]) | Creates a new hvac client instance. |
| *audit_hash*(name, input) | Call to deprecated function 'audit_hash'. |
| *auth_app_id*(app_id, user_id[, mount_point, ...]) | POST /auth/<mount point>/login |
| *auth_approle*(role_id[, secret_id, ...]) | Call to deprecated function 'auth_approle'. |
| *auth_aws_iam*(access_key, secret_key[, ...]) | Call to deprecated function 'auth_aws_iam'. |
| *auth_cubbyhole*(token) | Perform a login request with a wrapped token. |
| *auth_ec2*(pkcs7[, nonce, role, use_token, ...]) | Call to deprecated function 'auth_ec2'. |
| *auth_gcp*(*args, **kwargs) | Call to deprecated function 'auth_gcp'. |
| *auth_github*(*args, **kwargs) | Call to deprecated function 'auth_github'. |
| *auth_kubernetes*(role, jwt[, use_token, ...]) | POST /auth/<mount_point>/login |
| *auth_ldap*(*args, **kwargs) | Call to deprecated function 'auth_ldap'. |
| *auth_tls*([mount_point, use_token]) | Call to deprecated function 'auth_tls'. |
| *auth_userpass*(username, password[, ...]) | POST /auth/<mount point>/login/<username> |
| *cancel_generate_root*() | Call to deprecated function 'cancel_generate_root'. |
| *cancel_rekey*() | Call to deprecated function 'cancel_rekey'. |
| *close*() | Call to deprecated function 'close'. |
| *create_app_id*(app_id, policies[, ...]) | POST /auth/<mount point>/map/app-id/<app_id> |
| *create_ec2_role*(role[, bound_ami_id, ...]) | Call to deprecated function 'create_ec2_role'. |
| *create_ec2_role_tag*(role[, policies, ...]) | Call to deprecated function 'create_ec2_role_tag'. |
| *create_kubernetes_configuration*(kubernetes_host[, ...]) | POST /auth/<mount_point>/config |
| *create_kubernetes_role*(name, ...[, ttl, ...]) | POST /auth/<mount_point>/role/:name |
| *create_role*(role_name[, mount_point]) | Call to deprecated function 'create_role'. |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| create_role_custom_secret_id(role_name, ...) | Call to deprecated function 'create_role_custom_secret_id'. |
| create_role_secret_id(role_name[, meta, ...]) | Call to deprecated function 'create_role_secret_id'. |
| create_token([role, token_id, policies, ...]) | POST /auth/token/create |
| create_token_role(role[, allowed_policies, ...]) | POST /auth/token/roles/<role> |
| create_user_id(user_id, app_id[, ...]) | POST /auth/<mount point>/map/user-id/<user_id> |
| create_userpass(username, password, policies) | POST /auth/<mount point>/users/<username> |
| create_vault_ec2_certificate_configuration(...) | Call to deprecated function 'create_vault_ec2_certificate_configuration'. |
| create_vault_ec2_client_configuration([, ...]) | Call to deprecated function 'create_vault_ec2_client_configuration'. |
| delete(path) | DELETE /<path> |
| delete_app_id(app_id[, mount_point]) | DELETE /auth/<mount_point>/map/app-id/<app_id> |
| delete_ec2_role(role[, mount_point]) | Call to deprecated function 'delete_ec2_role'. |
| delete_kubernetes_role(role[, mount_point]) | DELETE /auth/<mount_point>/role/:role |
| delete_policy(name) | Call to deprecated function 'delete_policy'. |
| delete_role(role_name[, mount_point]) | Call to deprecated function 'delete_role'. |
| delete_role_secret_id(role_name, secret_id) | Call to deprecated function 'delete_role_secret_id'. |
| delete_role_secret_id_accessor(role_name, ...) | Call to deprecated function 'delete_role_secret_id_accessor'. |
| delete_token_role(role) | Deletes the named token role. |
| delete_user_id(user_id[, mount_point]) | DELETE /auth/<mount_point>/map/user-id/<user_id> |
| delete_userpass(username[, mount_point]) | DELETE /auth/<mount point>/users/<username> |
| delete_vault_ec2_client_configuration([...]) | Call to deprecated function 'delete_vault_ec2_client_configuration'. |
| disable_audit_backend(name) | Call to deprecated function 'disable_audit_backend'. |
| disable_auth_backend(mount_point) | Call to deprecated function 'disable_auth_backend'. |
| disable_secret_backend(mount_point) | Call to deprecated function 'disable_secret_backend'. |
| enable_audit_backend(backend_type[, ...]) | Call to deprecated function 'enable_audit_backend'. |
| enable_auth_backend(backend_type[, ...]) | Call to deprecated function 'enable_auth_backend'. |
| enable_secret_backend(backend_type[, ...]) | Call to deprecated function 'enable_secret_backend'. |
| generate_root(key, nonce) | Call to deprecated function 'generate_root'. |
| get_app_id(app_id[, mount_point, wrap_ttl]) | GET /auth/<mount_point>/map/app-id/<app_id> |
| get_auth_backend_tuning(backend_type[, ...]) | Call to deprecated function 'get_auth_backend_tuning'. |
| get_backed_up_keys() | Call to deprecated function 'get_backed_up_keys'. |
| get_ec2_role(role[, mount_point]) | Call to deprecated function 'get_ec2_role'. |
| get_kubernetes_configuration([mount_point]) | GET /auth/<mount_point>/config |
| get_kubernetes_role(name[, mount_point]) | GET /auth/<mount_point>/role/:name |
| get_policy(name[, parse]) | Retrieve the policy body for the named policy. |
| get_role(role_name[, mount_point]) | Call to deprecated function 'get_role'. |
| get_role_id(role_name[, mount_point]) | Call to deprecated function 'get_role_id'. |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| get_role_secret_id(role_name, secret_id[, ...]) | Call to deprecated function 'get_role_secret_id'. |
| get_role_secret_id_accessor(role_name, ...) | Call to deprecated function 'get_role_secret_id_accessor'. |
| get_secret_backend_tuning(backend_type[, ...]) | Call to deprecated function 'get_secret_backend_tuning'. |
| get_user_id(user_id[, mount_point, wrap_ttl]) | GET /auth/<mount_point>/map/user-id/<user_id> |
| get_vault_ec2_certificate_configuration(...) | Call to deprecated function 'get_vault_ec2_certificate_configuration'. |
| get_vault_ec2_client_configuration([mount_point]) | Call to deprecated function 'get_vault_ec2_client_configuration'. |
| initialize([secret_shares, ...]) | Call to deprecated function 'initialize'. |
| is_authenticated() | Helper method which returns the authentication status of the client |
| is_initialized() | Call to deprecated function 'is_initialized'. |
| is_sealed() | Call to deprecated function 'is_sealed'. |
| list(path) | GET /<path>?list=true |
| list_audit_backends() | Call to deprecated function 'list_audit_backends'. |
| list_auth_backends() | Call to deprecated function 'list_auth_backends'. |
| list_ec2_roles([mount_point]) | Call to deprecated function 'list_ec2_roles'. |
| list_kubernetes_roles([mount_point]) | GET /auth/<mount_point>/role?list=true |
| list_policies() | Call to deprecated function 'list_policies'. |
| list_role_secrets(role_name[, mount_point]) | Call to deprecated function 'list_role_secrets'. |
| list_roles([mount_point]) | Call to deprecated function 'list_roles'. |
| list_secret_backends() | Call to deprecated function 'list_secret_backends'. |
| list_token_roles() | GET /auth/token/roles?list=true |
| list_userpass([mount_point]) | GET /auth/<mount point>/users?list=true |
| list_vault_ec2_certificate_configurations([...]) | Call to deprecated function 'list_vault_ec2_certificate_configurations'. |
| login(url[, use_token]) | Perform a login request. |
| logout([revoke_token]) | Clears the token used for authentication, optionally revoking it before doing so. |
| lookup_token([token, accessor, wrap_ttl]) | GET /auth/token/lookup/<token> |
| read(path[, wrap_ttl]) | GET /<path> |
| read_lease(lease_id) | Call to deprecated function 'read_lease'. |
| read_userpass(username[, mount_point]) | GET /auth/<mount point>/users/<username> |
| rekey(key[, nonce]) | Call to deprecated function 'rekey'. |
| rekey_multi(keys[, nonce]) | Call to deprecated function 'rekey_multi'. |
| remount_secret_backend(from_mount_point, ...) | Call to deprecated function 'remount_secret_backend'. |
| renew_secret(lease_id[, increment]) | Call to deprecated function 'renew_secret'. |
| renew_self_token([increment, wrap_ttl]) | POST /auth/token/renew-self |
| renew_token([token, increment, wrap_ttl]) | POST /auth/token/renew |
| revoke_secret(lease_id) | Call to deprecated function 'revoke_secret'. |
| revoke_secret_prefix(path_prefix) | Call to deprecated function 'revoke_secret_prefix'. |
| revoke_self_token() | PUT /auth/token/revoke-self |
| revoke_token(token[, orphan, accessor]) | POST /auth/token/revoke |
| revoke_token_prefix(prefix) | POST /auth/token/revoke-prefix/<prefix> |
| rotate() | Call to deprecated function 'rotate'. |
| seal() | Call to deprecated function 'seal'. |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| set_policy(name, rules) | Call to deprecated function 'set_policy'. |
| set_role_id(role_name, role_id[, mount_point]) | Call to deprecated function 'set_role_id'. |
| start_generate_root(key[, otp]) | Call to deprecated function 'start_generate_root'. |
| start_rekey([secret_shares, ...]) | Call to deprecated function 'start_rekey'. |
| token_role(role) | Returns the named token role. |
| transit_create_key(name[, ...]) | Call to deprecated function 'transit_create_key'. |
| transit_decrypt_data(name, ciphertext[, ...]) | Call to deprecated function 'transit_decrypt_data'. |
| transit_delete_key(name[, mount_point]) | Call to deprecated function 'transit_delete_key'. |
| transit_encrypt_data(name, plaintext[, ...]) | Call to deprecated function 'transit_encrypt_data'. |
| transit_export_key(name, key_type[, ...]) | Call to deprecated function 'transit_export_key'. |
| transit_generate_data_key(name, key_type[, ...]) | Call to deprecated function 'transit_generate_data_key'. |
| transit_generate_hmac(name, hmac_input[, ...]) | Call to deprecated function 'transit_generate_hmac'. |
| transit_generate_rand_bytes([data_bytes, ...]) | Call to deprecated function 'transit_generate_rand_bytes'. |
| transit_hash_data(hash_input[, algorithm, ...]) | Call to deprecated function 'transit_hash_data'. |
| transit_list_keys([mount_point]) | Call to deprecated function 'transit_list_keys'. |
| transit_read_key(name[, mount_point]) | Call to deprecated function 'transit_read_key'. |
| transit_rewrap_data(name, ciphertext[, ...]) | Call to deprecated function 'transit_rewrap_data'. |
| transit_rotate_key(name[, mount_point]) | Call to deprecated function 'transit_rotate_key'. |
| transit_sign_data(name, input_data[, ...]) | Call to deprecated function 'transit_sign_data'. |
| transit_update_key(name[, ...]) | Call to deprecated function 'transit_update_key'. |
| transit_verify_signed_data(name, input_data) | Call to deprecated function 'transit_verify_signed_data'. |
| tune_auth_backend(backend_type[, ...]) | Call to deprecated function 'tune_auth_backend'. |
| tune_secret_backend(backend_type[, ...]) | Call to deprecated function 'tune_secret_backend'. |
| unseal(key) | Call to deprecated function 'unseal'. |
| unseal_multi(keys) | Call to deprecated function 'unseal_multi'. |
| unseal_reset() | Call to deprecated function 'unseal_reset'. |
| unwrap([token]) | Call to deprecated function 'unwrap'. |
| update_userpass_password(username, password) | POST /auth/<mount point>/users/<username>/password |
| update_userpass_policies(username, policies) | POST /auth/<mount point>/users/<username>/policies |
| urljoin(*args, **kwargs) | Call to deprecated function 'urljoin'. |
| write(path[, wrap_ttl]) | POST /<path> |

**Attributes**

| | |
|---|---|
| adapter | |
| allow_redirects | |
| auth | Accessor for the Client instance's auth methods. |
| generate_root_status | |

continues on next page

Table 3 – continued from previous page

| | |
|---|---|
| *ha_status* | Read the high availability status and current leader instance of Vault. |
| *key_status* | GET /sys/key-status |
| *rekey_status* | |
| *seal_status* | Read the seal status of the Vault. |
| *secrets* | Accessor for the Client instance's secrets engines. |
| *session* | |
| *sys* | Accessor for the Client instance's system backend methods. |
| *token* | |
| *url* | |

**__init__**(*url=None*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *adapter=<class 'hvac.adapters.JSONAdapter'>*, *namespace=None*, ***kwargs*)

Creates a new hvac client instance.

**Parameters**

- **url** (*str*) – Base URL for the Vault instance being addressed.
- **token** (*str*) – Authentication token to include in requests sent to Vault.
- **cert** (*tuple*) – Certificates for use in requests sent to the Vault instance. This should be a tuple with the certificate and then key.
- **verify** (*Union[bool,str]*) – Either a boolean to indicate whether TLS verification should be performed when sending requests to Vault, or a string pointing at the CA bundle to use for verification. See http://docs.python-requests.org/en/master/user/advanced/#ssl-cert-verification.
- **timeout** (*int*) – The timeout value for requests sent to Vault.
- **proxies** (*dict*) – Proxies to use when performing requests. See: http://docs.python-requests.org/en/master/user/advanced/#proxies
- **allow_redirects** (*bool*) – Whether to follow redirects when sending requests to Vault.
- **session** (*request.Session*) – Optional session object to use when performing request.
- **adapter** (*hvac.adapters.Adapter*) – Optional class to be used for performing requests. If none is provided, defaults to hvac.adapters.JSONRequest
- **kwargs** (*dict*) – Additional parameters to pass to the adapter constructor.
- **namespace** (*str*) – Optional Vault Namespace.

**property adapter**

**property allow_redirects**

**audit_hash**(*name*, *input*)

**Call to deprecated function 'audit_hash'. This method will be removed in version '0.9.0' Please use the 'calculate_ha**

Docstring content from this method's replacement copied below: Hash the given input data with the specified audit device's hash function and salt.

This endpoint can be used to discover whether a given plaintext string (the input parameter) appears in the audit log in obfuscated form.

**Supported methods:** POST: /sys/audit-hash/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – The path of the audit device to generate hashes for. This is part of the request URL.
- **input_to_hash** (*str | unicode*) – The input string to hash.

**Returns** The JSON response of the request.

**Return type** requests.Response

**property auth**

Accessor for the Client instance's auth methods. Provided via the *hvac.api.AuthMethods* class. :return: This Client instance's associated Auth instance. :rtype: hvac.api.AuthMethods

**auth_app_id**(*app_id*, *user_id*, *mount_point='app-id'*, *use_token=True*)

POST /auth/<mount point>/login

**Parameters**

- **app_id** –
- **user_id** –
- **mount_point** –
- **use_token** –

**Returns**

**Return type**

**auth_approle**(*role_id*, *secret_id=None*, *mount_point='approle'*, *use_token=True*)

**Call to deprecated function 'auth_approle'. This method will be removed in version '0.12.0' Please use the 'login' me**

Docstring content from this method's replacement copied below:

Login with APPROLE credentials.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **role_id** (*str | unicode*) – Role ID of the role.
- **secret_id** (*str | unicode*) – Secret ID of the role.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**auth_aws_iam**(*access_key*, *secret_key*, *session_token=None*, *header_value=None*, *mount_point='aws'*, *role=''*, *use_token=True*, *region='us-east-1'*)

**Call to deprecated function 'auth_aws_iam'. This method will be removed in version '0.11.2' Please use the 'iam_log**

Docstring content from this method's replacement copied below: Fetch a token

This endpoint verifies the pkcs7 signature of the instance identity document or the signature of the signed GetCallerIdentity request. With the ec2 auth method, or when inferring an EC2 instance, verifies that the instance is actually in a running state. Cross checks the constraints defined on the role with which the login is being performed. With the ec2 auth method, as an alternative to pkcs7 signature, the identity document along with its RSA digest can be supplied to this endpoint

**Parameters**

- **role** (*str*) – Name of the role against which the login is being attempted.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**auth_cubbyhole**(*token*)

Perform a login request with a wrapped token.

**Stores the unwrapped token in the resulting Vault response for use by the *hvac.adapters.Adapter()*** instance under the _adapater Client attribute.

**Parameters** **token** (*str | unicode*) – Wrapped token

**Returns** The (JSON decoded) response of the auth request

**Return type** dict

**auth_ec2**(*pkcs7*, *nonce=None*, *role=None*, *use_token=True*, *mount_point='aws-ec2'*)

**Call to deprecated function 'auth_ec2'. This method will be removed in version '0.11.2' Please use the 'ec2_login' me**

Docstring content from this method's replacement copied below: Retrieve a Vault token using an AWS authentication method mount's EC2 role.

**Parameters**

- **pkcs7** (*str*) – PKCS7 signature of the identity document with all newline characters removed.
- **nonce** (*str*) – The nonce to be used for subsequent login requests.
- **role** (*str*) – Name of the role against which the login is being attempted.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**auth_gcp**(**args*, ***kwargs*)

**Call to deprecated function 'auth_gcp'. This method will be removed in version '0.9.0' Please use the 'login' method**

Docstring content from this method's replacement copied below: Login to retrieve a Vault token via the GCP auth method.

**This endpoint takes a signed JSON Web Token (JWT) and a role name for some entity. It verifies the JWT** signature with Google Cloud to authenticate that entity and then authorizes the entity for the given role.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **role** (*str | unicode*) – The name of the role against which the login is being attempted.
- **jwt** (*str | unicode*) – A signed JSON web token
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**auth_github**(**args*, ***kwargs*)

**Call to deprecated function 'auth_github'. This method will be removed in version '0.8.0' Please use the 'login' meth** Docstring content from this method's replacement copied below: Login using GitHub access token.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **token** (*str | unicode*) – GitHub personal API token.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the login request.

**Return type** dict

**auth_kubernetes**(*role*, *jwt*, *use_token=True*, *mount_point='kubernetes'*)

POST /auth/<mount_point>/login

**Parameters**

- **role** (*str.*) – Name of the role against which the login is being attempted.
- **jwt** (*str.*) – Signed JSON Web Token (JWT) for authenticating a service account.
- **use_token** (*bool.*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the current Client class instance.
- **mount_point** (*str.*) – The "path" the k8s auth backend was mounted on. Vault currently defaults to "kubernetes".

**Returns** Parsed JSON response from the config POST request.

**Return type** dict.

**auth_ldap**(**args*, ***kwargs*)

**Call to deprecated function 'auth_ldap'. This method will be removed in version '0.8.0' Please use the 'login' method**
Docstring content from this method's replacement copied below:

Log in with LDAP credentials.

**Supported methods:** POST: /auth/{mount_point}/login/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – The username of the LDAP user
- **password** (*str | unicode*) – The password for the LDAP user
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the login_with_user request.

**Return type** requests.Response

**auth_tls**(*mount_point='cert'*, *use_token=True*)

**Call to deprecated function 'auth_tls'. This method will be removed in version '0.13.0' Please use the 'login' method**
Docstring content from this method's replacement copied below:

**Log in and fetch a token. If there is a valid chain to a CA configured in the method and all role constraints** are matched, a token will be issued. If the certificate has DNS SANs in it, each of those will be verified. If Common Name is required to be verified, then it should be a fully qualified DNS domain name and must be duplicated as a DNS SAN

**Supported methods:** POST: /auth/<mount point>/login Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Authenticate against only the named certificate role, returning its policy list if successful. If not set, defaults to trying all certificate roles and returning any one that matches.
- **cacert** (*str | bool*) – The value used here is for the Vault TLS Listener CA certificate, not the CA that issued the client authentication certificate. This can be omitted if the CA used to issue the Vault server certificate is trusted by the local system executing this command.
- **cert_pem** – Location of the cert.pem used to authenticate the host.
- **key_pem** – Location of the public key.pem used to authenticate the host.
- **key_pem** – str | unicode
- **mount_point** –
- **use_token** – If the returned token is stored in the client
- **use_token** – bool

**Tupe cert_pem** str | unicode

**Returns** The response of the login request.

**Return type** requests.Response

**auth_userpass**(*username*, *password*, *mount_point='userpass'*, *use_token=True*, ***kwargs*)
POST /auth/<mount point>/login/<username>

**Parameters**

- **username** –
- **password** –
- **mount_point** –
- **use_token** –
- **kwargs** –

**Returns**

**Return type**

**cancel_generate_root**()

**Call to deprecated function 'cancel_generate_root'. This method will be removed in version '0.9.0' Please use the 'ca**
Docstring content from this method's replacement copied below: Cancel any in-progress root generation attempt.

This clears any progress made. This must be called to change the OTP or PGP key being used.

**Supported methods:** DELETE: /sys/generate-root/attempt. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** request.Response

**cancel_rekey**()

**Call to deprecated function 'cancel_rekey'. This method will be removed in version '0.9.0' Please use the 'cancel_rek**
Docstring content from this method's replacement copied below: Cancel any in-progress rekey.

This clears the rekey settings as well as any progress made. This must be called to change the parameters of the rekey.

Note: Verification is still a part of a rekey. If rekeying is canceled during the verification flow, the current unseal keys remain valid.

**Supported methods:** DELETE: /sys/rekey/init. Produces: 204 (empty body) DELETE: /sys/rekey-recovery-key/init. Produces: 204 (empty body)

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The response of the request.

**Return type** requests.Response

**close**()
Call to deprecated function 'close'. This method will be removed in version '0.8.0' Please use the 'close' method on the 'hvac.adapters' class moving forward. Docstring content from this method's replacement copied below: Close the underlying Requests session.

**create_app_id**(*app_id*, *policies*, *display_name=None*, *mount_point='app-id'*, ***kwargs*)
POST /auth/<mount point>/map/app-id/<app_id>

**Parameters**

- **app_id** –

- **policies** –
- **display_name** –
- **mount_point** –
- **kwargs** –

**Returns**

**Return type**

**create_ec2_role**(*role*, *bound_ami_id=None*, *bound_account_id=None*, *bound_iam_role_arn=None*, *bound_iam_instance_profile_arn=None*, *bound_ec2_instance_id=None*, *bound_region=None*, *bound_vpc_id=None*, *bound_subnet_id=None*, *role_tag=None*, *ttl=None*, *max_ttl=None*, *period=None*, *policies=None*, *allow_instance_migration=False*, *disallow_reauthentication=False*, *resolve_aws_unique_ids=None*, *mount_point='aws-ec2'*)

**Call to deprecated function 'create_ec2_role'. This method will be removed in version '0.11.2' Please use the 'create_**

Docstring content from this method's replacement copied below: Register a role in the method.

**Parameters**

- **role** –
- **auth_type** –
- **bound_ami_id** –
- **bound_account_id** –
- **bound_region** –
- **bound_vpc_id** –
- **bound_subnet_id** –
- **bound_iam_role_arn** –
- **bound_iam_instance_profile_arn** –
- **bound_ec2_instance_id** –
- **role_tag** –
- **bound_iam_principal_arn** –
- **inferred_entity_type** –
- **inferred_aws_region** –
- **resolve_aws_unique_ids** –
- **ttl** –
- **max_ttl** –
- **period** –
- **policies** –
- **allow_instance_migration** –
- **disallow_reauthentication** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_ec2_role_tag**(*role*, *policies=None*, *max_ttl=None*, *instance_id=None*, *disallow_reauthentication=False*, *allow_instance_migration=False*, *mount_point='aws-ec2'*)

**Call to deprecated function 'create_ec2_role_tag'. This method will be removed in version '0.11.2' Please use the 'cr**

Docstring content from this method's replacement copied below: Create a role tag on the role, which helps in restricting the capabilities that are set on the role.

Role tags are not tied to any specific ec2 instance unless specified explicitly using the instance_id parameter. By default, role tags are designed to be used across all instances that satisfies the constraints on the role. Regardless of which instances have role tags on them, capabilities defined in a role tag must be a strict subset of the given role's capabilities. Note that, since adding and removing a tag is often a widely distributed privilege, care needs to be taken to ensure that the instances are attached with correct tags to not let them gain more privileges than what were intended. If a role tag is changed, the capabilities inherited by the instance will be those defined on the new role tag. Since those must be a subset of the role capabilities, the role should never provide more capabilities than any given instance can be allowed to gain in a worst-case scenario

**Parameters**

- **role** (*str*) – Name of the role.
- **policies** (*list*) – Policies to be associated with the tag. If set, must be a subset of the role's policies. If set, but set to an empty value, only the 'default' policy will be given to issued tokens.
- **max_ttl** (*str*) – The maximum allowed lifetime of tokens issued using this role.
- **instance_id** (*str*) – Instance ID for which this tag is intended for. If set, the created tag can only be used by the instance with the given ID.
- **disallow_reauthentication** (*bool*) – If set, only allows a single token to be granted per instance ID. This can be cleared with the auth/aws/identity-whitelist endpoint. Defaults to 'false'. Mutually exclusive with allow_instance_migration.
- **allow_instance_migration** (*bool*) – If set, allows migration of the underlying instance where the client resides. This keys off of pendingTime in the metadata document, so essentially, this disables the client nonce check whenever the instance is migrated to a new host and pendingTime is newer than the previously-remembered time. Use with caution. Defaults to 'false'. Mutually exclusive with disallow_reauthentication.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The create role tag response.

**Return type** dict

**create_kubernetes_configuration**(*kubernetes_host*, *kubernetes_ca_cert=None*, *token_reviewer_jwt=None*, *pem_keys=None*, *mount_point='kubernetes'*)

POST /auth/<mount_point>/config

**Parameters**

- **kubernetes_host** (*str.*) – A host:port pair, or a URL to the base of the Kubernetes API server.
- **kubernetes_ca_cert** (*str.*) – PEM encoded CA cert for use by the TLS client used to talk with the Kubernetes API.

- **token_reviewer_jwt** (*str.*) – A service account JWT used to access the TokenReview API to validate other JWTs during login. If not set the JWT used for login will be used to access the API.
- **pem_keys** (*list.*) – Optional list of PEM-formated public keys or certificates used to verify the signatures of Kubernetes service account JWTs. If a certificate is given, its public key will be extracted. Not every installation of Kubernetes exposes these keys.
- **mount_point** (*str.*) – The "path" the k8s auth backend was mounted on. Vault currently defaults to "kubernetes".

**Returns** Will be an empty body with a 204 status code upon success

**Return type** requests.Response.

**create_kubernetes_role**(*name*, *bound_service_account_names*, *bound_service_account_namespaces*, *ttl=''*, *max_ttl=''*, *period=''*, *policies=None*, *token_type=''*, *mount_point='kubernetes'*)

POST /auth/<mount_point>/role/:name

**Parameters**

- **name** (*str.*) – Name of the role.
- **bound_service_account_names** (*list.*) – List of service account names able to access this role. If set to "*" all names are allowed, both this and bound_service_account_namespaces can not be "*".
- **bound_service_account_namespaces** (*list.*) – List of namespaces allowed to access this role. If set to "*" all namespaces are allowed, both this and bound_service_account_names can not be set to "*".
- **ttl** (*str.*) – The TTL period of tokens issued using this role in seconds.
- **max_ttl** (*str.*) – The maximum allowed lifetime of tokens issued in seconds using this role.
- **period** (*str.*) – If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this parameter.
- **policies** (*list.*) – Policies to be set on tokens issued using this role
- **mount_point** (*str.*) – The "path" the k8s auth backend was mounted on. Vault currently defaults to "kubernetes".

**Returns** Will be an empty body with a 204 status code upon success

**Return type** requests.Response.

**create_role**(*role_name*, *mount_point='approle'*, ***kwargs*)

**Call to deprecated function 'create_role'. This method will be removed in version '0.12.0' Please use the 'create_or_u**

Docstring content from this method's replacement copied below:

Create/update approle.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}. Produces: 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the approle.
- **bind_secret_id** (*bool*) – Require secret_id to be presented when logging in using this approle.

- **secret_id_bound_cidrs** (*list*) – Blocks of IP addresses which can perform login operations.
- **secret_id_num_uses** (*int*) – Number of times any secret_id can be used to fetch a token. A value of zero allows unlimited uses.
- **secret_id_ttl** (*str | unicode*) – Duration after which a secret_id expires. This can be specified as an integer number of seconds or as a duration value like "5m".
- **enable_local_secret_ids** (*bool*) – Secret IDs generated using role will be cluster local.
- **token_ttl** (*str | unicode*) – Incremental lifetime for generated tokens. This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_max_ttl** (*str | unicode*) – Maximum lifetime for generated tokens: This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_policies** (*list*) – List of policies to encode onto generated tokens.
- **token_bound_cidrs** (*list*) – Blocks of IP addresses which can authenticate successfully.
- **token_explicit_max_ttl** (*str | unicode*) – If set, will encode an explicit max TTL onto the token. This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_no_default_policy** (*bool*) – Do not add the default policy to generated tokens, use only tokens specified in token_policies.
- **token_num_uses** (*int*) – Maximum number of times a generated token may be used. A value of zero allows unlimited uses.
- **token_period** (*str | unicode*) – The period, if any, to set on the token. This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_type** (*str | unicode*) – The type of token that should be generated, can be "service", "batch", or "default".
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**create_role_custom_secret_id**(*role_name*, *secret_id*, *meta=None*, *mount_point='approle'*)

**Call to deprecated function 'create_role_custom_secret_id'. This method will be removed in version '0.12.0' Please u**
Docstring content from this method's replacement copied below:

Generates and issues a new Secret ID on a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/custom-secret-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **secret_id** (*str | unicode*) – The Secret ID to read.
- **metadata** (*dict*) – Metadata to be tied to the Secret ID.
- **cidr_list** (*list*) – Blocks of IP addresses which can perform login operations.
- **token_bound_cidrs** (*list*) – Blocks of IP addresses which can authenticate successfully.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**create_role_secret_id**(*role_name*, *meta=None*, *cidr_list=None*, *token_bound_cidrs=None*, *wrap_ttl=None*, *mount_point='approle'*)

**Call to deprecated function 'create_role_secret_id'. This method will be removed in version '0.12.0' Please use the 'g**
Docstring content from this method's replacement copied below:

Generates and issues a new Secret ID on a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **metadata** (*dict*) – Metadata to be tied to the Secret ID.
- **cidr_list** (*list*) – Blocks of IP addresses which can perform login operations.
- **token_bound_cidrs** (*list*) – Blocks of IP addresses which can authenticate successfully.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**create_token**(*role=None*, *token_id=None*, *policies=None*, *meta=None*, *no_parent=False*, *lease=None*, *display_name=None*, *num_uses=None*, *no_default_policy=False*, *ttl=None*, *orphan=False*, *wrap_ttl=None*, *renewable=None*, *explicit_max_ttl=None*, *period=None*, *token_type=None*)

POST /auth/token/create

POST /auth/token/create/<role>

POST /auth/token/create-orphan

**Parameters**

- **role** –
- **token_id** –
- **policies** –
- **meta** –
- **no_parent** –
- **lease** –
- **display_name** –
- **num_uses** –
- **no_default_policy** –
- **ttl** –
- **orphan** –
- **wrap_ttl** –

- **renewable** –
- **explicit_max_ttl** –
- **period** –
- **token_type** –

**Returns**

**Return type**

**create_token_role**(*role*, *allowed_policies=None*, *disallowed_policies=None*, *orphan=None*, *period=None*, *renewable=None*, *path_suffix=None*, *explicit_max_ttl=None*)

POST /auth/token/roles/<role>

**Parameters**

- **role** –
- **allowed_policies** –
- **disallowed_policies** –
- **orphan** –
- **period** –
- **renewable** –
- **path_suffix** –
- **explicit_max_ttl** –

**Returns**

**Return type**

**create_user_id**(*user_id*, *app_id*, *cidr_block=None*, *mount_point='app-id'*, ***kwargs*)

POST /auth/<mount point>/map/user-id/<user_id>

**Parameters**

- **user_id** –
- **app_id** –
- **cidr_block** –
- **mount_point** –
- **kwargs** –

**Returns**

**Return type**

**create_userpass**(*username*, *password*, *policies*, *mount_point='userpass'*, ***kwargs*)

POST /auth/<mount point>/users/<username>

**Parameters**

- **username** –
- **password** –
- **policies** –
- **mount_point** –
- **kwargs** –

**Returns**

**Return type**

**create_vault_ec2_certificate_configuration**(*cert_name*, *aws_public_cert*, *mount_point='aws-ec2'*)

**Call to deprecated function 'create_vault_ec2_certificate_configuration'. This method will be removed in version '0.1**
Docstring content from this method's replacement copied below: Register AWS public key to be used to verify the instance identity documents.

While the PKCS#7 signature of the identity documents have DSA digest, the identity signature will have RSA digest, and hence the public keys for each type varies respectively. Indicate the type of the public key using the "type" parameter

**Supported methods:** POST: /auth/{mount_point}/config/certificate/:cert_name Produces: 204 (empty body)

**Parameters**

- **cert_name** (*string | unicode*) – Name of the certificate
- **aws_public_cert** – Base64 encoded AWS Public key required to verify PKCS7 signature of the EC2 instance metadata
- **document_type** (*string | unicode*) – Takes the value of either "pkcs7" or "identity", indicating the type of document which can be verified using the given certificate
- **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request

**Return type** request.Response

**create_vault_ec2_client_configuration**(*access_key*, *secret_key*, *endpoint=None*, *mount_point='aws-ec2'*)

**Call to deprecated function 'create_vault_ec2_client_configuration'. This method will be removed in version '0.11.2'**
Docstring content from this method's replacement copied below: Configure the credentials required to perform API calls to AWS as well as custom endpoints to talk to AWS API.

The instance identity document fetched from the PKCS#7 signature will provide the EC2 instance ID. The credentials configured using this endpoint will be used to query the status of the instances via DescribeInstances API. If static credentials are not provided using this endpoint, then the credentials will be retrieved from the environment variables AWS_ACCESS_KEY, AWS_SECRET_KEY and AWS_REGION respectively. If the credentials are still not found and if the method is configured on an EC2 instance with metadata querying capabilities, the credentials are fetched automatically

**Supported methods:** POST: /auth/{mount_point}/config Produces: 204 (empty body)

**Parameters**

- **max_retries** (*int*) – Number of max retries the client should use for recoverable errors. The default (-1) falls back to the AWS SDK's default behavior
- **access_key** (*str | unicode*) – AWS Access key with permissions to query AWS APIs. The permissions required depend on the specific configurations. If using the iam auth method without inferencing, then no credentials are necessary. If using the ec2 auth method or using the iam auth method with inferencing, then these credentials need access to ec2:DescribeInstances. If additionally a bound_iam_role is specified, then these credentials also need access to iam:GetInstanceProfile. If, however, an alternate sts configuration is set for the target account, then the credentials must be permissioned to call

sts:AssumeRole on the configured role, and that role must have the permissions described here

- **secret_key** (*str | unicode*) – AWS Secret key with permissions to query AWS APIs
- **endpoint** (*str | unicode*) – URL to override the default generated endpoint for making AWS EC2 API calls
- **iam_endpoint** (*str | unicode*) – URL to override the default generated endpoint for making AWS IAM API calls
- **sts_endpoint** (*str | unicode*) – URL to override the default generated endpoint for making AWS STS API calls
- **iam_server_id_header_value** (*str | unicode*) – The value to require in the X-Vault-AWS-IAM-Server-ID header as part of GetCallerIdentity requests that are used in the iam auth method. If not set, then no value is required or validated. If set, clients must include an X-Vault-AWS-IAM-Server-ID header in the headers of login requests, and further this header must be among the signed headers validated by AWS. This is to protect against different types of replay attacks, for example a signed request sent to a dev server being resent to a production server
- **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete**(*path*)
DELETE /<path>

**Parameters** **path** –

**Returns**

**Return type**

**delete_app_id**(*app_id*, *mount_point='app-id'*)
DELETE /auth/<mount_point>/map/app-id/<app_id>

**Parameters**

- **app_id** –
- **mount_point** –

**Returns**

**Return type**

**delete_ec2_role**(*role*, *mount_point='aws-ec2'*)

**Call to deprecated function 'delete_ec2_role'. This method will be removed in version '0.11.2' Please use the 'delete_**
Docstring content from this method's replacement copied below: Deletes the previously registered role

**Parameters**

- **role** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_kubernetes_role**(*role*, *mount_point='kubernetes'*)
DELETE /auth/<mount_point>/role/:role

**Parameters**

- **role** (*Name of the role.*) – str.
- **mount_point** (*str.*) – The "path" the k8s auth backend was mounted on. Vault currently defaults to "kubernetes".

**Returns** Will be an empty body with a 204 status code upon success.

**Return type** requests.Response.

**delete_policy**(*name*)

**Call to deprecated function 'delete_policy'. This method will be removed in version '0.9.0' Please use the 'delete_poli**
Docstring content from this method's replacement copied below: Delete the policy with the given name.

This will immediately affect all users associated with this policy.

**Supported methods:** DELETE: /sys/policy/{name}. Produces: 204 (empty body)

**Parameters** **name** (*str | unicode*) – Specifies the name of the policy to delete.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*role_name*, *mount_point='approle'*)

**Call to deprecated function 'delete_role'. This method will be removed in version '0.12.0' Please use the 'delete_role'**
Docstring content from this method's replacement copied below:

Delete role in the auth method.

**Supported methods:** DELETE: /auth/{mount_point}/role/{role_name}. Produces: 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**delete_role_secret_id**(*role_name*, *secret_id*, *mount_point='approle'*)

**Call to deprecated function 'delete_role_secret_id'. This method will be removed in version '0.12.0' Please use the 'd**
Docstring content from this method's replacement copied below:

Destroys a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id/destroy. Produces 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id** (*str | unicode*) – The Secret ID to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**delete_role_secret_id_accessor**(*role_name*, *secret_id_accessor*, *mount_point='approle'*)

**Call to deprecated function 'delete_role_secret_id_accessor'. This method will be removed in version '0.12.0' Please**
Docstring content from this method's replacement copied below:

Destroys a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id-accessor/destroy. Produces: 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id_accessor** (*str | unicode*) – The accessor for the Secret ID to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**delete_token_role**(*role*)
Deletes the named token role.

**Parameters** **role** –

**Returns**

**Return type**

**delete_user_id**(*user_id*, *mount_point='app-id'*)
DELETE /auth/<mount_point>/map/user-id/<user_id>

**Parameters**

- **user_id** –
- **mount_point** –

**Returns**

**Return type**

**delete_userpass**(*username*, *mount_point='userpass'*)
DELETE /auth/<mount point>/users/<username>

**Parameters**

- **username** –
- **mount_point** –

**Returns**

**Return type**

**delete_vault_ec2_client_configuration**(*mount_point='aws-ec2'*)

**Call to deprecated function 'delete_vault_ec2_client_configuration'. This method will be removed in version '0.11.2'**
Docstring content from this method's replacement copied below: Delete previously configured AWS access credentials,

**Supported methods:** DELETE: /auth/{mount_point}/config Produces: 204 (empty body)

**Parameters** **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**disable_audit_backend**(*name*)

**Call to deprecated function 'disable_audit_backend'. This method will be removed in version '0.9.0' Please use the 'd**
Docstring content from this method's replacement copied below: Disable the audit device at the given path.

**Supported methods:** DELETE: /sys/audit/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – The path of the audit device to delete. This is part of the request URL.

**Returns** The response of the request.

**Return type** requests.Response

**disable_auth_backend**(*mount_point*)

**Call to deprecated function 'disable_auth_backend'. This method will be removed in version '0.9.0' Please use the 'd**
Docstring content from this method's replacement copied below: Disable the auth method at the given auth path.

**Supported methods:** DELETE: /sys/auth/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.

**Returns** The response of the request.

**Return type** requests.Response

**disable_secret_backend**(*mount_point*)

**Call to deprecated function 'disable_secret_backend'. This method will be removed in version '0.9.0' Please use the '**
Docstring content from this method's replacement copied below: Disable the mount point specified by the provided path.

**Supported methods:** DELETE: /sys/mounts/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.

**Returns** The response of the request.

**Return type** requests.Response

**enable_audit_backend**(*backend_type*, *description=None*, *options=None*, *name=None*)

**Call to deprecated function 'enable_audit_backend'. This method will be removed in version '0.9.0' Please use the 'e**
Docstring content from this method's replacement copied below: Enable a new audit device at the supplied path.

The path can be a single word name or a more complex, nested path.

**Supported methods:** PUT: /sys/audit/{path}. Produces: 204 (empty body)

**Parameters**

- **device_type** (*str | unicode*) – Specifies the type of the audit device.
- **description** (*str | unicode*) – Human-friendly description of the audit device.
- **options** (*str | unicode*) – Configuration options to pass to the audit device itself. This is dependent on the audit device type.

- **path** (*str | unicode*) – Specifies the path in which to enable the audit device. This is part of the request URL.
- **local** (*bool*) – Specifies if the audit device is a local only.

**Returns** The response of the request.

**Return type** requests.Response

**enable_auth_backend**(*backend_type*, *description=None*, *mount_point=None*, *config=None*, *plugin_name=None*)

**Call to deprecated function 'enable_auth_backend'. This method will be removed in version '0.9.0' Please use the 'e**

Docstring content from this method's replacement copied below: Enable a new auth method.

After enabling, the auth method can be accessed and configured via the auth path specified as part of the URL. This auth path will be nested under the auth prefix.

**Supported methods:** POST: /sys/auth/{path}. Produces: 204 (empty body)

**Parameters**

- **method_type** (*str | unicode*) – The name of the authentication method type, such as "github" or "token".
- **description** (*str | unicode*) – A human-friendly description of the auth method.
- **config** (*dict*) – Configuration options for this auth method. These are the possible values:
  - **default_lease_ttl**: The default lease duration, specified as a string duration like "5s" or "30m".
  - **max_lease_ttl**: The maximum lease duration, specified as a string duration like "5s" or "30m".
  - **audit_non_hmac_request_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
  - **audit_non_hmac_response_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
  - **listing_visibility**: Speficies whether to show this mount in the UI-specific listing endpoint.
  - **passthrough_request_headers**: Comma-separated list of headers to whitelist and pass from the request to the backend.
- **plugin_name** (*str | unicode*) – The name of the auth plugin to use based from the name in the plugin catalog. Applies only to plugin methods.
- **local** (*bool*) – <Vault enterprise only> Specifies if the auth method is a local only. Local auth methods are not replicated nor (if a secondary) removed by replication.
- **path** (*str | unicode*) – The path to mount the method on. If not provided, defaults to the value of the "method_type" argument.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

**enable_secret_backend**(*backend_type*, *description=None*, *mount_point=None*, *config=None*, *options=None*)

**Call to deprecated function 'enable_secret_backend'. This method will be removed in version '0.9.0' Please use the '**

Docstring content from this method's replacement copied below: Enable a new secrets engine at the given path.

**Supported methods:** POST: /sys/mounts/{path}. Produces: 204 (empty body)

**Parameters**

- **backend_type** (*str | unicode*) – The name of the backend type, such as "github" or "token".
- **path** (*str | unicode*) – The path to mount the method on. If not provided, defaults to the value of the "backend_type" argument.
- **description** (*str | unicode*) – A human-friendly description of the mount.
- **config** (*dict*) – Configuration options for this mount. These are the possible values:
  - **default_lease_ttl**: The default lease duration, specified as a string duration like "5s" or "30m".
  - **max_lease_ttl**: The maximum lease duration, specified as a string duration like "5s" or "30m".
  - **force_no_cache**: Disable caching.
  - **plugin_name**: The name of the plugin in the plugin catalog to use.
  - **audit_non_hmac_request_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
  - **audit_non_hmac_response_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
  - **listing_visibility**: Specifies whether to show this mount in the UI-specific listing endpoint. ("unauth" or "hidden")
  - **passthrough_request_headers**: Comma-separated list of headers to whitelist and pass from the request to the backend.
- **options** (*dict*) – Specifies mount type specific options that are passed to the backend.
  - **version**: <KV> The version of the KV to mount. Set to "2" for mount KV v2.
- **plugin_name** (*str | unicode*) – Specifies the name of the plugin to use based from the name in the plugin catalog. Applies only to plugin backends.
- **local** (*bool*) – <Vault enterprise only> Specifies if the auth method is a local only. Local auth methods are not replicated nor (if a secondary) removed by replication.
- **seal_wrap** (*bool*) – <Vault enterprise only> Enable seal wrapping for the mount.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

**generate_root**(*key*, *nonce*)

**Call to deprecated function 'generate_root'. This method will be removed in version '0.9.0' Please use the 'generate_**
Docstring content from this method's replacement copied below: Enter a single master key share to progress the root generation attempt.

If the threshold number of master key shares is reached, Vault will complete the root generation and issue the new token. Otherwise, this API must be called multiple times until that threshold is met. The attempt nonce must be provided with each call.

**Supported methods:** PUT: /sys/generate-root/update. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single master key share.
- **nonce** (*str | unicode*) – The nonce of the attempt.

**Returns** The JSON response of the request.

**Return type** dict

**property generate_root_status**

**get_app_id**(*app_id*, *mount_point='app-id'*, *wrap_ttl=None*)
GET /auth/<mount_point>/map/app-id/<app_id>

**Parameters**

- **app_id** –
- **mount_point** –
- **wrap_ttl** –

**Returns**

**Return type**

**get_auth_backend_tuning**(*backend_type*, *mount_point=None*)

**Call to deprecated function 'get_auth_backend_tuning'. This method will be removed in version '0.9.0' Please use th**
Docstring content from this method's replacement copied below: Read the given auth path's configuration.

This endpoint requires sudo capability on the final path, but the same functionality can be achieved without sudo via sys/mounts/auth/[auth-path]/tune.

**Supported methods:** GET: /sys/auth/{path}/tune. Produces: 200 application/json

**Parameters** **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.

**Returns** The JSON response of the request.

**Return type** dict

**get_backed_up_keys**()

**Call to deprecated function 'get_backed_up_keys'. This method will be removed in version '0.9.0' Please use the 'rea**
Docstring content from this method's replacement copied below: Retrieve the backup copy of PGP-encrypted unseal keys.

The returned value is the nonce of the rekey operation and a map of PGP key fingerprint to hex-encoded PGP-encrypted key.

**Supported methods:** PUT: /sys/rekey/backup. Produces: 200 application/json PUT: /sys/rekey-recovery-key/backup. Produces: 200 application/json

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** dict

**get_ec2_role**(*role*, *mount_point='aws-ec2'*)

**Call to deprecated function 'get_ec2_role'. This method will be removed in version '0.11.2' Please use the 'read_role**
Docstring content from this method's replacement copied below: Returns the previously registered role configuration

**Parameters**

- **role** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**get_kubernetes_configuration**(*mount_point='kubernetes'*)
GET /auth/<mount_point>/config

**Parameters** **mount_point** (*str.*) – The "path" the k8s auth backend was mounted on. Vault currently defaults to "kubernetes".

**Returns** Parsed JSON response from the config GET request

**Return type** dict.

**get_kubernetes_role**(*name*, *mount_point='kubernetes'*)
GET /auth/<mount_point>/role/:name

**Parameters**

- **name** (*str.*) – Name of the role.
- **mount_point** (*str.*) – The "path" the k8s auth backend was mounted on. Vault currently defaults to "kubernetes".

**Returns** Parsed JSON response from the read role GET request

**Return type** dict.

**get_policy**(*name*, *parse=False*)
Retrieve the policy body for the named policy.

**Parameters**

- **name** (*str | unicode*) – The name of the policy to retrieve.
- **parse** (*bool*) – Specifies whether to parse the policy body using pyhcl or not.

**Returns** The (optionally parsed) policy body for the specified policy.

**Return type** str | dict

**get_role**(*role_name*, *mount_point='approle'*)

**Call to deprecated function 'get_role'. This method will be removed in version '0.12.0' Please use the 'read_role' met**
Docstring content from this method's replacement copied below:

Read role in the auth method.

**Supported methods:** GET: /auth/{mount_point}/role/{role_name}. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role request.

**Return type** dict

**get_role_id**(*role_name*, *mount_point='approle'*)

**Call to deprecated function 'get_role_id'. This method will be removed in version '0.12.0' Please use the 'read_role_i**
Docstring content from this method's replacement copied below:

Reads the Role ID of a role in the auth method.

**Supported methods:** GET: /auth/{mount_point}/role/{role_name}/role-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**get_role_secret_id**(*role_name*, *secret_id*, *mount_point='approle'*)

**Call to deprecated function 'get_role_secret_id'. This method will be removed in version '0.12.0' Please use the 'read**
Docstring content from this method's replacement copied below:

Read the properties of a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id/lookup. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id** (*str | unicode*) – The Secret ID to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**get_role_secret_id_accessor**(*role_name*, *secret_id_accessor*, *mount_point='approle'*)

**Call to deprecated function 'get_role_secret_id_accessor'. This method will be removed in version '0.12.0' Please use**
Docstring content from this method's replacement copied below:

Read the properties of a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id-accessor/lookup. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id_accessor** (*str | unicode*) – The accessor for the Secret ID to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**get_secret_backend_tuning**(*backend_type*, *mount_point=None*)

**Call to deprecated function 'get_secret_backend_tuning'. This method will be removed in version '0.9.0' Please use t**
Docstring content from this method's replacement copied below: Read the given mount's configuration.

Unlike the mounts endpoint, this will return the current time in seconds for each TTL, which may be the system default or a mount-specific value.

**Supported methods:** GET: /sys/mounts/{path}/tune. Produces: 200 application/json

**Parameters** **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.

**Returns** The JSON response of the request.

**Return type** requests.Response

**get_user_id**(*user_id*, *mount_point='app-id'*, *wrap_ttl=None*)
GET /auth/<mount_point>/map/user-id/<user_id>

**Parameters**

- **user_id** –
- **mount_point** –
- **wrap_ttl** –

**Returns**

**Return type**

**get_vault_ec2_certificate_configuration**(*cert_name*, *mount_point='aws-ec2'*)

**Call to deprecated function 'get_vault_ec2_certificate_configuration'. This method will be removed in version '0.11.2**
Docstring content from this method's replacement copied below: Return previously configured AWS public key.

**Supported methods:** GET: /v1/auth/{mount_point}/config/certificate/:cert_name Produces: 200 application/json

**Parameters**

- **cert_name** (*str | unicode*) – Name of the certificate
- **mount_point** – The path the AWS auth method was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**get_vault_ec2_client_configuration**(*mount_point='aws-ec2'*)

**Call to deprecated function 'get_vault_ec2_client_configuration'. This method will be removed in version '0.11.2' Ple**
Docstring content from this method's replacement copied below: Read previously configured AWS access credentials.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**property ha_status**
Read the high availability status and current leader instance of Vault.

**Returns** The JSON response returned by read_leader_status()

**Return type** dict

**initialize**(*secret_shares=5*, *secret_threshold=3*, *pgp_keys=None*)

**Call to deprecated function 'initialize'. This method will be removed in version '0.9.0' Please use the 'initialize' meth**
Docstring content from this method's replacement copied below: Initialize a new Vault.

The Vault must not have been previously initialized. The recovery options, as well as the stored shares option, are only available when using Vault HSM.

**Supported methods:** PUT: /sys/init. Produces: 200 application/json

**Parameters**

- **secret_shares** (*int*) – The number of shares to split the master key into.
- **secret_threshold** (*int*) – Specifies the number of shares required to reconstruct the master key. This must be less than or equal secret_shares. If using Vault HSM with auto-unsealing, this value must be the same as secret_shares.
- **pgp_keys** (*list*) – List of PGP public keys used to encrypt the output unseal keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as secret_shares.
- **root_token_pgp_key** (*str | unicode*) – Specifies a PGP public key used to encrypt the initial root token. The key must be base64-encoded from its original binary representation.
- **stored_shares** (*int*) – <enterprise only> Specifies the number of shares that should be encrypted by the HSM and stored for auto-unsealing. Currently must be the same as secret_shares.
- **recovery_shares** (*int*) – <enterprise only> Specifies the number of shares to split the recovery key into.
- **recovery_threshold** (*int*) – <enterprise only> Specifies the number of shares required to reconstruct the recovery key. This must be less than or equal to recovery_shares.
- **recovery_pgp_keys** (*list*) – <enterprise only> Specifies an array of PGP public keys used to encrypt the output recovery keys. Ordering is preserved. The keys must be

base64-encoded from their original binary representation. The size of this array must be the same as recovery_shares.

**Returns** The JSON response of the request.

**Return type** dict

**is_authenticated**()
Helper method which returns the authentication status of the client

**Returns**

**Return type**

**is_initialized**()

**Call to deprecated function 'is_initialized'. This method will be removed in version '0.9.0' Please use the 'is_initializ**
Docstring content from this method's replacement copied below: Determine is Vault is initialized or not.

**Returns** True if Vault is initialized, False otherwise.

**Return type** bool

**is_sealed**()

**Call to deprecated function 'is_sealed'. This method will be removed in version '0.9.0' Please use the 'is_sealed' meth**
Docstring content from this method's replacement copied below: Determine if Vault is sealed.

**Returns** True if Vault is seal, False otherwise.

**Return type** bool

**property key_status**
GET /sys/key-status

**Returns** Information about the current encryption key used by Vault.

**Return type** dict

**list**(*path*)
GET /<path>?list=true

**Parameters** **path** –

**Returns**

**Return type**

**list_audit_backends**()

**Call to deprecated function 'list_audit_backends'. This method will be removed in version '0.9.0' Please use the 'list_**
Docstring content from this method's replacement copied below: List enabled audit devices.

It does not list all available audit devices. This endpoint requires sudo capability in addition to any path-specific capabilities.

**Supported methods:** GET: /sys/audit. Produces: 200 application/json

**Returns** JSON response of the request.

**Return type** dict

**list_auth_backends**()

**Call to deprecated function 'list_auth_backends'. This method will be removed in version '0.9.0' Please use the 'list_**
Docstring content from this method's replacement copied below: List all enabled auth methods.

**Supported methods:** GET: /sys/auth. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**list_ec2_roles**(*mount_point='aws-ec2'*)

**Call to deprecated function 'list_ec2_roles'. This method will be removed in version '0.11.2' Please use the 'list_roles**
Docstring content from this method's replacement copied below: Lists all the roles that are registered with the method

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_kubernetes_roles**(*mount_point='kubernetes'*)
GET /auth/<mount_point>/role?list=true

**Parameters** **mount_point** (*str.*) – The "path" the k8s auth backend was mounted on. Vault currently defaults to "kubernetes".

**Returns** Parsed JSON response from the list roles GET request.

**Return type** dict.

**list_policies**()

**Call to deprecated function 'list_policies'. This method will be removed in version '0.9.0' Please use the 'list_policies'**
Docstring content from this method's replacement copied below: List all configured policies.

**Supported methods:** GET: /sys/policy. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**list_role_secrets**(*role_name*, *mount_point='approle'*)

**Call to deprecated function 'list_role_secrets'. This method will be removed in version '0.12.0' Please use the 'list_se**
Docstring content from this method's replacement copied below:

Lists accessors of all issued Secret IDs for a role in the auth method.

**Supported methods:** LIST: /auth/{mount_point}/role/{role_name}/secret-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**list_roles**(*mount_point='approle'*)

**Call to deprecated function 'list_roles'. This method will be removed in version '0.12.0' Please use the 'list_roles' me**
Docstring content from this method's replacement copied below:

List existing roles created in the auth method.

**Supported methods:** LIST: /auth/{mount_point}/role. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the list_roles request.

**Return type** dict

**list_secret_backends**()

**Call to deprecated function 'list_secret_backends'. This method will be removed in version '0.9.0' Please use the 'list**
Docstring content from this method's replacement copied below: Lists all the mounted secrets engines.

**Supported methods:** POST: /sys/mounts. Produces: 200 application/json

**Returns** JSON response of the request.

**Return type** dict

**list_token_roles**()
GET /auth/token/roles?list=true

**Returns**

**Return type**

**list_userpass**(*mount_point='userpass'*)
GET /auth/<mount point>/users?list=true

**Parameters** **mount_point** –

**Returns**

**Return type**

**list_vault_ec2_certificate_configurations**(*mount_point='aws-ec2'*)

**Call to deprecated function 'list_vault_ec2_certificate_configurations'. This method will be removed in version '0.11.**
Docstring content from this method's replacement copied below: List AWS public certificates that are registered with the method.

**Supported methods** LIST: /auth/{mount_point}/config/certificates Produces: 200 application/json

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**login**(*url*, *use_token=True*, ***kwargs*)
Perform a login request.

**Associated request is typically to a path prefixed with "/v1/auth") and optionally stores the client token sent**
in the resulting Vault response for use by the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.

**Parameters**

- **url** (*str / unicode*) – Path to send the authentication request to.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the params sent with the request.

**Returns** The response of the auth request.

**Return type** requests.Response

**logout**(*revoke_token=False*)

Clears the token used for authentication, optionally revoking it before doing so.

**Parameters** **revoke_token** –

**Returns**

**Return type**

**lookup_token**(*token=None*, *accessor=False*, *wrap_ttl=None*)

GET /auth/token/lookup/<token>

GET /auth/token/lookup-accessor/<token-accessor>

GET /auth/token/lookup-self

**Parameters**

- **token** (*str.*) –
- **accessor** (*str.*) –
- **wrap_ttl** (*int.*) –

**Returns**

**Return type**

**read**(*path*, *wrap_ttl=None*)

GET /<path>

**Parameters**

- **path** –
- **wrap_ttl** –

**Returns**

**Return type**

**read_lease**(*lease_id*)

**Call to deprecated function 'read_lease'. This method will be removed in version '0.9.0' Please use the 'read_lease' n**

Docstring content from this method's replacement copied below: Retrieve lease metadata.

**Supported methods:** PUT: /sys/leases/lookup. Produces: 200 application/json

**Parameters** **lease_id** (*str / unicode*) – the ID of the lease to lookup.

**Returns** Parsed JSON response from the leases PUT request

**Return type** dict.

**read_userpass**(*username*, *mount_point='userpass'*)
GET /auth/<mount point>/users/<username>

**Parameters**

- **username** –
- **mount_point** –

**Returns**

**Return type**

**rekey**(*key*, *nonce=None*)

**Call to deprecated function 'rekey'. This method will be removed in version '0.9.0' Please use the 'rekey' method on**
Docstring content from this method's replacement copied below: Enter a single recovery key share to progress the rekey of the Vault.

If the threshold number of recovery key shares is reached, Vault will complete the rekey. Otherwise, this API must be called multiple times until that threshold is met. The rekey nonce operation must be provided with each call.

**Supported methods:** PUT: /sys/rekey/update. Produces: 200 application/json PUT: /sys/rekey-recovery-key/update. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single recovery share key.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey operation.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** dict

**rekey_multi**(*keys*, *nonce=None*)

**Call to deprecated function 'rekey_multi'. This method will be removed in version '0.9.0' Please use the 'rekey_mult**
Docstring content from this method's replacement copied below: Enter multiple recovery key shares to progress the rekey of the Vault.

If the threshold number of recovery key shares is reached, Vault will complete the rekey.

**Parameters**

- **keys** (*list*) – Specifies multiple recovery share keys.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey operation.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The last response of the rekey request.

**Return type** response.Request

**property rekey_status**

**remount_secret_backend**(*from_mount_point*, *to_mount_point*)

**Call to deprecated function 'remount_secret_backend'. This method will be removed in version '0.9.0' Please use the**
Docstring content from this method's replacement copied below: Move an already-mounted backend to a new mount point.

**Supported methods:** POST: /sys/remount. Produces: 204 (empty body)

**Parameters**

- **from_path** (*str | unicode*) – Specifies the previous mount point.
- **to_path** (*str | unicode*) – Specifies the new destination mount point.

**Returns** The response of the request.

**Return type** requests.Response

**renew_secret**(*lease_id*, *increment=None*)

**Call to deprecated function 'renew_secret'. This method will be removed in version '0.9.0' Please use the 'renew_lea**
Docstring content from this method's replacement copied below: Renew a lease, requesting to extend the lease.

**Supported methods:** PUT: /sys/leases/renew. Produces: 200 application/json

**Parameters**

- **lease_id** (*str | unicode*) – The ID of the lease to extend.
- **increment** (*int*) – The requested amount of time (in seconds) to extend the lease.

**Returns** The JSON response of the request

**Return type** dict

**renew_self_token**(*increment=None*, *wrap_ttl=None*)
POST /auth/token/renew-self

**Parameters**

- **increment** –
- **wrap_ttl** –

**Returns**

**Return type**

**renew_token**(*token=None*, *increment=None*, *wrap_ttl=None*)
POST /auth/token/renew

POST /auth/token/renew-self

**Parameters**

- **token** –
- **increment** –
- **wrap_ttl** –

**Returns**

**Return type**

For calls expecting to hit the renew-self endpoint please use the "renew_self_token" method instead

**revoke_secret**(*lease_id*)

**Call to deprecated function 'revoke_secret'. This method will be removed in version '0.9.0' Please use the 'revoke_le**
Docstring content from this method's replacement copied below: Revoke a lease immediately.

**Supported methods:** PUT: /sys/leases/revoke. Produces: 204 (empty body)

**Parameters** **lease_id** (*str | unicode*) – Specifies the ID of the lease to revoke.

**Returns** The response of the request.

**Return type** requests.Response

**revoke_secret_prefix**(*path_prefix*)

**Call to deprecated function 'revoke_secret_prefix'. This method will be removed in version '0.9.0' Please use the 'rev**
Docstring content from this method's replacement copied below: Revoke a lease immediately.

**Supported methods:** PUT: /sys/leases/revoke. Produces: 204 (empty body)

**Parameters** **lease_id** (*str | unicode*) – Specifies the ID of the lease to revoke.

**Returns** The response of the request.

**Return type** requests.Response

**revoke_self_token**()
PUT /auth/token/revoke-self

**Returns**

**Return type**

**revoke_token**(*token*, *orphan=False*, *accessor=False*)
POST /auth/token/revoke

POST /auth/token/revoke-orphan

POST /auth/token/revoke-accessor

**Parameters**

- **token** –
- **orphan** –
- **accessor** –

**Returns**

**Return type**

**revoke_token_prefix**(*prefix*)
POST /auth/token/revoke-prefix/<prefix>

**Parameters** **prefix** –

**Returns**

**Return type**

**rotate**()

**Call to deprecated function 'rotate'. This method will be removed in version '0.9.0' Please use the 'rotate_encryption**
Docstring content from this method's replacement copied below: Trigger a rotation of the backend encryption key.

This is the key that is used to encrypt data written to the storage backend, and is not provided to operators. This operation is done online. Future values are encrypted with the new key, while old values are decrypted with previous encryption keys.

This path requires sudo capability in addition to update.

**Supported methods:** PUT: /sys/rorate. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

**seal**()

**Call to deprecated function 'seal'. This method will be removed in version '0.9.0' Please use the 'seal' method on the**
Docstring content from this method's replacement copied below: Seal the Vault.

In HA mode, only an active node can be sealed. Standby nodes should be restarted to get the same effect. Requires a token with root policy or sudo capability on the path.

**Supported methods:** PUT: /sys/seal. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

**property seal_status**

Read the seal status of the Vault.

This is an unauthenticated endpoint.

**Supported methods:** GET: /sys/seal-status. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**property secrets**

Accessor for the Client instance's secrets engines. Provided via the *hvac.api.SecretsEngines* class.

**Returns** This Client instance's associated SecretsEngines instance.

**Return type** *hvac.api.SecretsEngines*

**property session**

**set_policy**(*name*, *rules*)

**Call to deprecated function 'set_policy'. This method will be removed in version '0.9.0' Please use the 'create_or_up**
Docstring content from this method's replacement copied below: Add a new or update an existing policy.

Once a policy is updated, it takes effect immediately to all associated users.

**Supported methods:** PUT: /sys/policy/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the policy to create.
- **policy** (*str | unicode | dict*) – Specifies the policy document.

- **pretty_print** (*bool*) – If True, and provided a dict for the policy argument, send the policy JSON to Vault with "pretty" formatting.

**Returns** The response of the request.

**Return type** requests.Response

**set_role_id**(*role_name*, *role_id*, *mount_point='approle'*)

**Call to deprecated function 'set_role_id'. This method will be removed in version '0.12.0' Please use the 'update_role**
Docstring content from this method's replacement copied below:

Updates the Role ID of a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/role-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **role_id** (*str | unicode*) – New value for the Role ID.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**start_generate_root**(*key*, *otp=False*)

**Call to deprecated function 'start_generate_root'. This method will be removed in version '0.9.0' Please use the 'star**
Docstring content from this method's replacement copied below: Initialize a new root generation attempt.

Only a single root generation attempt can take place at a time. One (and only one) of otp or pgp_key are required.

**Supported methods:** PUT: /sys/generate-root/attempt. Produces: 200 application/json

**Parameters**

- **otp** (*str | unicode*) – Specifies a base64-encoded 16-byte value. The raw bytes of the token will be XOR'd with this value before being returned to the final unseal key provider.
- **pgp_key** (*str | unicode*) – Specifies a base64-encoded PGP public key. The raw bytes of the token will be encrypted with this value before being returned to the final unseal key provider.

**Returns** The JSON response of the request.

**Return type** dict

**start_rekey**(*secret_shares=5*, *secret_threshold=3*, *pgp_keys=None*, *backup=False*)

**Call to deprecated function 'start_rekey'. This method will be removed in version '0.9.0' Please use the 'start_rekey'**
Docstring content from this method's replacement copied below: Initializes a new rekey attempt.

Only a single recovery key rekeyattempt can take place at a time, and changing the parameters of a rekey requires canceling and starting a new rekey, which will also provide a new nonce.

**Supported methods:** PUT: /sys/rekey/init. Produces: 204 (empty body) PUT: /sys/rekey-recovery-key/init. Produces: 204 (empty body)

**Parameters**

- **secret_shares** (*int*) – Specifies the number of shares to split the master key into.
- **secret_threshold** (*int*) – Specifies the number of shares required to reconstruct the master key. This must be less than or equal to secret_shares.
- **pgp_keys** (*list*) – Specifies an array of PGP public keys used to encrypt the output unseal keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as secret_shares.
- **backup** (*bool*) – Specifies if using PGP-encrypted keys, whether Vault should also store a plaintext backup of the PGP-encrypted keys at core/unseal-keys-backup in the physical storage backend. These can then be retrieved and removed via the sys/rekey/backup endpoint.
- **require_verification** (*bool*) – This turns on verification functionality. When verification is turned on, after successful authorization with the current unseal keys, the new unseal keys are returned but the master key is not actually rotated. The new keys must be provided to authorize the actual rotation of the master key. This ensures that the new keys have been successfully saved and protects against a risk of the keys being lost after rotation but before they can be persisted. This can be used with without pgp_keys, and when used with it, it allows ensuring that the returned keys can be successfully decrypted before committing to the new shares, which the backup functionality does not provide.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON dict of the response.

**Return type** dict | request.Response

**property sys**

Accessor for the Client instance's system backend methods. Provided via the *hvac.api.SystemBackend* class.

**Returns** This Client instance's associated SystemBackend instance.

**Return type** *hvac.api.SystemBackend*

**property token**

**token_role**(*role*)

Returns the named token role.

**Parameters** **role** –

**Returns**

**Return type**

**transit_create_key**(*name*, *convergent_encryption=None*, *derived=None*, *exportable=None*, *key_type=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_create_key'. This method will be removed in version '0.9.0' Please use the 'creat**

Docstring content from this method's replacement copied below: Create a new named encryption key of the specified type.

The values set here cannot be changed after key creation.

**Supported methods:** POST: /{mount_point}/keys/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to create. This is specified as part of the URL.
- **convergent_encryption** (*bool*) – If enabled, the key will support convergent encryption, where the same plaintext creates the same ciphertext. This requires derived to be set to true. When enabled, each encryption(/decryption/rewrap/datakey) operation will derive a nonce value rather than randomly generate it.
- **derived** (*bool*) – Specifies if key derivation is to be used. If enabled, all encrypt/decrypt requests to this named key must provide a context which is used for key derivation.
- **exportable** (*bool*) – Enables keys to be exportable. This allows for all the valid keys in the key ring to be exported. Once set, this cannot be disabled.
- **allow_plaintext_backup** (*bool*) – If set, enables taking backup of named key in the plaintext format. Once set, this cannot be disabled.
- **key_type** (*str | unicode*) – Specifies the type of key to create. The currently-supported types are:
  - **aes256-gcm96**: AES-256 wrapped with GCM using a 96-bit nonce size AEAD
  - **chacha20-poly1305**: ChaCha20-Poly1305 AEAD (symmetric, supports derivation and convergent encryption)
  - **ed25519**: ED25519 (asymmetric, supports derivation).
  - **ecdsa-p256**: ECDSA using the P-256 elliptic curve (asymmetric)
  - **ecdsa-p384**: ECDSA using the P-384 elliptic curve (asymmetric)
  - **ecdsa-p521**: ECDSA using the P-521 elliptic curve (asymmetric)
  - **rsa-2048**: RSA with bit size of 2048 (asymmetric)
  - **rsa-3072**: RSA with bit size of 3072 (asymmetric)
  - **rsa-4096**: RSA with bit size of 4096 (asymmetric)
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**transit_decrypt_data**(*name*, *ciphertext*, *context=None*, *nonce=None*, *batch_input=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_decrypt_data'. This method will be removed in version '0.9.0' Please use the 'de**

Docstring content from this method's replacement copied below: Decrypt the provided ciphertext using the named key.

**Supported methods:** POST: /{mount_point}/decrypt/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to decrypt against. This is specified as part of the URL.
- **ciphertext** (*str | unicode*) – the ciphertext to decrypt.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled.

- **nonce** (*str | unicode*) – Specifies a base64 encoded nonce value used during encryption. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+.
- **batch_input** (*List[dict]*) – Specifies a list of items to be decrypted in a single batch. When this parameter is set, if the parameters 'ciphertext', 'context' and 'nonce' are also set, they will be ignored. Format for the input goes like this: [dict(context="b64_context", ciphertext="b64_plaintext"), ...]
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_delete_key**(*name*, *mount_point='transit'*)

**Call to deprecated function 'transit_delete_key'. This method will be removed in version '0.9.0' Please use the 'delete**
Docstring content from this method's replacement copied below: Delete a named encryption key.

It will no longer be possible to decrypt any data encrypted with the named key. Because this is a potentially catastrophic operation, the deletion_allowed tunable must be set in the key's /config endpoint.

**Supported methods:** DELETE: /{mount_point}/keys/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to delete. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**transit_encrypt_data**(*name*, *plaintext*, *context=None*, *key_version=None*, *nonce=None*, *batch_input=None*, *key_type=None*, *convergent_encryption=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_encrypt_data'. This method will be removed in version '0.9.0' Please use the 'en**
Docstring content from this method's replacement copied below: Encrypt the provided plaintext using the named key.

This path supports the create and update policy capabilities as follows: if the user has the create capability for this endpoint in their policies, and the key does not exist, it will be upserted with default values (whether the key requires derivation depends on whether the context parameter is empty or not). If the user only has update capability and the key does not exist, an error will be returned.

**Supported methods:** POST: /{mount_point}/encrypt/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to encrypt against. This is specified as part of the URL.
- **plaintext** (*str | unicode*) – Specifies base64 encoded plaintext to be encoded.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled for this key.

- **key_version** (*int*) – Specifies the version of the key to use for encryption. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **nonce** (*str | unicode*) – Specifies the base64 encoded nonce value. This must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+. The value must be exactly 96 bits (12 bytes) long and the user must ensure that for any given context (and thus, any given encryption key) this nonce value is never reused.
- **batch_input** (*List[dict]*) – Specifies a list of items to be encrypted in a single batch. When this parameter is set, if the parameters 'plaintext', 'context' and 'nonce' are also set, they will be ignored. The format for the input is: [dict(context="b64_context", plaintext="b64_plaintext"), ...]
- **type** (*str | unicode*) – This parameter is required when encryption key is expected to be created. When performing an upsert operation, the type of key to create.
- **convergent_encryption** (*str | unicode*) – This parameter will only be used when a key is expected to be created. Whether to support convergent encryption. This is only supported when using a key with key derivation enabled and will require all requests to carry both a context and 96-bit (12-byte) nonce. The given nonce will be used in place of a randomly generated nonce. As a result, when the same context and nonce are supplied, the same ciphertext is generated. It is very important when using this mode that you ensure that all nonces are unique for a given context. Failing to do so will severely impact the ciphertext's security.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_export_key**(*name*, *key_type*, *version=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_export_key'. This method will be removed in version '0.9.0' Please use the 'expo**
Docstring content from this method's replacement copied below: Return the named key.

The keys object shows the value of the key for each version. If version is specified, the specific version will be returned. If latest is provided as the version, the current key will be provided. Depending on the type of key, different information may be returned. The key must be exportable to support this operation and the version must still be valid.

**Supported methods:** GET: /{mount_point}/export/{key_type}/{name}(/{version}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to read information about. This is specified as part of the URL.
- **key_type** (*str | unicode*) – Specifies the type of the key to export. This is specified as part of the URL. Valid values are: encryption-key signing-key hmac-key
- **version** (*str | unicode*) – Specifies the version of the key to read. If omitted, all versions of the key will be returned. If the version is set to latest, the current key will be returned.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_generate_data_key**(*name*, *key_type*, *context=None*, *nonce=None*, *bits=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_generate_data_key'. This method will be removed in version '0.9.0' Please use t**
Docstring content from this method's replacement copied below: Generates a new high-entropy key and the value encrypted with the named key.

Optionally return the plaintext of the key as well. Whether plaintext is returned depends on the path; as a result, you can use Vault ACL policies to control whether a user is allowed to retrieve the plaintext value of a key. This is useful if you want an untrusted user or operation to generate keys that are then made available to trusted users.

**Supported methods:** POST: /{mount_point}/datakey/{key_type}/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to use to encrypt the datakey. This is specified as part of the URL.
- **key_type** (*str | unicode*) – Specifies the type of key to generate. If plaintext, the plaintext key will be returned along with the ciphertext. If wrapped, only the ciphertext value will be returned. This is specified as part of the URL.
- **context** (*str | unicode*) – Specifies the key derivation context, provided as a base64-encoded string. This must be provided if derivation is enabled.
- **nonce** (*str | unicode*) – Specifies a nonce value, provided as base64 encoded. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+. The value must be exactly 96 bits (12 bytes) long and the user must ensure that for any given context (and thus, any given encryption key) this nonce value is never reused.
- **bits** (*int*) – Specifies the number of bits in the desired key. Can be 128, 256, or 512.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_generate_hmac**(*name*, *hmac_input*, *key_version=None*, *algorithm=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_generate_hmac'. This method will be removed in version '0.9.0' Please use the '**
Docstring content from this method's replacement copied below: Return the digest of given data using the specified hash algorithm and the named key.

The key can be of any type supported by transit; the raw key will be marshaled into bytes to be used for the HMAC function. If the key is of a type that supports rotation, the latest (current) version will be used.

**Supported methods:** POST: /{mount_point}/hmac/{name}(/{algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to generate hmac against. This is specified as part of the URL.
- **hash_input** – Specifies the base64 encoded input data.

- **key_version** (*int*) – Specifies the version of the key to use for the operation. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_generate_rand_bytes**(*data_bytes=None*, *output_format=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_generate_rand_bytes'. This method will be removed in version '0.9.0' Please use**
Docstring content from this method's replacement copied below: Return high-quality random bytes of the specified length.

**Supported methods:** POST: /{mount_point}/random(/{bytes}). Produces: 200 application/json

**Parameters**

- **n_bytes** (*int*) – Specifies the number of bytes to return. This value can be specified either in the request body, or as a part of the URL.
- **output_format** (*str | unicode*) – Specifies the output encoding. Valid options are hex or base64.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_hash_data**(*hash_input*, *algorithm=None*, *output_format=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_hash_data'. This method will be removed in version '0.9.0' Please use the 'hash_**
Docstring content from this method's replacement copied below: Return the cryptographic hash of given data using the specified algorithm.

**Supported methods:** POST: /{mount_point}/hash(/{algorithm}). Produces: 200 application/json

**Parameters**

- **hash_input** (*str | unicode*) – Specifies the base64 encoded input data.
- **algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **output_format** (*str | unicode*) – Specifies the output encoding. This can be either hex or base64.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_list_keys**(*mount_point='transit'*)

**Call to deprecated function 'transit_list_keys'. This method will be removed in version '0.9.0' Please use the 'list_key**
Docstring content from this method's replacement copied below: List keys (if there are any).

Only the key names are returned (not the actual keys themselves).

An exception is thrown if there are no keys.

**Supported methods:** LIST: /{mount_point}/keys. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_read_key**(*name*, *mount_point='transit'*)

**Call to deprecated function 'transit_read_key'. This method will be removed in version '0.9.0' Please use the 'read_k**
Docstring content from this method's replacement copied below: Read information about a named encryption key.

The keys object shows the creation time of each key version; the values are not the keys themselves. Depending on the type of key, different information may be returned, e.g. an asymmetric key will return its public key in a standard format for the type.

**Supported methods:** GET: /{mount_point}/keys/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to read. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_key request.

**Return type** dict

**transit_rewrap_data**(*name*, *ciphertext*, *context=None*, *key_version=None*, *nonce=None*, *batch_input=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_rewrap_data'. This method will be removed in version '0.9.0' Please use the 'rev**
Docstring content from this method's replacement copied below: Rewrap the provided ciphertext using the latest version of the named key.

Because this never returns plaintext, it is possible to delegate this functionality to untrusted users or scripts.

**Supported methods:** POST: /{mount_point}/rewrap/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to re-encrypt against. This is specified as part of the URL.
- **ciphertext** (*str | unicode*) – Specifies the ciphertext to re-encrypt.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled.
- **key_version** (*int*) – Specifies the version of the key to use for the operation. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.

- **nonce** (*str | unicode*) – Specifies a base64 encoded nonce value used during encryption. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+.
- **batch_input** (*List[dict]*) – Specifies a list of items to be decrypted in a single batch. When this parameter is set, if the parameters 'ciphertext', 'context' and 'nonce' are also set, they will be ignored. Format for the input goes like this: [dict(context="b64_context", ciphertext="b64_plaintext"), ...]
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_rotate_key**(*name*, *mount_point='transit'*)

**Call to deprecated function 'transit_rotate_key'. This method will be removed in version '0.9.0' Please use the 'rotat**
Docstring content from this method's replacement copied below: Rotate the version of the named key.

After rotation, new plaintext requests will be encrypted with the new version of the key. To upgrade ciphertext to be encrypted with the latest version of the key, use the rewrap endpoint. This is only supported with keys that support encryption and decryption operations.

**Supported methods:** POST: /{mount_point}/keys/{name}/rotate. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to read information about. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**transit_sign_data**(*name*, *input_data*, *key_version=None*, *algorithm=None*, *context=None*, *prehashed=None*, *mount_point='transit'*, *signature_algorithm='pss'*)

**Call to deprecated function 'transit_sign_data'. This method will be removed in version '0.9.0' Please use the 'sign_d**
Docstring content from this method's replacement copied below: Return the cryptographic signature of the given data using the named key and the specified hash algorithm.

The key must be of a type that supports signing.

**Supported methods:** POST: /{mount_point}/sign/{name}(/{hash_algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to use for signing. This is specified as part of the URL.
- **hash_input** (*str | unicode*) – Specifies the base64 encoded input data.
- **key_version** (*int*) – Specifies the version of the key to use for signing. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.

- **hash_algorithm** (*str | unicode*) – Specifies the hash algorithm to use for supporting key types (notably, not including ed25519 which specifies its own hash algorithm). This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **context** (*str | unicode*) – Base64 encoded context for key derivation. Required if key derivation is enabled; currently only available with ed25519 keys.
- **prehashed** (*bool*) – Set to true when the input is already hashed. If the key type is rsa-2048 or rsa-4096, then the algorithm used to hash the input should be indicated by the hash_algorithm parameter. Just as the value to sign should be the base64-encoded representation of the exact binary data you want signed, when set, input is expected to be base64-encoded binary hashed data, not hex-formatted. (As an example, on the command line, you could generate a suitable input via openssl dgst -sha256 -binary | base64.)
- **signature_algorithm** (*str | unicode*) – When using a RSA key, specifies the RSA signature algorithm to use for signing. Supported signature types are: pss, pkcs1v15
- **marshaling_algorithm** (*str | unicode*) – Specifies the way in which the signature should be marshaled. This currently only applies to ECDSA keys. Supported types are: asn1, jws
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**transit_update_key**(*name*, *min_decryption_version=None*, *min_encryption_version=None*, *deletion_allowed=None*, *mount_point='transit'*)

**Call to deprecated function 'transit_update_key'. This method will be removed in version '0.9.0' Please use the 'upd**

Docstring content from this method's replacement copied below: Tune configuration values for a given key.

These values are returned during a read operation on the named key.

**Supported methods:** POST: /{mount_point}/keys/{name}/config. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to update configuration for.
- **min_decryption_version** (*int*) – Specifies the minimum version of ciphertext allowed to be decrypted. Adjusting this as part of a key rotation policy can prevent old copies of ciphertext from being decrypted, should they fall into the wrong hands. For signatures, this value controls the minimum version of signature that can be verified against. For HMACs, this controls the minimum version of a key allowed to be used as the key for verification.
- **min_encryption_version** (*int*) – Specifies the minimum version of the key that can be used to encrypt plaintext, sign payloads, or generate HMACs. Must be 0 (which will use the latest version) or a value greater or equal to min_decryption_version.
- **deletion_allowed** (*bool*) – Specifies if the key is allowed to be deleted.
- **exportable** (*bool*) – Enables keys to be exportable. This allows for all the valid keys in the key ring to be exported. Once set, this cannot be disabled.
- **allow_plaintext_backup** (*bool*) – If set, enables taking backup of named key in the plaintext format. Once set, this cannot be disabled.

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**transit_verify_signed_data**(*name*, *input_data*, *algorithm=None*, *signature=None*, *hmac=None*, *context=None*, *prehashed=None*, *mount_point='transit'*, *signature_algorithm='pss'*)

**Call to deprecated function 'transit_verify_signed_data'. This method will be removed in version '0.9.0' Please use t**

Docstring content from this method's replacement copied below: Return whether the provided signature is valid for the given data.

**Supported methods:** POST: /{mount_point}/verify/{name}(/{hash_algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key that was used to generate the signature or HMAC.
- **hash_input** – Specifies the base64 encoded input data.
- **signature** (*str | unicode*) – Specifies the signature output from the /transit/sign function. Either this must be supplied or hmac must be supplied.
- **hmac** (*str | unicode*) – Specifies the signature output from the /transit/hmac function. Either this must be supplied or signature must be supplied.
- **hash_algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **context** (*str | unicode*) – Base64 encoded context for key derivation. Required if key derivation is enabled; currently only available with ed25519 keys.
- **prehashed** (*bool*) – Set to true when the input is already hashed. If the key type is rsa-2048 or rsa-4096, then the algorithm used to hash the input should be indicated by the hash_algorithm parameter.
- **signature_algorithm** (*str | unicode*) – When using a RSA key, specifies the RSA signature algorithm to use for signature verification. Supported signature types are: pss, pkcs1v15
- **marshaling_algorithm** (*str | unicode*) – Specifies the way in which the signature should be marshaled. This currently only applies to ECDSA keys. Supported types are: asn1, jws
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**tune_auth_backend**(*backend_type*, *mount_point=None*, *default_lease_ttl=None*, *max_lease_ttl=None*, *description=None*, *audit_non_hmac_request_keys=None*, *audit_non_hmac_response_keys=None*, *listing_visibility=''*, *passthrough_request_headers=None*)

**Call to deprecated function 'tune_auth_backend'. This method will be removed in version '0.9.0' Please use the 'tun**

Docstring content from this method's replacement copied below: Tune configuration parameters for a given auth path.

This endpoint requires sudo capability on the final path, but the same functionality can be achieved without sudo via sys/mounts/auth/[auth-path]/tune.

**Supported methods:** POST: /sys/auth/{path}/tune. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.
- **default_lease_ttl** (*int*) – Specifies the default time-to-live. If set on a specific auth path, this overrides the global default.
- **max_lease_ttl** (*int*) – The maximum time-to-live. If set on a specific auth path, this overrides the global default.
- **description** (*str | unicode*) – Specifies the description of the mount. This overrides the current stored value, if any.
- **audit_non_hmac_request_keys** (*array*) – Specifies the list of keys that will not be HMAC'd by audit devices in the request data object.
- **audit_non_hmac_response_keys** (*list*) – Specifies the list of keys that will not be HMAC'd by audit devices in the response data object.
- **listing_visibility** (*list*) – Specifies whether to show this mount in the UI-specific listing endpoint. Valid values are "unauth" or "".
- **passthrough_request_headers** (*list*) – List of headers to whitelist and pass from the request to the backend.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

**tune_secret_backend**(*backend_type*, *mount_point=None*, *default_lease_ttl=None*, *max_lease_ttl=None*, *description=None*, *audit_non_hmac_request_keys=None*, *audit_non_hmac_response_keys=None*, *listing_visibility=None*, *passthrough_request_headers=None*)

**Call to deprecated function 'tune_secret_backend'. This method will be removed in version '0.9.0' Please use the 'tu**

Docstring content from this method's replacement copied below: Tune configuration parameters for a given mount point.

**Supported methods:** POST: /sys/mounts/{path}/tune. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.
- **mount_point** (*str*) – The path the associated secret backend is mounted
- **description** (*str*) – Specifies the description of the mount. This overrides the current stored value, if any.
- **default_lease_ttl** (*int*) – Default time-to-live. This overrides the global default. A value of 0 is equivalent to the system default TTL
- **max_lease_ttl** (*int*) – Maximum time-to-live. This overrides the global default. A value of 0 are equivalent and set to the system max TTL.

- **audit_non_hmac_request_keys** (*list*) – Specifies the comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
- **audit_non_hmac_response_keys** (*list*) – Specifies the comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
- **listing_visibility** (*str*) – Speficies whether to show this mount in the UI-specific listing endpoint. Valid values are "unauth" or "".
- **passthrough_request_headers** (*str*) – Comma-separated list of headers to whitelist and pass from the request to the backend.
- **options** (*dict*) – Specifies mount type specific options that are passed to the backend.
  - **version**: <KV> The version of the KV to mount. Set to "2" for mount KV v2.
- **force_no_cache** (*bool*) – Disable caching.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response from the request.

**Return type** request.Response

**unseal**(*key*)

**Call to deprecated function 'unseal'. This method will be removed in version '0.9.0' Please use the 'submit_unseal_k**
Docstring content from this method's replacement copied below: Enter a single master key share to progress the unsealing of the Vault.

If the threshold number of master key shares is reached, Vault will attempt to unseal the Vault. Otherwise, this API must be called multiple times until that threshold is met.

Either the key or reset parameter must be provided; if both are provided, reset takes precedence.

**Supported methods:** PUT: /sys/unseal. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single master key share. This is required unless reset is true.
- **reset** (*bool*) – Specifies if previously-provided unseal keys are discarded and the unseal process is reset.
- **migrate** – Available in 1.0 Beta - Used to migrate the seal from shamir to autoseal or autoseal to shamir. Must be provided on all unseal key calls.

**Type** migrate: bool

**Returns** The JSON response of the request.

**Return type** dict

**unseal_multi**(*keys*)

**Call to deprecated function 'unseal_multi'. This method will be removed in version '0.9.0' Please use the 'submit_un**
Docstring content from this method's replacement copied below: Enter multiple master key share to progress the unsealing of the Vault.

**Parameters**

- **keys** (*List[str]*) – List of master key shares.

- **migrate** – Available in 1.0 Beta - Used to migrate the seal from shamir to autoseal or autoseal to shamir. Must be provided on all unseal key calls.

**Type** migrate: bool

**Returns** The JSON response of the last unseal request.

**Return type** dict

**unseal_reset**()

**Call to deprecated function 'unseal_reset'. This method will be removed in version '0.9.0' Please use the 'submit_uns**
Docstring content from this method's replacement copied below: Enter a single master key share to progress the unsealing of the Vault.

If the threshold number of master key shares is reached, Vault will attempt to unseal the Vault. Otherwise, this API must be called multiple times until that threshold is met.

Either the key or reset parameter must be provided; if both are provided, reset takes precedence.

**Supported methods:** PUT: /sys/unseal. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single master key share. This is required unless reset is true.
- **reset** (*bool*) – Specifies if previously-provided unseal keys are discarded and the unseal process is reset.
- **migrate** – Available in 1.0 Beta - Used to migrate the seal from shamir to autoseal or autoseal to shamir. Must be provided on all unseal key calls.

**Type** migrate: bool

**Returns** The JSON response of the request.

**Return type** dict

**unwrap**(*token=None*)

**Call to deprecated function 'unwrap'. This method will be removed in version '0.9.0' Please use the 'unwrap' metho**
Docstring content from this method's replacement copied below: Return the original response inside the given wrapping token.

Unlike simply reading cubbyhole/response (which is deprecated), this endpoint provides additional validation checks on the token, returns the original value on the wire rather than a JSON string representation of it, and ensures that the response is properly audit-logged.

**Supported methods:** POST: /sys/wrapping/unwrap. Produces: 200 application/json

**Parameters** **token** (*str | unicode*) – Specifies the wrapping token ID. This is required if the client token is not the wrapping token. Do not use the wrapping token in both locations.

**Returns** The JSON response of the request.

**Return type** dict

**update_userpass_password**(*username*, *password*, *mount_point='userpass'*)
POST /auth/<mount point>/users/<username>/password

**Parameters**

- **username** –

- **password** –
- **mount_point** –

**Returns**

**Return type**

**update_userpass_policies**(*username*, *policies*, *mount_point='userpass'*)
POST /auth/<mount point>/users/<username>/policies

**Parameters**

- **username** –
- **policies** –
- **mount_point** –

**Returns**

**Return type**

**property url**

**static urljoin**(**args*, ***kwargs*)

**Call to deprecated function 'urljoin'. This method will be removed in version '0.8.0' Please use the 'urljoin' method**
Docstring content from this method's replacement copied below: Joins given arguments into a url. Trailing and leading slashes are stripped for each argument.

**Parameters** **args** (*str | unicode*) – Multiple parts of a URL to be combined into one string.

**Returns** Full URL combining all provided arguments

**Return type** str | unicode

**write**(*path*, *wrap_ttl=None*, ***kwargs*)
POST /<path>

**Parameters**

- **path** –
- **wrap_ttl** –
- **kwargs** –

**Returns**

**Return type**

## 4.2 hvac.api

Collection of Vault API endpoint classes.

**Classes**

| | |
|---|---|
| *AuthMethods*(adapter) | Auth Methods. |
| *SecretsEngines*(adapter) | Secrets Engines. |
| *SystemBackend*(adapter) | |

continues on next page

Table 4 – continued from previous page

| | |
|---|---|
| *VaultApiBase*(adapter) | Base class for API endpoints. |
| *VaultApiCategory*(adapter) | Base class for API categories. |

**class** hvac.api.**AuthMethods**(*adapter*)

Bases: hvac.api.vault_api_category.VaultApiCategory

Auth Methods. **Attributes**

| | |
|---|---|
| *implemented_classes* | Built-in mutable sequence. |
| *unimplemented_classes* | Built-in mutable sequence. |

**implemented_classes = [<class 'hvac.api.auth_methods.approle.AppRole'>, <class 'hvac.ap**

**unimplemented_classes = ['AppId', 'AliCloud', 'Token']**

**class** hvac.api.**SecretsEngines**(*adapter*)

Bases: hvac.api.vault_api_category.VaultApiCategory

Secrets Engines. **Attributes**

| | |
|---|---|
| *implemented_classes* | Built-in mutable sequence. |
| *unimplemented_classes* | Built-in mutable sequence. |

**implemented_classes = [<class 'hvac.api.secrets_engines.aws.Aws'>, <class 'hvac.api.se**

**unimplemented_classes = ['AliCloud', 'Azure', 'GcpKms', 'Nomad', 'Ssh', 'TOTP', 'Cassa**

**class** hvac.api.**SystemBackend**(*adapter*)

Bases: hvac.api.vault_api_category.VaultApiCategory, hvac.api.system_backend.audit.Audit, hvac.api.system_backend.auth.Auth, hvac.api.system_backend.capabilities.Capabilities, hvac.api.system_backend.health.Health, hvac.api.system_backend.init.Init, hvac.api.system_backend.key.Key, hvac.api.system_backend.leader.Leader, hvac.api.system_backend.lease.Lease, hvac.api.system_backend.mount.Mount, hvac.api.system_backend.namespace.Namespace, hvac.api.system_backend.policy.Policy, hvac.api.system_backend.raft.Raft, hvac.api.system_backend.seal.Seal, hvac.api.system_backend.wrapping.Wrapping **Methods**

| | |
|---|---|
| *__init__*(adapter) | API Category class constructor. |

**Attributes**

| | |
|---|---|
| *implemented_classes* | Built-in mutable sequence. |
| *unimplemented_classes* | Built-in mutable sequence. |

**__init__**(*adapter*)

API Category class constructor.

**Parameters** **adapter** (hvac.adapters.Adapter) – Instance of *hvac.adapters.Adapter*; used for performing HTTP requests.

**implemented_classes = [<class 'hvac.api.system_backend.audit.Audit'>, <class 'hvac.api**

**unimplemented_classes = []**

**class** hvac.api.**VaultApiBase**(*adapter*)
Bases: object

Base class for API endpoints. **Methods**

| | |
|---|---|
| *__init__*(adapter) | Default api class constructor. |

**__init__**(*adapter*)
Default api class constructor.

**Parameters** **adapter** (*hvac.adapters.Adapter*) – Instance of *hvac.adapters.Adapter*; used for performing HTTP requests.

**class** hvac.api.**VaultApiCategory**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

Base class for API categories. **Methods**

| | |
|---|---|
| *__init__*(adapter) | API Category class constructor. |
| *get_private_attr_name*(class_name) | Helper method to prepend a leading underscore to a provided class name. |

**Attributes**

| | |
|---|---|
| *adapter* | Retrieve the adapter instance under the "_adapter" property in use by this class. |
| *implemented_classes* | List of implemented classes under this category. |
| *unimplemented_classes* | List of known unimplemented classes under this category. |

**__init__**(*adapter*)
API Category class constructor.

**Parameters** **adapter** (*hvac.adapters.Adapter*) – Instance of *hvac.adapters.Adapter*; used for performing HTTP requests.

**property adapter**
Retrieve the adapter instance under the "_adapter" property in use by this class.

**Returns** The adapter instance in use by this class.

**Return type** *hvac.adapters.Adapter*

**static get_private_attr_name**(*class_name*)
Helper method to prepend a leading underscore to a provided class name.

**Parameters** **class_name** (*str|unicode*) – Name of a class under this category.

**Returns** The private attribute label for the provided class.

**Return type** str

**abstract property implemented_classes**
List of implemented classes under this category.

**Returns** List of implemented classes under this category.

**Return type** List[*hvac.api.VaultApiBase*]

**property unimplemented_classes**
List of known unimplemented classes under this category.

**Returns** List of known unimplemented classes under this category.

**Return type** List[str]

## 4.3 hvac.api.auth_methods

Collection of classes for various Vault auth methods.

**Classes**

| | |
|---|---|
| *AuthMethods*(adapter) | Auth Methods. |
| *AppRole*(adapter) | USERPASS Auth Method (API). |
| *Azure*(adapter) | Azure Auth Method (API). |
| *Gcp*(adapter) | Google Cloud Auth Method (API). |
| *Github*(adapter) | GitHub Auth Method (API). |
| *JWT*(adapter) | JWT auth method which can be used to authenticate with Vault by providing a JWT. |
| *Kubernetes*(adapter) | Kubernetes Auth Method (API). |
| *Ldap*(adapter) | LDAP Auth Method (API). |
| *Userpass*(adapter) | USERPASS Auth Method (API). |
| *Mfa*(adapter) | Multi-factor authentication Auth Method (API). |
| *OIDC*(adapter) | OIDC auth method which can be used to authenticate with Vault using OIDC. |
| *Okta*(adapter) | Okta Auth Method (API). |
| *Radius*(adapter) | RADIUS Auth Method (API). |
| *Aws*(adapter) | AWS Auth Method (API). |
| *Cert*(adapter) | Cert Auth Method (API). |

**class** hvac.api.auth_methods.**AppRole**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

USERPASS Auth Method (API). Reference: https://www.vaultproject.io/api-docs/auth/approle/index.html

**Methods**

| | |
|---|---|
| *create_custom_secret_id*(role_name, secret_id) | Generates and issues a new Secret ID on a role in the auth method. |
| *create_or_update_approle*(role_name[, ...]) | Create/update approle. |
| *delete_role*(role_name[, mount_point]) | Delete role in the auth method. |
| *destroy_secret_id*(role_name, secret_id[, ...]) | Destroys a Secret ID for a role in the auth method. |
| *destroy_secret_id_accessor*(role_name, ...[, ...]) | Destroys a Secret ID for a role in the auth method. |
| *generate_secret_id*(role_name[, metadata, ...]) | Generates and issues a new Secret ID on a role in the auth method. |
| *list_roles*([mount_point]) | List existing roles created in the auth method. |
| *list_secret_id_accessors*(role_name[, ...]) | Lists accessors of all issued Secret IDs for a role in the auth method. |
| *login*(role_id[, secret_id, use_token, ...]) | Login with APPROLE credentials. |

continues on next page

Table 13 – continued from previous page

| | |
|---|---|
| read_role(role_name[, mount_point]) | Read role in the auth method. |
| read_role_id(role_name[, mount_point]) | Reads the Role ID of a role in the auth method. |
| read_secret_id(role_name, secret_id[, ...]) | Read the properties of a Secret ID for a role in the auth method. |
| read_secret_id_accessor(role_name, ...[, ...]) | Read the properties of a Secret ID for a role in the auth method. |
| update_role_id(role_name, role_id[, mount_point]) | Updates the Role ID of a role in the auth method. |

**create_custom_secret_id**(*role_name*, *secret_id*, *metadata=None*, *cidr_list=None*, *token_bound_cidrs=None*, *mount_point='approle'*)

Generates and issues a new Secret ID on a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/custom-secret-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **secret_id** (*str | unicode*) – The Secret ID to read.
- **metadata** (*dict*) – Metadata to be tied to the Secret ID.
- **cidr_list** (*list*) – Blocks of IP addresses which can perform login operations.
- **token_bound_cidrs** (*list*) – Blocks of IP addresses which can authenticate successfully.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**create_or_update_approle**(*role_name*, *bind_secret_id=None*, *secret_id_bound_cidrs=None*, *secret_id_num_uses=None*, *secret_id_ttl=None*, *enable_local_secret_ids=None*, *token_ttl=None*, *token_max_ttl=None*, *token_policies=None*, *token_bound_cidrs=None*, *token_explicit_max_ttl=None*, *token_no_default_policy=None*, *token_num_uses=None*, *token_period=None*, *token_type=None*, *mount_point='approle'*)

Create/update approle.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}. Produces: 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the approle.
- **bind_secret_id** (*bool*) – Require secret_id to be presented when logging in using this approle.
- **secret_id_bound_cidrs** (*list*) – Blocks of IP addresses which can perform login operations.
- **secret_id_num_uses** (*int*) – Number of times any secret_id can be used to fetch a token. A value of zero allows unlimited uses.

- **secret_id_ttl** (*str | unicode*) – Duration after which a secret_id expires. This can be specified as an integer number of seconds or as a duration value like "5m".
- **enable_local_secret_ids** (*bool*) – Secret IDs generated using role will be cluster local.
- **token_ttl** (*str | unicode*) – Incremental lifetime for generated tokens. This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_max_ttl** (*str | unicode*) – Maximum lifetime for generated tokens: This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_policies** (*list*) – List of policies to encode onto generated tokens.
- **token_bound_cidrs** (*list*) – Blocks of IP addresses which can authenticate successfully.
- **token_explicit_max_ttl** (*str | unicode*) – If set, will encode an explicit max TTL onto the token. This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_no_default_policy** (*bool*) – Do not add the default policy to generated tokens, use only tokens specified in token_policies.
- **token_num_uses** (*int*) – Maximum number of times a generated token may be used. A value of zero allows unlimited uses.
- **token_period** (*str | unicode*) – The period, if any, to set on the token. This can be specified as an integer number of seconds or as a duration value like "5m".
- **token_type** (*str | unicode*) – The type of token that should be generated, can be "service", "batch", or "default".
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**delete_role**(*role_name*, *mount_point='approle'*)
Delete role in the auth method.

**Supported methods:** DELETE: /auth/{mount_point}/role/{role_name}. Produces: 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**destroy_secret_id**(*role_name*, *secret_id*, *mount_point='approle'*)
Destroys a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id/destroy. Produces 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id** (*str | unicode*) – The Secret ID to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**destroy_secret_id_accessor**(*role_name*, *secret_id_accessor*, *mount_point='approle'*)
Destroys a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id-accessor/destroy. Produces: 204 (empty body)

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id_accessor** (*str | unicode*) – The accessor for the Secret ID to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**generate_secret_id**(*role_name*, *metadata=None*, *cidr_list=None*, *token_bound_cidrs=None*, *mount_point='approle'*)

Generates and issues a new Secret ID on a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **metadata** (*dict*) – Metadata to be tied to the Secret ID.
- **cidr_list** (*list*) – Blocks of IP addresses which can perform login operations.
- **token_bound_cidrs** (*list*) – Blocks of IP addresses which can authenticate successfully.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**list_roles**(*mount_point='approle'*)

List existing roles created in the auth method.

**Supported methods:** LIST: /auth/{mount_point}/role. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the list_roles request.

**Return type** dict

**list_secret_id_accessors**(*role_name*, *mount_point='approle'*)

Lists accessors of all issued Secret IDs for a role in the auth method.

**Supported methods:** LIST: /auth/{mount_point}/role/{role_name}/secret-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**login**(*role_id*, *secret_id=None*, *use_token=True*, *mount_point='approle'*)
Login with APPROLE credentials.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **role_id** (*str | unicode*) – Role ID of the role.
- **secret_id** (*str | unicode*) – Secret ID of the role.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**read_role**(*role_name*, *mount_point='approle'*)
Read role in the auth method.

**Supported methods:** GET: /auth/{mount_point}/role/{role_name}. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role request.

**Return type** dict

**read_role_id**(*role_name*, *mount_point='approle'*)
Reads the Role ID of a role in the auth method.

**Supported methods:** GET: /auth/{mount_point}/role/{role_name}/role-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**read_secret_id**(*role_name*, *secret_id*, *mount_point='approle'*)
Read the properties of a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id/lookup. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id** (*str | unicode*) – The Secret ID to read.

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**read_secret_id_accessor**(*role_name*, *secret_id_accessor*, *mount_point='approle'*)
Read the properties of a Secret ID for a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/secret-id-accessor/lookup. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role
- **secret_id_accessor** (*str | unicode*) – The accessor for the Secret ID to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**update_role_id**(*role_name*, *role_id*, *mount_point='approle'*)
Updates the Role ID of a role in the auth method.

**Supported methods:** POST: /auth/{mount_point}/role/{role_name}/role-id. Produces: 200 application/json

**Parameters**

- **role_name** (*str | unicode*) – The name for the role.
- **role_id** (*str | unicode*) – New value for the Role ID.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_role_id request.

**Return type** dict

**class** hvac.api.auth_methods.**AuthMethods**(*adapter*)
Bases: hvac.api.vault_api_category.VaultApiCategory

Auth Methods. **Attributes**

| | |
|---|---|
| *implemented_classes* | Built-in mutable sequence. |
| *unimplemented_classes* | Built-in mutable sequence. |

**implemented_classes = [<class 'hvac.api.auth_methods.approle.AppRole'>, <class 'hvac.a**

**unimplemented_classes = ['AppId', 'AliCloud', 'Token']**

**class** hvac.api.auth_methods.**Aws**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

AWS Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/aws/index.html **Methods**

| | |
|---|---|
| configure([max_retries, access_key, ...]) | Configure the credentials required to perform API calls to AWS as well as custom endpoints to talk to AWS API. |
| configure_identity_integration([iam_alias, ...]) | Configure the way that Vault interacts with the Identity store. |
| configure_identity_whitelist_tidy([...]) | Configure the periodic tidying operation of the whitelisted identity entries. |
| configure_role_tag_blacklist_tidy([...]) | Configure the periodic tidying operation of the blacklisted role tag entries. |
| create_certificate_configuration(cert_name, ...) | Register AWS public key to be used to verify the instance identity documents. |
| create_role(role[, auth_type, bound_ami_id, ...]) | Register a role in the method. |
| create_role_tags(role[, policies, max_ttl, ...]) | Create a role tag on the role, which helps in restricting the capabilities that are set on the role. |
| create_sts_role(account_id, sts_role[, ...]) | Allow the explicit association of STS roles to satellite AWS accounts (i.e. |
| delete_blacklist_tags(role_tag[, mount_point]) | Deletes a blacklisted role tag |
| delete_certificate_configuration(cert_name) | Remove previously configured AWS public key. |
| delete_config([mount_point]) | Delete previously configured AWS access credentials, |
| delete_identity_whitelist_entries(instance_id) | Deletes a cache of the successful login from an instance |
| delete_identity_whitelist_tidy([mount_point]) | Delete previously configured periodic whitelist tidying settings. |
| delete_role(role[, mount_point]) | Deletes the previously registered role |
| delete_role_tag_blacklist_tidy([mount_point]) | Delete previously configured periodic blacklist tidying settings. |
| delete_sts_role(account_id[, mount_point]) | Delete a previously configured AWS account/STS role association. |
| ec2_login(pkcs7[, nonce, role, use_token, ...]) | Retrieve a Vault token using an AWS authentication method mount's EC2 role. |
| iam_login(access_key, secret_key[, ...]) | Fetch a token |
| list_blacklist_tags([mount_point]) | Lists all the role tags that are blacklisted |
| list_certificate_configurations([mount_point]) | List AWS public certificates that are registered with the method. |
| list_identity_whitelist([mount_point]) | Lists all the instance IDs that are in the whitelist of successful logins |
| list_roles([mount_point]) | Lists all the roles that are registered with the method |
| list_sts_roles([mount_point]) | List AWS Account IDs for which an STS role is registered. |
| place_role_tags_in_blacklist(role_tag[, ...]) | Places a valid role tag in a blacklist |
| read_certificate_configuration(cert_name, ...]) | Return previously configured AWS public key. |
| read_config([mount_point]) | Read previously configured AWS access credentials. |
| read_identity_integration([mount_point]) | Return previously configured identity integration configuration. |
| read_identity_whitelist(instance_id[, ...]) | Returns an entry in the whitelist. |

continues on next page

Table 15 – continued from previous page

| | |
|---|---|
| *read_identity_whitelist_tidy*([mount_point]) | Read previously configured periodic whitelist tidying settings. |
| *read_role*(role[, mount_point]) | Returns the previously registered role configuration |
| *read_role_tag_blacklist*(role_tag[, mount_point]) | Returns the blacklist entry of a previously blacklisted role tag |
| *read_role_tag_blacklist_tidy*([mount_point]) | Read previously configured periodic blacklist tidying settings. |
| *read_sts_role*(account_id[, mount_point]) | Return previously configured STS role. |
| *tidy_blacklist_tags*([saftey_buffer, mount_point]) | Cleans up the entries in the blacklist based on expiration time on the entry and safety_buffer |
| *tidy_identity_whitelist_entries*([...]) | Cleans up the entries in the whitelist based on expiration time and safety_buffer |

**configure**(*max_retries=None*, *access_key=None*, *secret_key=None*, *endpoint=None*, *iam_endpoint=None*, *sts_endpoint=None*, *iam_server_id_header_value=None*, *mount_point='aws'*)

Configure the credentials required to perform API calls to AWS as well as custom endpoints to talk to AWS API.

The instance identity document fetched from the PKCS#7 signature will provide the EC2 instance ID. The credentials configured using this endpoint will be used to query the status of the instances via DescribeInstances API. If static credentials are not provided using this endpoint, then the credentials will be retrieved from the environment variables AWS_ACCESS_KEY, AWS_SECRET_KEY and AWS_REGION respectively. If the credentials are still not found and if the method is configured on an EC2 instance with metadata querying capabilities, the credentials are fetched automatically

**Supported methods:** POST: /auth/{mount_point}/config Produces: 204 (empty body)

**Parameters**

- **max_retries** (*int*) – Number of max retries the client should use for recoverable errors. The default (-1) falls back to the AWS SDK's default behavior
- **access_key** (*str | unicode*) – AWS Access key with permissions to query AWS APIs. The permissions required depend on the specific configurations. If using the iam auth method without inferencing, then no credentials are necessary. If using the ec2 auth method or using the iam auth method with inferencing, then these credentials need access to ec2:DescribeInstances. If additionally a bound_iam_role is specified, then these credentials also need access to iam:GetInstanceProfile. If, however, an alternate sts configuration is set for the target account, then the credentials must be permissioned to call sts:AssumeRole on the configured role, and that role must have the permissions described here
- **secret_key** (*str | unicode*) – AWS Secret key with permissions to query AWS APIs
- **endpoint** (*str | unicode*) – URL to override the default generated endpoint for making AWS EC2 API calls
- **iam_endpoint** (*str | unicode*) – URL to override the default generated endpoint for making AWS IAM API calls
- **sts_endpoint** (*str | unicode*) – URL to override the default generated endpoint for making AWS STS API calls
- **iam_server_id_header_value** (*str | unicode*) – The value to require in the X-Vault-AWS-IAM-Server-ID header as part of GetCallerIdentity requests that are used

in the iam auth method. If not set, then no value is required or validated. If set, clients must include an X-Vault-AWS-IAM-Server-ID header in the headers of login requests, and further this header must be among the signed headers validated by AWS. This is to protect against different types of replay attacks, for example a signed request sent to a dev server being resent to a production server

- **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**configure_identity_integration**(*iam_alias=None*, *ec2_alias=None*, *mount_point='aws'*)

Configure the way that Vault interacts with the Identity store.

The default (as of Vault 1.0.3) is role_id for both values.

**Supported methods:** POST: /auth/{mount_point}/config/identity Produces: 204 (empty body)

**Parameters**

- **iam_alias** (*str | unicode*) – How to generate the identity alias when using the iam auth method. Valid choices are role_id, unique_id, and full_arn When role_id is selected, the randomly generated ID of the role is used. When unique_id is selected, the IAM Unique ID of the IAM principal (either the user or role) is used as the identity alias name. When full_arn is selected, the ARN returned by the sts:GetCallerIdentity call is used as the alias name. This is either arn:aws:iam::<account_id>:user/<optional_path/><user_name> or arn:aws:sts::<account_id>:assumed-role/<role_name_without_path>/<role_session_name>. Note: if you select full_arn and then delete and recreate the IAM role, Vault won't be aware and any identity aliases set up for the role name will still be valid
- **ec2_alias** (*str | unicode*) – Configures how to generate the identity alias when using the ec2 auth method. Valid choices are role_id, instance_id, and image_id. When role_id is selected, the randomly generated ID of the role is used. When instance_id is selected, the instance identifier is used as the identity alias name. When image_id is selected, AMI ID of the instance is used as the identity alias name
- **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request

**Return type** request.Response

**configure_identity_whitelist_tidy**(*safety_buffer=None*, *disable_periodic_tidy=None*, *mount_point='aws'*)

Configure the periodic tidying operation of the whitelisted identity entries.

**Parameters**

- **safety_buffer** (*str*) – The amount of extra time that must have passed beyond the roletag expiration, before it is removed from the method storage.
- **disable_periodic_tidy** (*bool*) – If set to 'true', disables the periodic tidying of the identity-whitelist/<instance_id> entries.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**configure_role_tag_blacklist_tidy**(*safety_buffer=None*, *disable_periodic_tidy=None*, *mount_point='aws'*)

Configure the periodic tidying operation of the blacklisted role tag entries.

**Parameters**

- **safety_buffer** (*str*) – The amount of extra time that must have passed beyond the roletag expiration, before it is removed from the method storage.
- **disable_periodic_tidy** (*bool*) – If set to 'true', disables the periodic tidying of the roletag-blacklist/<instance_id> entries.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_certificate_configuration**(*cert_name*, *aws_public_cert*, *document_type=None*, *mount_point='aws'*)

Register AWS public key to be used to verify the instance identity documents.

While the PKCS#7 signature of the identity documents have DSA digest, the identity signature will have RSA digest, and hence the public keys for each type varies respectively. Indicate the type of the public key using the "type" parameter

**Supported methods:** POST: /auth/{mount_point}/config/certificate/:cert_name Produces: 204 (empty body)

**Parameters**

- **cert_name** (*string | unicode*) – Name of the certificate
- **aws_public_cert** – Base64 encoded AWS Public key required to verify PKCS7 signature of the EC2 instance metadata
- **document_type** (*string | unicode*) – Takes the value of either "pkcs7" or "identity", indicating the type of document which can be verified using the given certificate
- **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request

**Return type** request.Response

**create_role**(*role*, *auth_type=None*, *bound_ami_id=None*, *bound_account_id=None*, *bound_region=None*, *bound_vpc_id=None*, *bound_subnet_id=None*, *bound_iam_role_arn=None*, *bound_iam_instance_profile_arn=None*, *bound_ec2_instance_id=None*, *role_tag=None*, *bound_iam_principal_arn=None*, *inferred_entity_type=None*, *inferred_aws_region=None*, *resolve_aws_unique_ids=None*, *ttl=None*, *max_ttl=None*, *period=None*, *policies=None*, *allow_instance_migration=None*, *disallow_reauthentication=None*, *mount_point='aws'*)

Register a role in the method.

**Parameters**

- **role** –
- **auth_type** –
- **bound_ami_id** –
- **bound_account_id** –
- **bound_region** –

- **bound_vpc_id** –
- **bound_subnet_id** –
- **bound_iam_role_arn** –
- **bound_iam_instance_profile_arn** –
- **bound_ec2_instance_id** –
- **role_tag** –
- **bound_iam_principal_arn** –
- **inferred_entity_type** –
- **inferred_aws_region** –
- **resolve_aws_unique_ids** –
- **ttl** –
- **max_ttl** –
- **period** –
- **policies** –
- **allow_instance_migration** –
- **disallow_reauthentication** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_role_tags**(*role*, *policies=None*, *max_ttl=None*, *instance_id=None*, *allow_instance_migration=None*, *disallow_reauthentication=None*, *mount_point='aws'*)

Create a role tag on the role, which helps in restricting the capabilities that are set on the role.

Role tags are not tied to any specific ec2 instance unless specified explicitly using the instance_id parameter. By default, role tags are designed to be used across all instances that satisfies the constraints on the role. Regardless of which instances have role tags on them, capabilities defined in a role tag must be a strict subset of the given role's capabilities. Note that, since adding and removing a tag is often a widely distributed privilege, care needs to be taken to ensure that the instances are attached with correct tags to not let them gain more privileges than what were intended. If a role tag is changed, the capabilities inherited by the instance will be those defined on the new role tag. Since those must be a subset of the role capabilities, the role should never provide more capabilities than any given instance can be allowed to gain in a worst-case scenario

**Parameters**

- **role** (*str*) – Name of the role.
- **policies** (*list*) – Policies to be associated with the tag. If set, must be a subset of the role's policies. If set, but set to an empty value, only the 'default' policy will be given to issued tokens.
- **max_ttl** (*str*) – The maximum allowed lifetime of tokens issued using this role.
- **instance_id** (*str*) – Instance ID for which this tag is intended for. If set, the created tag can only be used by the instance with the given ID.

- **disallow_reauthentication** (*bool*) – If set, only allows a single token to be granted per instance ID. This can be cleared with the auth/aws/identity-whitelist endpoint. Defaults to 'false'. Mutually exclusive with allow_instance_migration.
- **allow_instance_migration** (*bool*) – If set, allows migration of the underlying instance where the client resides. This keys off of pendingTime in the metadata document, so essentially, this disables the client nonce check whenever the instance is migrated to a new host and pendingTime is newer than the previously-remembered time. Use with caution. Defaults to 'false'. Mutually exclusive with disallow_reauthentication.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The create role tag response.

**Return type** dict

**create_sts_role**(*account_id*, *sts_role*, *mount_point='aws'*)

**Allow the explicit association of STS roles to satellite AWS accounts (i.e. those which are not the** account in which the Vault server is running.)

Vault will use credentials obtained by assuming these STS roles when validating IAM principals or EC2 instances in the particular AWS account

**Supported methods:** POST: /v1/auth/{mount_point}/config/sts/:account_id Produces: 204 (empty body)

**Parameters**

- **account_id** (*str*) – AWS account ID to be associated with STS role. If set, Vault will use assumed credentials to verify any login attempts from EC2 instances in this account.
- **sts_role** (*str*) – AWS ARN for STS role to be assumed when interacting with the account specified. The Vault server must have permissions to assume this role.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_blacklist_tags**(*role_tag*, *mount_point='aws'*)

Deletes a blacklisted role tag

**Parameters**

- **role_tag** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_certificate_configuration**(*cert_name*, *mount_point='aws'*)

Remove previously configured AWS public key.

**Supported methods:** DELETE: /auth/{mount_point}/config/certificate/:cert_name Produces: 204 (empty body)

**Parameters**

- **cert_name** (*str | unicode*) – Name of the certificate
- **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request

**Return type** request.Response

**delete_config**(*mount_point='aws'*)
Delete previously configured AWS access credentials,

**Supported methods:** DELETE: /auth/{mount_point}/config Produces: 204 (empty body)

**Parameters** **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_identity_whitelist_entries**(*instance_id*, *mount_point='aws'*)
Deletes a cache of the successful login from an instance

**Parameters**

- **instance_id** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_identity_whitelist_tidy**(*mount_point='aws'*)
Delete previously configured periodic whitelist tidying settings.

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*role*, *mount_point='aws'*)
Deletes the previously registered role

**Parameters**

- **role** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role_tag_blacklist_tidy**(*mount_point='aws'*)
Delete previously configured periodic blacklist tidying settings.

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_sts_role**(*account_id*, *mount_point='aws'*)
Delete a previously configured AWS account/STS role association.

**Parameters**

- **account_id** –

- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**ec2_login**(*pkcs7*, *nonce=None*, *role=None*, *use_token=True*, *mount_point='aws'*)
Retrieve a Vault token using an AWS authentication method mount's EC2 role.

**Parameters**

- **pkcs7** (*str*) – PKCS7 signature of the identity document with all newline characters removed.
- **nonce** (*str*) – The nonce to be used for subsequent login requests.
- **role** (*str*) – Name of the role against which the login is being attempted.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**iam_login**(*access_key*, *secret_key*, *session_token=None*, *header_value=None*, *role=None*, *use_token=True*, *region='us-east-1'*, *mount_point='aws'*)
Fetch a token

This endpoint verifies the pkcs7 signature of the instance identity document or the signature of the signed GetCallerIdentity request. With the ec2 auth method, or when inferring an EC2 instance, verifies that the instance is actually in a running state. Cross checks the constraints defined on the role with which the login is being performed. With the ec2 auth method, as an alternative to pkcs7 signature, the identity document along with its RSA digest can be supplied to this endpoint

**Parameters**

- **role** (*str*) – Name of the role against which the login is being attempted.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_blacklist_tags**(*mount_point='aws'*)
Lists all the role tags that are blacklisted

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_certificate_configurations**(*mount_point='aws'*)
List AWS public certificates that are registered with the method.

**Supported methods** LIST: /auth/{mount_point}/config/certificates Produces: 200 application/json

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_identity_whitelist**(*mount_point='aws'*)
Lists all the instance IDs that are in the whitelist of successful logins

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_roles**(*mount_point='aws'*)
Lists all the roles that are registered with the method

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_sts_roles**(*mount_point='aws'*)
List AWS Account IDs for which an STS role is registered.

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**place_role_tags_in_blacklist**(*role_tag*, *mount_point='aws'*)
Places a valid role tag in a blacklist

This ensures that the role tag cannot be used by any instance to perform a login operation again. Note that if the role tag was previously used to perform a successful login, placing the tag in the blacklist does not invalidate the already issued token

**Parameters**

- **role_tag** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_certificate_configuration**(*cert_name*, *mount_point='aws'*)
Return previously configured AWS public key.

**Supported methods:** GET: /v1/auth/{mount_point}/config/certificate/:cert_name Produces: 200 application/json

**Parameters**

- **cert_name** (*str | unicode*) – Name of the certificate
- **mount_point** – The path the AWS auth method was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**read_config**(*mount_point='aws'*)
Read previously configured AWS access credentials.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**read_identity_integration**(*mount_point='aws'*)
Return previously configured identity integration configuration.

**Supported methods:** GET: /auth/{mount_point}/config/identity. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The path the AWS auth method was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**read_identity_whitelist**(*instance_id*, *mount_point='aws'*)
Returns an entry in the whitelist. An entry will be created/updated by every successful login

**Parameters**

- **instance_id** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_identity_whitelist_tidy**(*mount_point='aws'*)
Read previously configured periodic whitelist tidying settings.

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_role**(*role*, *mount_point='aws'*)
Returns the previously registered role configuration

**Parameters**

- **role** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_role_tag_blacklist**(*role_tag*, *mount_point='aws'*)
Returns the blacklist entry of a previously blacklisted role tag

**Parameters**

- **role_tag** –

- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_role_tag_blacklist_tidy**(*mount_point='aws'*)
Read previously configured periodic blacklist tidying settings.

**Parameters** **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_sts_role**(*account_id*, *mount_point='aws'*)
Return previously configured STS role.

**Parameters**

- **account_id** (*str*) – AWS account ID that has been previously associated with STS role.
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**tidy_blacklist_tags**(*saftey_buffer='72h'*, *mount_point='aws'*)
Cleans up the entries in the blacklist based on expiration time on the entry and safety_buffer

**Parameters**

- **saftey_buffer** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**tidy_identity_whitelist_entries**(*saftey_buffer='72h'*, *mount_point='aws'*)
Cleans up the entries in the whitelist based on expiration time and safety_buffer

**Parameters**

- **saftey_buffer** –
- **mount_point** (*str*) – The path the AWS auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**class** hvac.api.auth_methods.**Azure**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

Azure Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/azure/index.html **Methods**

| | |
|---|---|
| *configure*(tenant_id, resource[, ...]) | Configure the credentials required for the plugin to perform API calls to Azure. |
| *create_role*(name[, policies, ttl, max_ttl, ...]) | Create a role in the method. |

continues on next page

Table 16 – continued from previous page

| | |
|---|---|
| delete_config([mount_point]) | Delete the previously configured Azure config and credentials. |
| delete_role(name[, mount_point]) | Delete the previously registered role. |
| list_roles([mount_point]) | List all the roles that are registered with the plugin. |
| login(role, jwt[, subscription_id, ...]) | Fetch a token. |
| read_config([mount_point]) | Return the previously configured config, including credentials. |
| read_role(name[, mount_point]) | Read the previously registered role configuration. |

**configure**(*tenant_id*, *resource*, *environment=None*, *client_id=None*, *client_secret=None*, *mount_point='azure'*)

Configure the credentials required for the plugin to perform API calls to Azure.

These credentials will be used to query the metadata about the virtual machine.

**Supported methods:** POST: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **tenant_id** (*str | unicode*) – The tenant id for the Azure Active Directory organization.
- **resource** (*str | unicode*) – The configured URL for the application registered in Azure Active Directory.
- **environment** (*str | unicode*) – The Azure cloud environment. Valid values: AzurePublicCloud, AzureUSGovernmentCloud, AzureChinaCloud, AzureGermanCloud.
- **client_id** (*str | unicode*) – The client id for credentials to query the Azure APIs. Currently read permissions to query compute resources are required.
- **client_secret** (*str | unicode*) – The client secret for credentials to query the Azure APIs.
- **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_role**(*name*, *policies=None*, *ttl=None*, *max_ttl=None*, *period=None*, *bound_service_principal_ids=None*, *bound_group_ids=None*, *bound_locations=None*, *bound_subscription_ids=None*, *bound_resource_groups=None*, *bound_scale_sets=None*, *num_uses=None*, *mount_point='azure'*)

Create a role in the method.

Role types have specific entities that can perform login operations against this endpoint. Constraints specific to the role type must be set on the role. These are applied to the authenticated entities attempting to login.

**Supported methods:** POST: /auth/{mount_point}/role/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **policies** (*str | list*) – Policies to be set on tokens issued using this role.
- **num_uses** (*int*) – Number of uses to set on a token produced by this role.

- **ttl** (*str | unicode*) – The TTL period of tokens issued using this role in seconds.
- **max_ttl** (*str | unicode*) – The maximum allowed lifetime of tokens issued in seconds using this role.
- **period** (*str | unicode*) – If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this parameter.
- **bound_service_principal_ids** (*list*) – The list of Service Principal IDs that login is restricted to.
- **bound_group_ids** (*list*) – The list of group ids that login is restricted to.
- **bound_locations** (*list*) – The list of locations that login is restricted to.
- **bound_subscription_ids** (*list*) – The list of subscription IDs that login is restricted to.
- **bound_resource_groups** (*list*) – The list of resource groups that login is restricted to.
- **bound_scale_sets** (*list*) – The list of scale set names that the login is restricted to.
- **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_config**(*mount_point='azure'*)

Delete the previously configured Azure config and credentials.

**Supported methods:** DELETE: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters** **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*name*, *mount_point='azure'*)

Delete the previously registered role.

**Supported methods:** DELETE: /auth/{mount_point}/role/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_roles**(*mount_point='azure'*)

List all the roles that are registered with the plugin.

**Supported methods:** LIST: /auth/{mount_point}/role. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The "data" key from the JSON response of the request.

**Return type** dict

**login**(*role*, *jwt*, *subscription_id=None*, *resource_group_name=None*, *vm_name=None*, *vmss_name=None*, *use_token=True*, *mount_point='azure'*)

Fetch a token.

This endpoint takes a signed JSON Web Token (JWT) and a role name for some entity. It verifies the JWT signature to authenticate that entity and then authorizes the entity for the given role.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **role** (*str | unicode*) – Name of the role against which the login is being attempted.
- **jwt** (*str | unicode*) – Signed JSON Web Token (JWT) from Azure MSI.
- **subscription_id** (*str | unicode*) – The subscription ID for the machine that generated the MSI token. This information can be obtained through instance metadata.
- **resource_group_name** (*str | unicode*) – The resource group for the machine that generated the MSI token. This information can be obtained through instance metadata.
- **vm_name** (*str | unicode*) – The virtual machine name for the machine that generated the MSI token. This information can be obtained through instance metadata. If vmss_name is provided, this value is ignored.
- **vmss_name** (*str | unicode*) – The virtual machine scale set name for the machine that generated the MSI token. This information can be obtained through instance metadata.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_config**(*mount_point='azure'*)

Return the previously configured config, including credentials.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**read_role**(*name*, *mount_point='azure'*)

Read the previously registered role configuration.

**Supported methods:** GET: /auth/{mount_point}/role/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The "data" key from the JSON response of the request.

**Return type** dict

**class** hvac.api.auth_methods.**Cert**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Cert Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/cert/index.html **Miscellaneous**

| |
|---|
| *CertificateAuthError* |

**Methods**

| | |
|---|---|
| *configure_tls_certificate*([mount_point, ...]) | Configure options for the method. |
| *create_ca_certificate_role*(name, certificate) | Create CA Certificate Role Sets a CA cert and associated parameters in a role name. |
| *delete_certificate_role*(name[, mount_point]) | List existing LDAP existing groups that have been created in this auth method. |
| *list_certificate_roles*([mount_point]) | Lists configured certificate names. |
| *login*([name, cacert, cert_pem, key_pem, ...]) | Log in and fetch a token. |
| *read_ca_certificate_role*(name[, mount_point]) | Gets information associated with the named role. |

**exception CertificateAuthError**

Bases: Exception

**configure_tls_certificate**(*mount_point='cert'*, *disable_binding=False*)

Configure options for the method.

**Supported methods:** POST: /auth/<mount point>/config. Produces: 204 (empty body)

**Parameters**

- **disable_binding** (*bool*) – If set, during renewal, skips the matching of presented client identity with the client identity used during login.
- **mount_point** –

**create_ca_certificate_role**(*name*, *certificate*, *allowed_common_names=''*, *allowed_dns_sans=''*, *allowed_email_sans=''*, *allowed_uri_sans=''*, *allowed_organizational_units=''*, *required_extensions=''*, *display_name=''*, *token_ttl=0*, *token_max_ttl=0*, *token_policies=[]*, *token_bound_cidrs=[]*, *token_explicit_max_ttl=0*, *token_no_default_policy=False*, *token_num_uses=0*, *token_period=0*, *token_type=''*, *mount_point='cert'*)

Create CA Certificate Role Sets a CA cert and associated parameters in a role name.

**Supported methods:** POST: /auth/<mount point>/certs/:name. Produces: 204 (empty body)

»Parameters :param name: The name of the certificate role. :type name: str :param certificate: The PEM-format CA certificate. :type certificate: str :param allowed_common_names: Constrain the Common Names in the client certificate with a globbed pattern. Value is a comma-separated list of patterns. Authentication requires at least one Name matching at least one pattern. If not set, defaults to allowing all names. :type allowed_common_names: str | list :param allowed_dns_sans: Constrain the Alternative Names in the client certificate with a globbed pattern. Value is a comma-separated list of patterns. Authentication requires at least one DNS matching at least one pattern. If not set, defaults to allowing all dns. :type allowed_dns_sans: str | list :param allowed_email_sans: Constrain the Alternative Names in the client certificate with a globbed pattern. Value is a comma-separated list of patterns. Authentication requires at least one Email matching at least one pattern. If not set, defaults to allowing all emails. :type allowed_email_sans: str | list :param allowed_uri_sans: Constrain the Alternative Names in the client certificate with a globbed pattern. Value is a comma-separated list of URI patterns. Authentication requires at least one URI matching at least one pattern. If not set, defaults to allowing all URIs. :type allowed_uri_sans: str | list :param allowed_organizational_units: Constrain the Organizational Units (OU) in the client certificate with a globbed pattern. Value is a comma-separated list of OU patterns. Authentication requires at least one OU matching at least one pattern. If not set, defaults to allowing all OUs. :type allowed_organizational_units: str | list :param required_extensions: Require specific Custom Extension OIDs to exist and match the pattern. Value is a comma separated string or array of oid:value. Expects the extension value to be some type of ASN1 encoded string. All conditions must be met. Supports globbing on value. :type required_extensions: str | list :param display_name: The display_name to set on tokens issued when authenticating against this CA certificate. If not set, defaults to the name of the role. :type display_name: str | unicode :param token_ttl: The incremental lifetime for generated tokens. This current value of this will be referenced at renewal time. :type token_ttl: int | str :param token_max_ttl: The maximum lifetime for generated tokens. This current value of this will be referenced at renewal time. :type token_max_ttl: int | str :param token_policies: List of policies to encode onto generated tokens. Depending on the auth method, this list may be supplemented by user/group/other values. :type token_policies: list | str :param token_bound_cidrs: List of CIDR blocks; if set, specifies blocks of IP addresses which can authenticate successfully, and ties the resulting token to these blocks as well. :type token_bound_cidrs: list | str :param token_explicit_max_ttl: If set, will encode an explicit max TTL onto the token. This is a hard cap even if token_ttl and token_max_ttl would otherwise allow a renewal. :type token_explicit_max_ttl: int | str :param token_no_default_policy: If set, the default policy will not be set on generated tokens; otherwise it will be added to the policies set in token_policies. :type token_no_default_policy: bool :param token_num_uses: The maximum number of times a generated token may be used (within its lifetime); 0 means unlimited. If you require the token to have the ability to create child tokens, you will need to set this value to 0. :type token_num_uses: int :param token_period: The period, if any, to set on the token. :type token_period: int | str :param token_type: The type of token that should be generated. Can be service, batch, or default to use the mount's tuned default (which unless changed will be service tokens). For token store roles, there are two additional possibilities: default-service and default-batch which specify the type to return unless the client requests a different type at generation time. :type token_type: str :param mount_point: :type mount_point:

**delete_certificate_role**(*name*, *mount_point='cert'*)

List existing LDAP existing groups that have been created in this auth method.

**Supported methods:** DELETE: /auth/{mount_point}/groups. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name of the certificate role.
- **mount_point** –

**list_certificate_roles**(*mount_point='cert'*)

Lists configured certificate names.

**Supported methods:** LIST: /auth/<mount point>/certs. Produces: 200 application/json

**Parameters** **mount_point** –

**Returns** The response of the list_certificate request.

**Return type** requests.Response

**login**(*name=''*, *cacert=False*, *cert_pem=''*, *key_pem=''*, *mount_point='cert'*, *use_token=True*)

**Log in and fetch a token. If there is a valid chain to a CA configured in the method and all role constraints** are matched, a token will be issued. If the certificate has DNS SANs in it, each of those will be verified. If Common Name is required to be verified, then it should be a fully qualified DNS domain name and must be duplicated as a DNS SAN

**Supported methods:** POST: /auth/<mount point>/login Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Authenticate against only the named certificate role, returning its policy list if successful. If not set, defaults to trying all certificate roles and returning any one that matches.
- **cacert** (*str | bool*) – The value used here is for the Vault TLS Listener CA certificate, not the CA that issued the client authentication certificate. This can be omitted if the CA used to issue the Vault server certificate is trusted by the local system executing this command.
- **cert_pem** – Location of the cert.pem used to authenticate the host.
- **key_pem** – Location of the public key.pem used to authenticate the host.
- **key_pem** – str | unicode
- **mount_point** –
- **use_token** – If the returned token is stored in the client
- **use_token** – bool

**Tupe cert_pem** str | unicode

**Returns** The response of the login request.

**Return type** requests.Response

**read_ca_certificate_role**(*name*, *mount_point='cert'*)

Gets information associated with the named role.

**Supported methods:** GET: /auth/<mount point>/certs/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – The name of the certificate role
- **mount_point** –

**Returns** The JSON response of the read_ca_certificate_role request.

**Return type** dict

**class** hvac.api.auth_methods.**Gcp**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Google Cloud Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/{mount_point}/index.html **Methods**

| | |
|---|---|
| *configure*([credentials, ...]) | Configure the credentials required for the GCP auth method to perform API calls to Google Cloud. |
| *create_role*(name, role_type, project_id[, ...]) | Register a role in the GCP auth method. |
| *delete_config*([mount_point]) | Delete all GCP configuration data. |
| *delete_role*(role[, mount_point]) | Delete the previously registered role. |
| *edit_labels_on_gce_role*(name[, add, remove, ...]) | Edit labels for an existing GCE role in the backend. |
| *edit_service_accounts_on_iam_role*(name[, ...]) | Edit service accounts for an existing IAM role in the GCP auth method. |
| *list_roles*([mount_point]) | List all the roles that are registered with the plugin. |
| *login*(role, jwt[, use_token, mount_point]) | Login to retrieve a Vault token via the GCP auth method. |
| *read_config*([mount_point]) | Read the configuration, if any, including credentials. |
| *read_role*(name[, mount_point]) | Read the previously registered role configuration. |

**configure**(*credentials=None*, *google_certs_endpoint='https://www.googleapis.com/oauth2/v3/certs'*, *mount_point='gcp'*)

Configure the credentials required for the GCP auth method to perform API calls to Google Cloud.

These credentials will be used to query the status of IAM entities and get service account or other Google public certificates to confirm signed JWTs passed in during login.

**Supported methods:** POST: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **credentials** (*str | unicode*) – A JSON string containing the contents of a GCP credentials file. The credentials file must have the following permissions: *iam.serviceAccounts.get*, *iam.serviceAccountKeys.get*. If this value is empty, Vault will try to use Application Default Credentials from the machine on which the Vault server is running. The project must have the iam.googleapis.com API enabled.
- **google_certs_endpoint** (*str | unicode*) – The Google OAuth2 endpoint from which to obtain public certificates. This is used for testing and should generally not be set by end users.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_role**(*name*, *role_type*, *project_id*, *ttl=None*, *max_ttl=None*, *period=None*, *policies=None*, *bound_service_accounts=None*, *max_jwt_exp=None*, *allow_gce_inference=None*, *bound_zones=None*, *bound_regions=None*, *bound_instance_groups=None*, *bound_labels=None*, *mount_point='gcp'*)

Register a role in the GCP auth method.

**Role types have specific entities that can perform login operations against this endpoint. Constraints specific** to the role type must be set on the role. These are applied to the authenticated entities attempting to login.

**Supported methods:** POST: /auth/{mount_point}/role/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name of the role.
- **role_type** (*str | unicode*) – The type of this role. Certain fields correspond to specific roles and will be rejected otherwise.
- **project_id** (*str | unicode*) – The GCP project ID. Only entities belonging to this project can authenticate with this role.
- **ttl** (*str | unicode*) – The TTL period of tokens issued using this role. This can be specified as an integer number of seconds or as a duration value like "5m".
- **max_ttl** (*str | unicode*) – The maximum allowed lifetime of tokens issued in seconds using this role. This can be specified as an integer number of seconds or as a duration value like "5m".
- **period** (*str | unicode*) – If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this parameter. This can be specified as an integer number of seconds or as a duration value like "5m".
- **policies** (*list*) – The list of policies to be set on tokens issued using this role.
- **bound_service_accounts** (*list*) – <required for iam> A list of service account emails or IDs that login is restricted to. If set to *, all service accounts are allowed (role will still be bound by project). Will be inferred from service account used to issue metadata token for GCE instances.
- **max_jwt_exp** (*str | unicode*) – <iam only> The number of seconds past the time of authentication that the login param JWT must expire within. For example, if a user attempts to login with a token that expires within an hour and this is set to 15 minutes, Vault will return an error prompting the user to create a new signed JWT with a shorter exp. The GCE metadata tokens currently do not allow the exp claim to be customized.
- **allow_gce_inference** (*bool*) – <iam only> A flag to determine if this role should allow GCE instances to authenticate by inferring service accounts from the GCE identity metadata token.
- **bound_zones** (*list*) – <gce only> The list of zones that a GCE instance must belong to in order to be authenticated. If bound_instance_groups is provided, it is assumed to be a zonal group and the group must belong to this zone.
- **bound_regions** (*list*) – <gce only> The list of regions that a GCE instance must belong to in order to be authenticated. If bound_instance_groups is provided, it is assumed to be a regional group and the group must belong to this region. If bound_zones are provided, this attribute is ignored.
- **bound_instance_groups** (*list*) – <gce only> The instance groups that an authorized instance must belong to in order to be authenticated. If specified, either bound_zones or bound_regions must be set too.
- **bound_labels** (*list*) – <gce only> A list of GCP labels formatted as "key:value" strings that must be set on authorized GCE instances. Because GCP labels are not currently ACL'd, we recommend that this be used in conjunction with other restrictions.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** requests.Response

**delete_config**(*mount_point='gcp'*)
Delete all GCP configuration data. This operation is idempotent.

**Supported methods:** DELETE: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*role*, *mount_point='gcp'*)
Delete the previously registered role.

**Supported methods:** DELETE: /auth/{mount_point}/role/{role}. Produces: 204 (empty body)

**Parameters**

- **role** (*str | unicode*) – The name of the role to delete.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**edit_labels_on_gce_role**(*name*, *add=None*, *remove=None*, *mount_point='gcp'*)
Edit labels for an existing GCE role in the backend.

This allows you to add or remove labels (keys, values, or both) from the list of keys on the role.

**Supported methods:** POST: /auth/{mount_point}/role/{name}/labels. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name of an existing gce role. This will return an error if role is not a gce type role.
- **add** (*list*) – The list of key:value labels to add to the GCE role's bound labels.
- **remove** (*list*) – The list of label keys to remove from the role's bound labels. If any of the specified keys do not exist, no error is returned (idempotent).
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the edit_labels_on_gce_role request.

**Return type** requests.Response

**edit_service_accounts_on_iam_role**(*name*, *add=None*, *remove=None*, *mount_point='gcp'*)
Edit service accounts for an existing IAM role in the GCP auth method.

This allows you to add or remove service accounts from the list of service accounts on the role.

**Supported methods:** POST: /auth/{mount_point}/role/{name}/service-accounts. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name of an existing iam type role. This will return an error if role is not an iam type role.
- **add** (*list*) – The list of service accounts to add to the role's service accounts.
- **remove** (*list*) – The list of service accounts to remove from the role's service accounts.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_roles**(*mount_point='gcp'*)

List all the roles that are registered with the plugin.

**Supported methods:** LIST: /auth/{mount_point}/roles. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**login**(*role*, *jwt*, *use_token=True*, *mount_point='gcp'*)

Login to retrieve a Vault token via the GCP auth method.

**This endpoint takes a signed JSON Web Token (JWT) and a role name for some entity. It verifies the JWT** signature with Google Cloud to authenticate that entity and then authorizes the entity for the given role.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **role** (*str | unicode*) – The name of the role against which the login is being attempted.
- **jwt** (*str | unicode*) – A signed JSON web token
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_config**(*mount_point='gcp'*)

Read the configuration, if any, including credentials.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**read_role**(*name*, *mount_point='gcp'*)
Read the previously registered role configuration.

**Supported methods:** GET: /auth/{mount_point}/role/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – The name of the role to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The data key from the JSON response of the read_role request.

**Return type** JSON

**class** hvac.api.auth_methods.**Github**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

GitHub Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/github/index.html **Methods**

| | |
|---|---|
| *configure*(organization[, base_url, ttl, ...]) | Configure the connection parameters for GitHub. |
| *login*(token[, use_token, mount_point]) | Login using GitHub access token. |
| *map_team*(team_name[, policies, mount_point]) | Map a list of policies to a team that exists in the configured GitHub organization. |
| *map_user*(user_name[, policies, mount_point]) | Map a list of policies to a specific GitHub user exists in the configured organization. |
| *read_configuration*([mount_point]) | Read the GitHub configuration. |
| *read_team_mapping*(team_name[, mount_point]) | Read the GitHub team policy mapping. |
| *read_user_mapping*(user_name[, mount_point]) | Read the GitHub user policy mapping. |

**configure**(*organization*, *base_url=None*, *ttl=None*, *max_ttl=None*, *mount_point='github'*)
Configure the connection parameters for GitHub.

This path honors the distinction between the create and update capabilities inside ACL policies.

**Supported methods:** POST: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **organization** (*str | unicode*) – The organization users must be part of.
- **base_url** (*str | unicode*) – The API endpoint to use. Useful if you are running GitHub Enterprise or an API-compatible authentication server.
- **ttl** (*str | unicode*) – Duration after which authentication will be expired.
- **max_ttl** (*str | unicode*) – Maximum duration after which authentication will be expired.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the configure_method request.

**Return type** requests.Response

**login**(*token*, *use_token=True*, *mount_point='github'*)
Login using GitHub access token.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **token** (*str | unicode*) – GitHub personal API token.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the login request.

**Return type** dict

**map_team**(*team_name*, *policies=None*, *mount_point='github'*)

Map a list of policies to a team that exists in the configured GitHub organization.

**Supported methods:** POST: /auth/{mount_point}/map/teams/{team_name}. Produces: 204 (empty body)

**Parameters**

- **team_name** (*str | unicode*) – GitHub team name in "slugified" format
- **policies** (*List[str]*) – Comma separated list of policies to assign
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the map_github_teams request.

**Return type** requests.Response

**map_user**(*user_name*, *policies=None*, *mount_point='github'*)

Map a list of policies to a specific GitHub user exists in the configured organization.

**Supported methods:** POST: /auth/{mount_point}/map/users/{user_name}. Produces: 204 (empty body)

**Parameters**

- **user_name** (*str | unicode*) – GitHub user name
- **policies** (*List[str]*) – Comma separated list of policies to assign
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the map_github_users request.

**Return type** requests.Response

**read_configuration**(*mount_point='github'*)

Read the GitHub configuration.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_configuration request.

**Return type** dict

**read_team_mapping**(*team_name*, *mount_point='github'*)
Read the GitHub team policy mapping.

**Supported methods:** GET: /auth/{mount_point}/map/teams/{team_name}. Produces: 200 application/json

**Parameters**

- **team_name** (*str | unicode*) – GitHub team name
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_team_mapping request.

**Return type** dict

**read_user_mapping**(*user_name*, *mount_point='github'*)
Read the GitHub user policy mapping.

**Supported methods:** GET: /auth/{mount_point}/map/users/{user_name}. Produces: 200 application/json

**Parameters**

- **user_name** (*str | unicode*) – GitHub user name
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_user_mapping request.

**Return type** dict

**class** hvac.api.auth_methods.**JWT**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

JWT auth method which can be used to authenticate with Vault by providing a JWT.

The OIDC method allows authentication via a configured OIDC provider using the user's web browser. This method may be initiated from the Vault UI or the command line. Alternatively, a JWT can be provided directly. The JWT is cryptographically verified using locally-provided keys, or, if configured, an OIDC Discovery service can be used to fetch the appropriate keys. The choice of method is configured per role.

Reference: https://www.vaultproject.io/api/auth/jwt **Attributes**

| | |
|---|---|
| *DEFAULT_PATH* | str(object='') -> str |

**Methods**

| | |
|---|---|
| *configure*([oidc_discovery_url, ...]) | Configure the validation information to be used globally across all roles. |
| *create_role*(name, user_claim, ...[, ...]) | Register a role in the JWT method. |
| *delete_role*(name[, path]) | Delete the previously registered role. |
| *jwt_login*(role, jwt[, path]) | Fetch a token. |
| *list_roles*([path]) | List all the roles that are registered with the plugin. |
| *oidc_authorization_url_request*(role, ...[, path]) | Obtain an authorization URL from Vault to start an OIDC login flow. |
| *oidc_callback*(state, nonce, code[, path]) | Exchange an authorization code for an OIDC ID Token. |

continues on next page

Table 22 – continued from previous page

| | |
|---|---|
| read_config([path]) | Read the previously configured config. |
| read_role(name[, path]) | Read the previously registered role configuration. |
| resolve_path(path) | Return the class's default path if no explicit path is specified. |

**DEFAULT_PATH = 'jwt'**

**configure**(*oidc_discovery_url=None*, *oidc_discovery_ca_pem=None*, *oidc_client_id=None*, *oidc_client_secret=None*, *oidc_response_mode=None*, *oidc_response_types=None*, *jwks_url=None*, *jwks_ca_pem=None*, *jwt_validation_pubkeys=None*, *bound_issuer=None*, *jwt_supported_algs=None*, *default_role=None*, *provider_config=None*, *path=None*)

Configure the validation information to be used globally across all roles.

One (and only one) of oidc_discovery_url and jwt_validation_pubkeys must be set.

**Supported methods:** POST: /auth/{path}/config.

**Parameters**

- **oidc_discovery_url** (*str | unicode*) – The OIDC Discovery URL, without any .well-known component (base path). Cannot be used with "jwks_url" or "jwt_validation_pubkeys".
- **oidc_discovery_ca_pem** (*str | unicode*) – The CA certificate or chain of certificates, in PEM format, to use to validate connections to the OIDC Discovery URL. If not set, system certificates are used.
- **oidc_client_id** (*str | unicode*) – The OAuth Client ID from the provider for OIDC roles.
- **oidc_client_secret** (*str | unicode*) – The OAuth Client Secret from the provider for OIDC roles.
- **oidc_response_mode** (*str | unicode*) – The response mode to be used in the OAuth2 request. Allowed values are "query" and form_post". Defaults to "query".
- **oidc_response_types** (*str | unicode*) – The response types to request. Allowed values are "code" and "id_token". Defaults to "code". Note: "id_token" may only be used if "oidc_response_mode" is set to "form_post".
- **jwks_url** (*str | unicode*) – JWKS URL to use to authenticate signatures. Cannot be used with "oidc_discovery_url" or "jwt_validation_pubkeys".
- **jwks_ca_pem** (*str | unicode*) – The CA certificate or chain of certificates, in PEM format, to use to validate connections to the JWKS URL. If not set, system certificates are used.
- **jwt_validation_pubkeys** (*str | unicode*) – A list of PEM-encoded public keys to use to authenticate signatures locally. Cannot be used with "jwks_url" or "oidc_discovery_url".
- **bound_issuer** (*str | unicode*) – in a JWT.
- **jwt_supported_algs** (*str | unicode*) – A list of supported signing algorithms. Defaults to [RS256].
- **default_role** (*str | unicode*) – The default role to use if none is provided during login.
- **provider_config** (*map*) – TypeError

- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the configure request.

**Return type** requests.Response

**create_role**(*name*, *user_claim*, *allowed_redirect_uris*, *role_type='jwt'*, *bound_audiences=None*, *clock_skew_leeway=None*, *expiration_leeway=None*, *not_before_leeway=None*, *bound_subject=None*, *bound_claims=None*, *groups_claim=None*, *claim_mappings=None*, *oidc_scopes=None*, *bound_claims_type='string'*, *verbose_oidc_logging=False*, *token_ttl=None*, *token_max_ttl=None*, *token_policies=None*, *token_bound_cidrs=None*, *token_explicit_max_ttl=None*, *token_no_default_policy=None*, *token_num_uses=None*, *token_period=None*, *token_type=None*, *path=None*)

Register a role in the JWT method.

Role types have specific entities that can perform login operations against this endpoint. Constraints specific to the role type must be set on the role. These are applied to the authenticated entities attempting to login. At least one of the bound values must be set.

**Supported methods:** POST: /auth/{path}/role/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **role_type** (*str | unicode*) – Type of role, either "oidc" or "jwt" (default).
- **bound_audiences** (*list*) – List of aud claims to match against. Any match is sufficient. Required for "jwt" roles, optional for "oidc" roles.
- **user_claim** (*str | unicode*) – The claim to use to uniquely identify the user; this will be used as the name for the Identity entity alias created due to a successful login. The claim value must be a string.
- **clock_skew_leeway** (*int*) – Only applicable with "jwt" roles.
- **expiration_leeway** (*int*) – Only applicable with "jwt" roles.
- **not_before_leeway** (*int*) – Only applicable with "jwt" roles.
- **bound_subject** (*str | unicode*) – If set, requires that the sub claim matches this value.
- **bound_claims** (*dict*) – If set, a dict of claims (keys) to match against respective claim values (values). The expected value may be a single string or a list of strings. The interpretation of the bound claim values is configured with bound_claims_type. Keys support JSON pointer syntax for referencing claims.
- **groups_claim** (*str | unicode*) – The claim to use to uniquely identify the set of groups to which the user belongs; this will be used as the names for the Identity group aliases created due to a successful login. The claim value must be a list of strings. Supports JSON pointer syntax for referencing claims.
- **claim_mappings** (*map*) – If set, a map of claims (keys) to be copied to specified metadata fields (values). Keys support JSON pointer syntax for referencing claims.
- **oidc_scopes** (*list*) – If set, a list of OIDC scopes to be used with an OIDC role. The standard scope "openid" is automatically included and need not be specified.
- **allowed_redirect_uris** (*list*) – The list of allowed values for redirect_uri during OIDC logins.

- **bound_claims_type** (*str | unicode*) – Configures the interpretation of the bound_claims values. If "string" (the default), the values will treated as string literals and must match exactly. If set to "glob", the values will be interpreted as globs, with * matching any number of characters.
- **verbose_oidc_logging** (*bool*) – Log received OIDC tokens and claims when debug-level logging is active. Not recommended in production since sensitive information may be present in OIDC responses.
- **token_ttl** (*int | str*) – The incremental lifetime for generated tokens. This current value of this will be referenced at renewal time.
- **token_max_ttl** (*int | str*) – The maximum lifetime for generated tokens. This current value of this will be referenced at renewal time.
- **token_policies** (*list[str]*) – List of policies to encode onto generated tokens. Depending on the auth method, this list may be supplemented by user/group/other values.
- **token_bound_cidrs** (*list[str]*) – List of CIDR blocks; if set, specifies blocks of IP addresses which can authenticate successfully, and ties the resulting token to these blocks as well.
- **token_explicit_max_ttl** (*int | str*) – If set, will encode an explicit max TTL onto the token. This is a hard cap even if token_ttl and token_max_ttl would otherwise allow a renewal.
- **token_no_default_policy** (*bool*) – If set, the default policy will not be set on generated tokens; otherwise it will be added to the policies set in token_policies.
- **token_num_uses** (*str | unicode*) – The maximum number of times a generated token may be used (within its lifetime); 0 means unlimited. If you require the token to have the ability to create child tokens, you will need to set this value to 0.
- **token_period** (*int | str*) – The period, if any, to set on the token.
- **token_type** (*str*) – The type of token that should be generated. Can be service, batch, or default.
- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_role request.

**Return type** dict

**delete_role**(*name*, *path=None*)

Delete the previously registered role.

**Supported methods:** DELETE: /auth/{path}/role/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the delete_role request.

**Return type** requests.Response

**jwt_login**(*role*, *jwt*, *path=None*)

Fetch a token.

This endpoint takes a signed JSON Web Token (JWT) and a role name for some entity. It verifies the JWT signature to authenticate that entity and then authorizes the entity for the given role.

**Supported methods:** POST: /auth/{path}/login.

**Parameters**

- **role** (*str | unicode*) – not provided.
- **jwt** (*str | unicode*) – Signed JSON Web Token (JWT).
- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the jwt_login request.

**Return type** requests.Response

**list_roles**(*path=None*)

List all the roles that are registered with the plugin.

**Supported methods:** LIST: /auth/{path}/role.

**Parameters** **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the list_roles request.

**Return type** dict

**oidc_authorization_url_request**(*role*, *redirect_uri*, *path=None*)

Obtain an authorization URL from Vault to start an OIDC login flow.

**Supported methods:** POST: /auth/{path}/auth_url.

**Parameters**

- **role** (*str | unicode*) – not provided.
- **redirect_uri** (*str | unicode*) – more information.
- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the _authorization_url_request request.

**Return type** requests.Response

**oidc_callback**(*state*, *nonce*, *code*, *path=None*)

Exchange an authorization code for an OIDC ID Token.

The ID token will be further validated against any bound claims, and if valid a Vault token will be returned.

**Supported methods:** GET: /auth/{path}/callback.

**Parameters**

- **state** (*str | unicode*) – Opaque state ID that is part of the Authorization URL and will be included in the the redirect following successful authentication on the provider.
- **nonce** (*str | unicode*) – Opaque nonce that is part of the Authorization URL and will be included in the the redirect following successful authentication on the provider.
- **code** (*str | unicode*) – Provider-generated authorization code that Vault will exchange for an ID token.
- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the _callback request.

**Return type** requests.Response

**read_config**(*path=None*)
Read the previously configured config.

**Supported methods:** GET: /auth/{path}/config.

**Returns** The response of the read_config request.

**Return type** dict

**read_role**(*name*, *path=None*)
Read the previously registered role configuration.

**Supported methods:** GET: /auth/{path}/role/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the read_role request.

**Return type** dict

**resolve_path**(*path*)
Return the class's default path if no explicit path is specified.

**Parameters** **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The default path for this auth method if no explicit path is specified.

**Return type** str

**class** hvac.api.auth_methods.**Kubernetes**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

Kubernetes Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/kubernetes/index.html **Methods**

| | |
|---|---|
| *configure*(kubernetes_host[, ...]) | Configure the connection parameters for Kubernetes. |
| *create_role*(name, ...[, ttl, max_ttl, ...]) | Create a role in the method. |
| *delete_role*(name[, mount_point]) | Delete the previously registered role. |
| *list_roles*([mount_point]) | List all the roles that are registered with the plugin. |
| *login*(role, jwt[, use_token, mount_point]) | Fetch a token. |
| *read_config*([mount_point]) | Return the previously configured config, including credentials. |
| *read_role*(name[, mount_point]) | Returns the previously registered role configuration. |

**configure**(*kubernetes_host*, *kubernetes_ca_cert=None*, *token_reviewer_jwt=None*, *pem_keys=None*, *issuer=None*, *mount_point='kubernetes'*)
Configure the connection parameters for Kubernetes.

This path honors the distinction between the create and update capabilities inside ACL policies.

**Supported methods:** POST: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **kubernetes_host** (*str | unicode*) – Host must be a host string, a host:port pair, or a URL to the base of the Kubernetes API server. Example: https://k8s.example.com:443
- **kubernetes_ca_cert** (*str | unicode*) – PEM encoded CA cert for use by the TLS client used to talk with the Kubernetes API. NOTE: Every line must end with a newline:
- **token_reviewer_jwt** (*str | unicode*) – A service account JWT used to access the TokenReview API to validate other JWTs during login. If not set the JWT used for login will be used to access the API.
- **pem_keys** (*list*) – Optional list of PEM-formatted public keys or certificates used to verify the signatures of Kubernetes service account JWTs. If a certificate is given, its public key will be extracted. Not every installation of Kubernetes exposes these keys.
- **issuer** – Optional JWT issuer.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the configure_method request.

**Return type** requests.Response

**create_role**(*name*, *bound_service_account_names*, *bound_service_account_namespaces*, *ttl=None*, *max_ttl=None*, *period=None*, *policies=None*, *mount_point='kubernetes'*)

Create a role in the method.

Registers a role in the auth method. Role types have specific entities that can perform login operations against this endpoint. Constraints specific to the role type must be set on the role. These are applied to the authenticated entities attempting to login.

**Supported methods:** POST: /auth/{mount_point}/role/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **bound_service_account_names** (*list | str | unicode*) – List of service account names able to access this role. If set to "*" all names are allowed.
- **bound_service_account_namespaces** (*list | str | unicode*) – List of namespaces allowed to access this role. If set to "*" all namespaces are allowed.
- **ttl** (*str | unicode*) – The TTL period of tokens issued using this role in seconds.
- **max_ttl** (*str | unicode*) – The maximum allowed lifetime of tokens issued in seconds using this role.
- **period** (*str | unicode*) – If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this parameter.
- **policies** (*list | str | unicode*) – Policies to be set on tokens issued using this role.
- **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*name*, *mount_point='kubernetes'*)
Delete the previously registered role.

**Supported methods:** DELETE: /auth/{mount_point}/role/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the kubernetes auth method was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_roles**(*mount_point='kubernetes'*)
List all the roles that are registered with the plugin.

**Supported methods:** LIST: /auth/{mount_point}/role. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the kubernetes auth method was mounted on.

**Returns** The "data" key from the JSON response of the request.

**Return type** dict

**login**(*role*, *jwt*, *use_token=True*, *mount_point='kubernetes'*)
Fetch a token.

This endpoint takes a signed JSON Web Token (JWT) and a role name for some entity. It verifies the JWT signature to authenticate that entity and then authorizes the entity for the given role.

**Supported methods:** POST: /auth/{mount_point}/login. Produces: 200 application/json

**Parameters**

- **role** (*str | unicode*) – Name of the role against which the login is being attempted.
- **jwt** (*str | unicode*) – Signed JSON Web Token (JWT) from Azure MSI.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the azure auth method was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_config**(*mount_point='kubernetes'*)
Return the previously configured config, including credentials.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the kubernetes auth method was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**read_role**(*name*, *mount_point='kubernetes'*)
Returns the previously registered role configuration.

**Supported methods:** POST: /auth/{mount_point}/role/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the kubernetes auth method was mounted on.

**Returns** The "data" key from the JSON response of the request.

**Return type** dict

**class** hvac.api.auth_methods.**Ldap**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

LDAP Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/ldap/index.html **Methods**

| | |
|---|---|
| *configure*([user_dn, group_dn, url, ...]) | Configure the LDAP auth method. |
| *create_or_update_group*(name[, policies, ...]) | Create or update LDAP group policies. |
| *create_or_update_user*(username[, policies, ...]) | Create or update LDAP users policies and group associations. |
| *delete_group*(name[, mount_point]) | Delete a LDAP group and policy association. |
| *delete_user*(username[, mount_point]) | Delete a LDAP user and policy association. |
| *list_groups*([mount_point]) | List existing LDAP existing groups that have been created in this auth method. |
| *list_users*([mount_point]) | List existing users in the method. |
| *login*(username, password[, use_token, ...]) | Log in with LDAP credentials. |
| *read_configuration*([mount_point]) | Retrieve the LDAP configuration for the auth method. |
| *read_group*(name[, mount_point]) | Read policies associated with a LDAP group. |
| *read_user*(username[, mount_point]) | Read policies associated with a LDAP user. |

**configure**(*user_dn=None*, *group_dn=None*, *url=None*, *case_sensitive_names=None*, *starttls=None*, *tls_min_version=None*, *tls_max_version=None*, *insecure_tls=None*, *certificate=None*, *bind_dn=None*, *bind_pass=None*, *user_attr=None*, *discover_dn=None*, *deny_null_bind=True*, *upn_domain=None*, *group_filter=None*, *group_attr=None*, *use_token_groups=None*, *mount_point='ldap'*)
Configure the LDAP auth method.

**Supported methods:** POST: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **user_dn** (*str | unicode*) – Base DN under which to perform user search. Example: ou=Users,dc=example,dc=com
- **group_dn** (*str | unicode*) – LDAP search base to use for group membership search. This can be the root containing either groups or users. Example: ou=Groups,dc=example,dc=com

- **url** (*str | unicode*) – The LDAP server to connect to. Examples: ldap://ldap.myorg.com, ldaps://ldap.myorg.com:636. Multiple URLs can be specified with commas, e.g. ldap://ldap.myorg.com,ldap://ldap2.myorg.com; these will be tried in-order.
- **case_sensitive_names** (*bool*) – If set, user and group names assigned to policies within the backend will be case sensitive. Otherwise, names will be normalized to lower case. Case will still be preserved when sending the username to the LDAP server at login time; this is only for matching local user/group definitions.
- **starttls** (*bool*) – If true, issues a StartTLS command after establishing an unencrypted connection.
- **tls_min_version** (*str | unicode*) – Minimum TLS version to use. Accepted values are tls10, tls11 or tls12.
- **tls_max_version** (*str | unicode*) – Maximum TLS version to use. Accepted values are tls10, tls11 or tls12.
- **insecure_tls** (*bool*) – If true, skips LDAP server SSL certificate verification - insecure, use with caution!
- **certificate** (*str | unicode*) – CA certificate to use when verifying LDAP server certificate, must be x509 PEM encoded.
- **bind_dn** (*str | unicode*) – Distinguished name of object to bind when performing user search. Example: cn=vault,ou=Users,dc=example,dc=com
- **bind_pass** (*str | unicode*) – Password to use along with binddn when performing user search.
- **user_attr** (*str | unicode*) – Attribute on user attribute object matching the username passed when authenticating. Examples: sAMAccountName, cn, uid
- **discover_dn** (*bool*) – Use anonymous bind to discover the bind DN of a user.
- **deny_null_bind** (*bool*) – This option prevents users from bypassing authentication when providing an empty password.
- **upn_domain** (*str | unicode*) – The userPrincipalDomain used to construct the UPN string for the authenticating user. The constructed UPN will appear as [username]@UPNDomain. Example: example.com, which will cause vault to bind as username@example.com.
- **group_filter** (*str | unicode*) – Go template used when constructing the group membership query. The template can access the following context variables: [UserDN, Username]. The default is *(|(memberUid={{.Username}})(member={{.UserDN}})(uniqueMember={{.UserDN}}))*, which is compatible with several common directory schemas. To support nested group resolution for Active Directory, instead use the following query: (&(objectClass=group)(member:1.2.840.113556.1.4.1941:={{.UserDN}})).
- **group_attr** (*str | unicode*) – LDAP attribute to follow on objects returned by groupfilter in order to enumerate user group membership. Examples: for groupfilter queries returning group objects, use: cn. For queries returning user objects, use: memberOf. The default is cn.
- **use_token_groups** (*bool*) – If true, groups are resolved through Active Directory tokens. This may speed up nested group membership resolution in large directories.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the configure request.

**Return type** requests.Response

**create_or_update_group**(*name*, *policies=None*, *mount_point='ldap'*)
Create or update LDAP group policies.

**Supported methods:** POST: /auth/{mount_point}/groups/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name of the LDAP group
- **policies** (*list*) – List of policies associated with the group. This parameter is transformed to a comma-delimited string before being passed to Vault.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_or_update_group request.

**Return type** requests.Response

**create_or_update_user**(*username*, *policies=None*, *groups=None*, *mount_point='ldap'*)
Create or update LDAP users policies and group associations.

**Supported methods:** POST: /auth/{mount_point}/users/{username}. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – The username of the LDAP user
- **policies** (*str | unicode*) – List of policies associated with the user. This parameter is transformed to a comma-delimited string before being passed to Vault.
- **groups** (*str | unicode*) – List of groups associated with the user. This parameter is transformed to a comma-delimited string before being passed to Vault.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_or_update_user request.

**Return type** requests.Response

**delete_group**(*name*, *mount_point='ldap'*)
Delete a LDAP group and policy association.

**Supported methods:** DELETE: /auth/{mount_point}/groups/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name of the LDAP group
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the delete_group request.

**Return type** requests.Response

**delete_user**(*username*, *mount_point='ldap'*)
Delete a LDAP user and policy association.

**Supported methods:** DELETE: /auth/{mount_point}/users/{username}. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – The username of the LDAP user

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the delete_user request.

**Return type** requests.Response

**list_groups**(*mount_point='ldap'*)
List existing LDAP existing groups that have been created in this auth method.

**Supported methods:** LIST: /auth/{mount_point}/groups. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the list_groups request.

**Return type** dict

**list_users**(*mount_point='ldap'*)
List existing users in the method.

**Supported methods:** LIST: /auth/{mount_point}/users. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the list_users request.

**Return type** dict

**login**(*username*, *password*, *use_token=True*, *mount_point='ldap'*)
Log in with LDAP credentials.

**Supported methods:** POST: /auth/{mount_point}/login/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – The username of the LDAP user
- **password** (*str | unicode*) – The password for the LDAP user
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the login_with_user request.

**Return type** requests.Response

**read_configuration**(*mount_point='ldap'*)
Retrieve the LDAP configuration for the auth method.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_configuration request.

**Return type** dict

**read_group**(*name*, *mount_point='ldap'*)
Read policies associated with a LDAP group.

**Supported methods:** GET: /auth/{mount_point}/groups/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – The name of the LDAP group
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_group request.

**Return type** dict

**read_user**(*username*, *mount_point='ldap'*)
Read policies associated with a LDAP user.

**Supported methods:** GET: /auth/{mount_point}/users/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – The username of the LDAP user
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_user request.

**Return type** dict

**class** hvac.api.auth_methods.**Mfa**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

Multi-factor authentication Auth Method (API).

**Warning:** This class's methods correspond to a legacy / unsupported set of Vault API routes. Please see the reference link for additional context.

**Methods**

| | |
|---|---|
| *configure*(mount_point[, mfa_type, force]) | Configure MFA for a supported method. |
| *configure_duo_access*(mount_point, host, ...) | Configure the access keys and host for Duo API connections. |
| *configure_duo_behavior*(mount_point[, ...]) | Configure Duo second factor behavior. |
| *read_configuration*(mount_point) | Read the MFA configuration. |
| *read_duo_behavior_configuration*(mount_point) | Read the Duo second factor behavior configuration. |

Reference: https://www.vaultproject.io/docs/auth/mfa.html

**configure**(*mount_point*, *mfa_type='duo'*, *force=False*)
Configure MFA for a supported method.

This endpoint allows you to turn on multi-factor authentication with a given backend. Currently only Duo is supported.

**Supported methods:** POST: /auth/{mount_point}/mfa_config. Produces: 204 (empty body)

**Parameters**

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.
- **mfa_type** (*str | unicode*) – Enables MFA with given backend (available: duo)
- **force** (*bool*) – If True, make the "mfa_config" request regardless of circumstance. If False (the default), verify the provided mount_point is available and one of the types of methods supported by this feature.

**Returns** The response of the configure MFA request.

**Return type** requests.Response

**configure_duo_access**(*mount_point*, *host*, *integration_key*, *secret_key*)

Configure the access keys and host for Duo API connections.

To authenticate users with Duo, the backend needs to know what host to connect to and must authenticate with an integration key and secret key. This endpoint is used to configure that information.

**Supported methods:** POST: /auth/{mount_point}/duo/access. Produces: 204 (empty body)

**Parameters**

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.
- **host** (*str | unicode*) – Duo API host
- **integration_key** (*Duo secret key*) – Duo integration key
- **secret_key** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the configure_duo_access request.

**Return type** requests.Response

**configure_duo_behavior**(*mount_point*, *push_info=None*, *user_agent=None*, *username_format='%s'*)

Configure Duo second factor behavior.

This endpoint allows you to configure how the original auth method username maps to the Duo username by providing a template format string.

**Supported methods:** POST: /auth/{mount_point}/duo/config. Produces: 204 (empty body)

**Parameters**

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.
- **push_info** (*str | unicode*) – A string of URL-encoded key/value pairs that provides additional context about the authentication attempt in the Duo Mobile app
- **user_agent** (*str | unicode*) – User agent to connect to Duo (default "")
- **username_format** (*str | unicode*) – Format string given auth method username as argument to create Duo username (default '%s')

**Returns** The response of the configure_duo_behavior request.

**Return type** requests.Response

**read_configuration**(*mount_point*)

Read the MFA configuration.

**Supported methods:** GET: /auth/{mount_point}/mfa_config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_configuration request.

**Return type** dict

**read_duo_behavior_configuration**(*mount_point*)

Read the Duo second factor behavior configuration.

**Supported methods:** GET: /auth/{mount_point}/duo/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_duo_behavior_configuration request.

**Return type** dict

**class** hvac.api.auth_methods.**OIDC**(*adapter*)

Bases: hvac.api.auth_methods.jwt.JWT

OIDC auth method which can be used to authenticate with Vault using OIDC.

The OIDC method allows authentication via a configured OIDC provider using the user's web browser. This method may be initiated from the Vault UI or the command line. Alternatively, a JWT can be provided directly. The JWT is cryptographically verified using locally-provided keys, or, if configured, an OIDC Discovery service can be used to fetch the appropriate keys. The choice of method is configured per role.

Note: this class is duplicative of the JWT class (as both JWT and OIDC share the same family of Vault API routes).

Reference: https://www.vaultproject.io/api/auth/jwt **Attributes**

| | |
|---|---|
| *DEFAULT_PATH* | str(object='') -> str |

**Methods**

| | |
|---|---|
| *create_role*(name, user_claim, ... [, ... ]) | Register a role in the OIDC method. |

**DEFAULT_PATH = 'oidc'**

**create_role**(*name*, *user_claim*, *allowed_redirect_uris*, *role_type='oidc'*, *bound_audiences=None*, *clock_skew_leeway=None*, *expiration_leeway=None*, *not_before_leeway=None*, *bound_subject=None*, *bound_claims=None*, *groups_claim=None*, *claim_mappings=None*, *oidc_scopes=None*, *bound_claims_type='string'*, *verbose_oidc_logging=False*, *token_ttl=None*, *token_max_ttl=None*, *token_policies=None*, *token_bound_cidrs=None*, *token_explicit_max_ttl=None*, *token_no_default_policy=None*, *token_num_uses=None*, *token_period=None*, *token_type=None*, *path=None*)

Register a role in the OIDC method.

Role types have specific entities that can perform login operations against this endpoint. Constraints specific to the role type must be set on the role. These are applied to the authenticated entities attempting to login. At least one of the bound values must be set.

**Supported methods:** POST: /auth/{path}/role/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **role_type** (*str | unicode*) – Type of role, either "oidc" or "jwt" (default).
- **bound_audiences** (*list*) – List of aud claims to match against. Any match is sufficient. Required for "jwt" roles, optional for "oidc" roles.
- **user_claim** (*str | unicode*) – The claim to use to uniquely identify the user; this will be used as the name for the Identity entity alias created due to a successful login. The claim value must be a string.
- **clock_skew_leeway** (*int*) – Only applicable with "jwt" roles.
- **expiration_leeway** (*int*) – Only applicable with "jwt" roles.
- **not_before_leeway** (*int*) – Only applicable with "jwt" roles.
- **bound_subject** (*str | unicode*) – If set, requires that the sub claim matches this value.
- **bound_claims** (*dict*) – If set, a dict of claims (keys) to match against respective claim values (values). The expected value may be a single string or a list of strings. The interpretation of the bound claim values is configured with bound_claims_type. Keys support JSON pointer syntax for referencing claims.
- **groups_claim** (*str | unicode*) – The claim to use to uniquely identify the set of groups to which the user belongs; this will be used as the names for the Identity group aliases created due to a successful login. The claim value must be a list of strings. Supports JSON pointer syntax for referencing claims.
- **claim_mappings** (*map*) – If set, a map of claims (keys) to be copied to specified metadata fields (values). Keys support JSON pointer syntax for referencing claims.
- **oidc_scopes** (*list*) – If set, a list of OIDC scopes to be used with an OIDC role. The standard scope "openid" is automatically included and need not be specified.
- **allowed_redirect_uris** (*list*) – The list of allowed values for redirect_uri during OIDC logins.
- **bound_claims_type** (*str | unicode*) – Configures the interpretation of the bound_claims values. If "string" (the default), the values will treated as string literals and must match exactly. If set to "glob", the values will be interpreted as globs, with * matching any number of characters.
- **verbose_oidc_logging** (*bool*) – Log received OIDC tokens and claims when debug-level logging is active. Not recommended in production since sensitive information may be present in OIDC responses.
- **token_ttl** (*int | str*) – The incremental lifetime for generated tokens. This current value of this will be referenced at renewal time.
- **token_max_ttl** (*int | str*) – The maximum lifetime for generated tokens. This current value of this will be referenced at renewal time.
- **token_policies** (*list[str]*) – List of policies to encode onto generated tokens. Depending on the auth method, this list may be supplemented by user/group/other values.
- **token_bound_cidrs** (*list[str]*) – List of CIDR blocks; if set, specifies blocks of IP addresses which can authenticate successfully, and ties the resulting token to these blocks as well.

- **token_explicit_max_ttl** (*int | str*) – If set, will encode an explicit max TTL onto the token. This is a hard cap even if token_ttl and token_max_ttl would otherwise allow a renewal.
- **token_no_default_policy** (*bool*) – If set, the default policy will not be set on generated tokens; otherwise it will be added to the policies set in token_policies.
- **token_num_uses** (*str | unicode*) – The maximum number of times a generated token may be used (within its lifetime); 0 means unlimited. If you require the token to have the ability to create child tokens, you will need to set this value to 0.
- **token_period** (*int | str*) – The period, if any, to set on the token.
- **token_type** (*str*) – The type of token that should be generated. Can be service, batch, or default.
- **path** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_role request.

**Return type** dict

**class** hvac.api.auth_methods.**Okta**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Okta Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/okta/index.html **Methods**

| | |
|---|---|
| *configure*(org_name[, api_token, base_url, ...]) | Configure the connection parameters for Okta. |
| *delete_group*(name[, mount_point]) | Delete an existing group from the method. |
| *delete_user*(username[, mount_point]) | Delete an existing username from the method. |
| *list_groups*([mount_point]) | List the groups configured in the Okta method. |
| *list_users*([mount_point]) | List the users configured in the Okta method. |
| *login*(username, password[, use_token, ...]) | Login with the username and password. |
| *read_config*([mount_point]) | Read the Okta configuration. |
| *read_group*(name[, mount_point]) | Read the properties of an existing group. |
| *read_user*(username[, mount_point]) | Read the properties of an existing username. |
| *register_group*(name[, policies, mount_point]) | Register a new group and maps a set of policies to it. |
| *register_user*(username[, groups, policies, ...]) | Register a new user and maps a set of policies to it. |

**configure**(*org_name*, *api_token=None*, *base_url=None*, *ttl=None*, *max_ttl=None*, *bypass_okta_mfa=None*, *mount_point='okta'*)

Configure the connection parameters for Okta.

This path honors the distinction between the create and update capabilities inside ACL policies.

**Supported methods:** POST: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **org_name** (*str | unicode*) – Name of the organization to be used in the Okta API.
- **api_token** (*str | unicode*) – Okta API token. This is required to query Okta for user group membership. If this is not supplied only locally configured groups will be enabled.

- **base_url** (*str | unicode*) – If set, will be used as the base domain for API requests. Examples are okta.com, oktapreview.com, and okta-emea.com.
- **ttl** (*str | unicode*) – Duration after which authentication will be expired.
- **max_ttl** (*str | unicode*) – Maximum duration after which authentication will be expired.
- **bypass_okta_mfa** (*bool*) – Whether to bypass an Okta MFA request. Useful if using one of Vault's built-in MFA mechanisms, but this will also cause certain other statuses to be ignored, such as PASSWORD_EXPIRED.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_group**(*name*, *mount_point='okta'*)

Delete an existing group from the method.

**Supported methods:** DELETE: /auth/{mount_point}/groups/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name for the group.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_user**(*username*, *mount_point='okta'*)

Delete an existing username from the method.

**Supported methods:** DELETE: /auth/{mount_point}/users/{username}. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – Username for this user.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_groups**(*mount_point='okta'*)

List the groups configured in the Okta method.

**Supported methods:** LIST: /auth/{mount_point}/groups. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_users**(*mount_point='okta'*)

List the users configured in the Okta method.

**Supported methods:** LIST: /auth/{mount_point}/users. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**login**(*username*, *password*, *use_token=True*, *mount_point='okta'*)
Login with the username and password.

**Supported methods:** POST: /auth/{mount_point}/login/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – Username for this user.
- **password** (*str | unicode*) – Password for the authenticating user.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the login request.

**Return type** dict

**read_config**(*mount_point='okta'*)
Read the Okta configuration.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_group**(*name*, *mount_point='okta'*)
Read the properties of an existing group.

**Supported methods:** GET: /auth/{mount_point}/groups/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – The name for the group.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_user**(*username*, *mount_point='okta'*)
Read the properties of an existing username.

**Supported methods:** GET: /auth/{mount_point}/users/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – Username for this user.

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**register_group**(*name*, *policies=None*, *mount_point='okta'*)

Register a new group and maps a set of policies to it.

**Supported methods:** POST: /auth/{mount_point}/groups/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – The name of the group.
- **policies** (*list*) – The list or comma-separated string of policies associated with the group.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**register_user**(*username*, *groups=None*, *policies=None*, *mount_point='okta'*)

Register a new user and maps a set of policies to it.

**Supported methods:** POST: /auth/{mount_point}/users/{username}. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – Name of the user.
- **groups** (*list*) – List or comma-separated string of groups associated with the user.
- **policies** (*list*) – List or comma-separated string of policies associated with the user.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**class** hvac.api.auth_methods.**Radius**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

RADIUS Auth Method (API).

Reference: https://www.vaultproject.io/docs/auth/radius.html **Methods**

| | |
|---|---|
| *configure*(host, secret[, port, ...]) | Configure the RADIUS auth method. |
| *delete_user*(username[, mount_point]) | Delete a RADIUS user and policy association. |
| *list_users*([mount_point]) | List existing users in the method. |
| *login*(username, password[, use_token, ...]) | Log in with RADIUS credentials. |
| *read_configuration*([mount_point]) | Retrieve the RADIUS configuration for the auth method. |
| *read_user*(username[, mount_point]) | Read policies associated with a RADIUS user. |
| *register_user*(username[, policies, mount_point]) | Create or update RADIUS user with a set of policies. |

**configure**(*host*, *secret*, *port=None*, *unregistered_user_policies=None*, *dial_timeout=None*, *nas_port=None*, *mount_point='radius'*)

Configure the RADIUS auth method.

**Supported methods:** POST: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **host** (*str | unicode*) – The RADIUS server to connect to. Examples: radius.myorg.com, 127.0.0.1
- **secret** (*str | unicode*) – The RADIUS shared secret.
- **port** (*int*) – The UDP port where the RADIUS server is listening on. Defaults is 1812.
- **unregistered_user_policies** (*list*) – A comma-separated list of policies to be granted to unregistered users.
- **dial_timeout** (*int*) – Number of second to wait for a backend connection before timing out. Default is 10.
- **nas_port** (*int*) – The NAS-Port attribute of the RADIUS request. Defaults is 10.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the configure request.

**Return type** requests.Response

**delete_user**(*username*, *mount_point='radius'*)

Delete a RADIUS user and policy association.

**Supported methods:** DELETE: /auth/{mount_point}/users/{username}. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – The username of the RADIUS user
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the delete_user request.

**Return type** requests.Response

**list_users**(*mount_point='radius'*)

List existing users in the method.

**Supported methods:** LIST: /auth/{mount_point}/users. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the list_users request.

**Return type** dict

**login**(*username*, *password*, *use_token=True*, *mount_point='radius'*)

Log in with RADIUS credentials.

**Supported methods:** POST: /auth/{mount_point}/login/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – The username of the RADIUS user
- **password** (*str | unicode*) – The password for the RADIUS user

- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the login_with_user request.

**Return type** requests.Response

**read_configuration**(*mount_point='radius'*)

Retrieve the RADIUS configuration for the auth method.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_configuration request.

**Return type** dict

**read_user**(*username*, *mount_point='radius'*)

Read policies associated with a RADIUS user.

**Supported methods:** GET: /auth/{mount_point}/users/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – The username of the RADIUS user
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_user request.

**Return type** dict

**register_user**(*username*, *policies=None*, *mount_point='radius'*)

Create or update RADIUS user with a set of policies.

**Supported methods:** POST: /auth/{mount_point}/users/{username}. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – Username for this RADIUS user.
- **policies** (*list*) – List of policies associated with the user. This parameter is transformed to a comma-delimited string before being passed to Vault.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the register_user request.

**Return type** requests.Response

**class** hvac.api.auth_methods.**Userpass**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

USERPASS Auth Method (API). Reference: https://www.vaultproject.io/api/auth/userpass/index.html **Methods**

| | |
|---|---|
| *create_or_update_user*(username, password[, ...]) | Create/update user in userpass. |
| *delete_user*(username[, mount_point]) | Delete user in the auth method. |
| *list_user*([mount_point]) | List existing users that have been created in the auth method |
| *login*(username, password[, mount_point]) | Log in with USERPASS credentials. |
| *read_user*(username[, mount_point]) | Read user in the auth method. |
| *update_password_on_user*(username, password) | update password for the user in userpass. |

**create_or_update_user**(*username*, *password*, *policies=None*, *mount_point='userpass'*)
Create/update user in userpass.

**Supported methods:** POST: /auth/{mount_point}/users/{username}. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – The username for the user.
- **password** (*str | unicode*) – The password for the user. Only required when creating the user.
- **policies** (*str | unicode*) – The list of policies to be set on username created.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**delete_user**(*username*, *mount_point='userpass'*)
Delete user in the auth method.

**Supported methods:** GET: /auth/{mount_point}/users/{username}. Produces: 200 application/json

**Parameters**

- **username** – The username for the user.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_group request.

**Return type** dict

**list_user**(*mount_point='userpass'*)
List existing users that have been created in the auth method

**Supported methods:** LIST: /auth/{mount_point}/users. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the list_groups request.

**Return type** dict

**login**(*username*, *password*, *mount_point='userpass'*)
Log in with USERPASS credentials.

**Supported methods:** POST: /auth/{mount_point}/login/{username}. Produces: 200 application/json

**Parameters**

- **username** (*str | unicode*) – The username for the user.
- **password** (*str | unicode*) – The password for the user. Only required when creating the user.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**read_user**(*username*, *mount_point='userpass'*)
Read user in the auth method.

**Supported methods:** GET: /auth/{mount_point}/users/{username}. Produces: 200 application/json

**Parameters**

- **username** – The username for the user.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_group request.

**Return type** dict

**update_password_on_user**(*username*, *password*, *mount_point='userpass'*)
update password for the user in userpass.

**Supported methods:** POST: /auth/{mount_point}/users/{username}/password. Produces: 204 (empty body)

**Parameters**

- **username** (*str | unicode*) – The username for the user.
- **password** (*str | unicode*) – The password for the user. Only required when creating the user.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

## 4.4 hvac.api.secrets_engines

Vault secrets engines endpoints

**Classes**

| | |
|---|---|
| *Aws*(adapter) | AWS Secrets Engine (API). |
| *Azure*(adapter) | Azure Secrets Engine (API). |
| *Gcp*(adapter) | Google Cloud Secrets Engine (API). |
| *ActiveDirectory*(adapter) | Active Directory Secrets Engine (API). |
| *Identity*(adapter) | Identity Secrets Engine (API). |
| *Kv*(adapter[, default_kv_version]) | Class containing methods for the key/value secrets_engines backend API routes. |
| *KvV1*(adapter) | KV Secrets Engine - Version 1 (API). |
| *KvV2*(adapter) | KV Secrets Engine - Version 2 (API). |
| *Pki*(adapter) | Pki Secrets Engine (API). |
| *Transform*(adapter) | Transform Secrets Engine (API). |
| *Transit*(adapter) | Transit Secrets Engine (API). |
| *SecretsEngines*(adapter) | Secrets Engines. |
| *Database*(adapter) | Database Secrets Engine (API). |

continues on next page

Table 31 – continued from previous page

| | |
|---|---|
| *RabbitMQ*(adapter) | RabbitMQ Secrets Engine (API). |

**class** hvac.api.secrets_engines.**ActiveDirectory**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Active Directory Secrets Engine (API). Reference: https://www.vaultproject.io/api/secret/ad/index.html **Methods**

| | |
|---|---|
| *configure*([binddn, bindpass, url, userdn, ...]) | Configure shared information for the ad secrets engine. |
| *create_or_update_role*(name[, ...]) | This endpoint creates or updates the ad role definition. |
| *delete_role*(name[, mount_point]) | This endpoint deletes a ad role with the given name. |
| *generate_credentials*(name[, mount_point]) | This endpoint retrieves the previous and current LDAP password for |
| *list_roles*([mount_point]) | This endpoint lists all existing roles in the secrets engine. |
| *read_config*([mount_point]) | Read the configured shared information for the ad secrets engine. |
| *read_role*(name[, mount_point]) | This endpoint queries for information about a ad role with the given name. |

**configure**(*binddn=None*, *bindpass=None*, *url=None*, *userdn=None*, *upndomain=None*, *ttl=None*, *max_ttl=None*, *mount_point='ad'*, **args*, ***kwargs*)

Configure shared information for the ad secrets engine.

**Supported methods:** POST: /{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **binddn** (*str | unicode*) – Distinguished name of object to bind when performing user and group search.
- **bindpass** (*str | unicode*) – Password to use along with binddn when performing user search.
- **url** (*str | unicode*) – Base DN under which to perform user search.
- **userdn** (*str | unicode*) – Base DN under which to perform user search.
- **upndomain** (*str | unicode*) – userPrincipalDomain used to construct the UPN string for the authenticating user.
- **ttl** (*int | str*) – – The default password time-to-live in seconds. Once the ttl has passed, a password will be rotated the next time it's requested.
- **max_ttl** (*int | str*) – The maximum password time-to-live in seconds. No role will be allowed to set a custom ttl greater than the max_ttl integer number of seconds or Go duration format string.**
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_or_update_role**(*name*, *service_account_name=None*, *ttl=None*, *mount_point='ad'*)

This endpoint creates or updates the ad role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of an existing role against which to create this ad credential.
- **service_account_name** (*str | unicode*) – The name of a pre-existing service account in Active Directory that maps to this role. This value is required on create and optional on update.
- **ttl** (*str | unicode*) – Specifies the TTL for this role. This is provided as a string duration with a time suffix like "30s" or "1h" or as seconds. If not provided, the default Vault TTL is used.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: ad).

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*name*, *mount_point='ad'*)

This endpoint deletes a ad role with the given name. Even if the role does not exist, this endpoint will still return a successful response. :param name: Specifies the name of the role to delete. :type name: str | unicode :param mount_point: Specifies the place where the secrets engine will be accessible (default: ad). :type mount_point: str | unicode :return: The response of the request. :rtype: requests.Response

**generate_credentials**(*name*, *mount_point='ad'*)

**This endpoint retrieves the previous and current LDAP password for** the associated account (or rotate if required)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to request credentials from.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: ad).

**Returns** The response of the request.

**Return type** requests.Response

**list_roles**(*mount_point='ad'*)

This endpoint lists all existing roles in the secrets engine. :return: The response of the request. :rtype: requests.Response

**read_config**(*mount_point='ad'*)

Read the configured shared information for the ad secrets engine.

Credentials will be omitted from returned data.

**Supported methods:** GET: /{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_role**(*name*, *mount_point='ad'*)

This endpoint queries for information about a ad role with the given name. If no role exists with that name, a 404 is returned. :param name: Specifies the name of the role to query. :type name: str | unicode

:param mount_point: Specifies the place where the secrets engine will be accessible (default: ad). :type mount_point: str | unicode :return: The response of the request. :rtype: requests.Response

**class** hvac.api.secrets_engines.**Aws**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

AWS Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/aws/index.html **Methods**

| | |
|---|---|
| *configure_lease*(lease, lease_max[, mount_point]) | Configure lease settings for the AWS secrets engine. |
| *configure_root_iam_credentials*(access_key, ...) | Configure the root IAM credentials to communicate with AWS. |
| *create_or_update_role*(name, credential_type) | Create or update the role with the given name. |
| *delete_role*(name[, mount_point]) | Delete an existing role by the given name. |
| *generate_credentials*(name[, role_arn, ttl, ...]) | Generates credential based on the named role. |
| *list_roles*([mount_point]) | List all existing roles in the secrets engine. |
| *read_lease_config*([mount_point]) | Read the current lease settings for the AWS secrets engine. |
| *read_role*(name[, mount_point]) | Query an existing role by the given name. |
| *rotate_root_iam_credentials*([mount_point]) | Rotate static root IAM credentials. |

**configure_lease**(*lease*, *lease_max*, *mount_point='aws'*)

Configure lease settings for the AWS secrets engine.

It is optional, as there are default values for lease and lease_max.

**Supported methods:** POST: /{mount_point}/config/lease. Produces: 204 (empty body)

**Parameters**

- **lease** (*str | unicode*) – Specifies the lease value provided as a string duration with time suffix. "h" (hour) is the largest suffix.
- **lease_max** (*str | unicode*) – Specifies the maximum lease value provided as a string duration with time suffix. "h" (hour) is the largest suffix.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**configure_root_iam_credentials**(*access_key*, *secret_key*, *region=None*, *iam_endpoint=None*, *sts_endpoint=None*, *max_retries=None*, *mount_point='aws'*)

Configure the root IAM credentials to communicate with AWS.

There are multiple ways to pass root IAM credentials to the Vault server, specified below with the highest precedence first. If credentials already exist, this will overwrite them.

The official AWS SDK is used for sourcing credentials from env vars, shared files, or IAM/ECS instances.

- Static credentials provided to the API as a payload
- Credentials in the AWS_ACCESS_KEY, AWS_SECRET_KEY, and AWS_REGION environment variables on the server

- Shared credentials files
- Assigned IAM role or ECS task role credentials

At present, this endpoint does not confirm that the provided AWS credentials are valid AWS credentials with proper permissions.

**Supported methods:** POST: /{mount_point}/config/root. Produces: 204 (empty body)

**Parameters**

- **access_key** (*str | unicode*) – Specifies the AWS access key ID.
- **secret_key** (*str | unicode*) – Specifies the AWS secret access key.
- **region** (*str | unicode*) – Specifies the AWS region. If not set it will use the AWS_REGION env var, AWS_DEFAULT_REGION env var, or us-east-1 in that order.
- **iam_endpoint** (*str | unicode*) – Specifies a custom HTTP IAM endpoint to use.
- **sts_endpoint** (*str | unicode*) – Specifies a custom HTTP STS endpoint to use.
- **max_retries** (*int*) – Number of max retries the client should use for recoverable errors. The default (-1) falls back to the AWS SDK's default behavior.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_or_update_role**(*name*, *credential_type*, *policy_document=None*, *default_sts_ttl=None*, *max_sts_ttl=None*, *role_arns=None*, *policy_arns=None*, *legacy_params=False*, *iam_tags=None*, *mount_point='aws'*)

Create or update the role with the given name.

If a role with the name does not exist, it will be created. If the role exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/roles/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to create. This is part of the request URL.
- **credential_type** (*str | unicode*) – Specifies the type of credential to be used when retrieving credentials from the role. Must be one of iam_user, assumed_role, or federation_token.
- **policy_document** (*dict | str | unicode*) – The IAM policy document for the role. The behavior depends on the credential type. With iam_user, the policy document will be attached to the IAM user generated and augment the permissions the IAM user has. With assumed_role and federation_token, the policy document will act as a filter on what the credentials can do.
- **default_sts_ttl** (*str | unicode*) – The default TTL for STS credentials. When a TTL is not specified when STS credentials are requested, and a default TTL is specified on the role, then this default TTL will be used. Valid only when credential_type is one of assumed_role or federation_token.
- **max_sts_ttl** (*str | unicode*) – The max allowed TTL for STS credentials (credentials TTL are capped to max_sts_ttl). Valid only when credential_type is one of assumed_role or federation_token.

- **role_arns** (*list | str | unicode*) – Specifies the ARNs of the AWS roles this Vault role is allowed to assume. Required when credential_type is assumed_role and prohibited otherwise. This is a comma-separated string or JSON array. String types supported for Vault legacy parameters.
- **policy_arns** (*list*) – Specifies the ARNs of the AWS managed policies to be attached to IAM users when they are requested. Valid only when credential_type is iam_user. When credential_type is iam_user, at least one of policy_arns or policy_document must be specified. This is a comma-separated string or JSON array.
- **legacy_params** (*bool*) – Flag to send legacy (Vault versions < 0.11.0) parameters in the request. When this is set to True, policy_document and policy_arns are the only parameters used from this method.
- **iam_tags** (*list*) – A list of strings representing a key/value pair to be used for any IAM user that is created by this role. Format is a key and value separated by an =.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*name*, *mount_point='aws'*)

Delete an existing role by the given name.

If the role does not exist, a 404 is returned.

**Supported methods:** DELETE: /{mount_point}/roles/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – the name of the role to delete. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**generate_credentials**(*name*, *role_arn=None*, *ttl=None*, *endpoint='creds'*, *mount_point='aws'*)

Generates credential based on the named role.

This role must be created before queried.

The /aws/creds and /aws/sts endpoints are almost identical. The exception is when retrieving credentials for a role that was specified with the legacy arn or policy parameter. In this case, credentials retrieved through /aws/sts must be of either the assumed_role or federation_token types, and credentials retrieved through /aws/creds must be of the iam_user type.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to generate credentials against. This is part of the request URL.
- **role_arn** (*str | unicode*) – The ARN of the role to assume if credential_type on the Vault role is assumed_role. Must match one of the allowed role ARNs in the Vault role. Optional if the Vault role only allows a single AWS role ARN; required otherwise.
- **ttl** (*str | unicode*) – Specifies the TTL for the use of the STS token. This is specified as a string with a duration suffix. Valid only when credential_type is assumed_role or federation_token. When not specified, the default sts_ttl set for the role will be used.

If that is also not set, then the default value of 3600s will be used. AWS places limits on the maximum TTL allowed. See the AWS documentation on the DurationSeconds parameter for AssumeRole (for assumed_role credential types) and GetFederationToken (for federation_token credential types) for more details.

- **endpoint** (*str | unicode*) – Supported endpoints: GET: /{mount_point}/creds/{name}. Produces: 200 application/json GET: /{mount_point}/sts/{name}. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_roles**(*mount_point='aws'*)

List all existing roles in the secrets engine.

**Supported methods:** LIST: /{mount_point}/roles. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_lease_config**(*mount_point='aws'*)

Read the current lease settings for the AWS secrets engine.

**Supported methods:** GET: /{mount_point}/config/lease. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_role**(*name*, *mount_point='aws'*)

Query an existing role by the given name.

If the role does not exist, a 404 is returned.

**Supported methods:** GET: /{mount_point}/roles/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to read. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**rotate_root_iam_credentials**(*mount_point='aws'*)

Rotate static root IAM credentials.

When you have configured Vault with static credentials, you can use this endpoint to have Vault rotate the access key it used. Note that, due to AWS eventual consistency, after calling this endpoint, subsequent calls from Vault to AWS may fail for a few seconds until AWS becomes consistent again.

In order to call this endpoint, Vault's AWS access key MUST be the only access key on the IAM user; otherwise, generation of a new access key will fail. Once this method is called, Vault will now be the only entity that knows the AWS secret key is used to access AWS.

**Supported methods:** POST: /{mount_point}/config/rotate-root. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**class** hvac.api.secrets_engines.**Azure**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Azure Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/azure/index.html **Methods**

| | |
|---|---|
| *configure*(subscription_id, tenant_id[, ...]) | Configure the credentials required for the plugin to perform API calls to Azure. |
| *create_or_update_role*(name, azure_roles[, ...]) | Create or update a Vault role. |
| *delete_config*([mount_point]) | Delete the stored Azure configuration and credentials. |
| *generate_credentials*(name[, mount_point]) | Generate a new service principal based on the named role. |
| *list_roles*([mount_point]) | List all of the roles that are registered with the plugin. |
| *read_config*([mount_point]) | Read the stored configuration, omitting client_secret. |

**configure**(*subscription_id*, *tenant_id*, *client_id=None*, *client_secret=None*, *environment=None*, *mount_point='azure'*)

Configure the credentials required for the plugin to perform API calls to Azure.

These credentials will be used to query roles and create/delete service principals. Environment variables will override any parameters set in the config.

**Supported methods:** POST: /{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **subscription_id** (*str | unicode*) – The subscription id for the Azure Active Directory
- **tenant_id** (*str | unicode*) – The tenant id for the Azure Active Directory.
- **client_id** (*str | unicode*) – The OAuth2 client id to connect to Azure.
- **client_secret** (*str | unicode*) – The OAuth2 client secret to connect to Azure.
- **environment** (*str | unicode*) – The Azure environment. If not specified, Vault will use Azure Public Cloud.
- **mount_point** (*str | unicode*) – The OAuth2 client secret to connect to Azure.

**Returns** The response of the request.

**Return type** requests.Response

**create_or_update_role**(*name*, *azure_roles*, *ttl=None*, *max_ttl=None*, *mount_point='azure'*)

Create or update a Vault role.

The provided Azure roles must exist for this call to succeed. See the Azure secrets roles docs for more information about roles.

**Supported methods:** POST: /{mount_point}/roles/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **azure_roles** (*list(dict)*) – List of Azure roles to be assigned to the generated service principal.
- **ttl** (*str | unicode*) – Specifies the default TTL for service principals generated using this role. Accepts time suffixed strings ("1h") or an integer number of seconds. Defaults to the system/engine default TTL time.
- **max_ttl** (*str | unicode*) – Specifies the maximum TTL for service principals generated using this role. Accepts time suffixed strings ("1h") or an integer number of seconds. Defaults to the system/engine max TTL time.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_config**(*mount_point='azure'*)

Delete the stored Azure configuration and credentials.

**Supported methods:** DELETE: /auth/{mount_point}/config. Produces: 204 (empty body)

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**generate_credentials**(*name*, *mount_point='azure'*)

Generate a new service principal based on the named role.

**Supported methods:** GET: /{mount_point}/creds/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to create credentials against.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**list_roles**(*mount_point='azure'*)

List all of the roles that are registered with the plugin.

**Supported methods:** LIST: /{mount_point}/roles. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**read_config**(*mount_point='azure'*)
Read the stored configuration, omitting client_secret.

**Supported methods:** GET: /{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The data key from the JSON response of the request.

**Return type** dict

**class** hvac.api.secrets_engines.**Database**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

Database Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/databases/index.html **Methods**

| | |
|---|---|
| *configure*(name, plugin_name[, ...]) | This endpoint configures the connection string used to communicate with the desired database. |
| *create_role*(name, db_name, creation_statements) | This endpoint creates or updates a role definition. |
| *create_static_role*(name, db_name, username, ...) | This endpoint creates or updates a static role definition. |
| *delete_connection*(name[, mount_point]) | This endpoint deletes a connection. |
| *delete_role*(name[, mount_point]) | This endpoint deletes the role definition. |
| *delete_static_role*(name[, mount_point]) | This endpoint deletes the static role definition. |
| *generate_credentials*(name[, mount_point]) | This endpoint generates a new set of dynamic credentials based on the named role. |
| *get_static_credentials*(name[, mount_point]) | This endpoint returns the current credentials based on the named static role. |
| *list_connections*([mount_point]) | This endpoint returns a list of available connections. |
| *list_roles*([mount_point]) | This endpoint returns a list of available roles. |
| *list_static_roles*([mount_point]) | This endpoint returns a list of available static roles. |
| *read_connection*(name[, mount_point]) | This endpoint returns the configuration settings for a connection. |
| *read_role*(name[, mount_point]) | This endpoint queries the role definition. |
| *reset_connection*(name[, mount_point]) | This endpoint closes a connection and it's underlying plugin and restarts it with the configuration stored in the barrier. |
| *rotate_root_credentials*(name[, mount_point]) | This endpoint is used to rotate the root superuser credentials stored for the database connection. |

**configure**(*name*, *plugin_name*, *verify_connection=None*, *allowed_roles=None*, *root_rotation_statements=None*, *mount_point='database'*, **args*, ***kwargs*)
This endpoint configures the connection string used to communicate with the desired database. In addition to the parameters listed here, each Database plugin has additional, database plugin specific, parameters for this endpoint. Please read the HTTP API for the plugin you'd wish to configure to see the full list of additional parameters.

**Parameters**

- **name** (*str | unicode*) – Specifies the name for this database connection. This is specified as part of the URL.

- **plugin_name** (*str | unicode*) – Specifies the name of the plugin to use for this connection.
- **verify_connection** (*bool*) – Specifies if the connection is verified during initial configuration.
- **allowed_roles** (*list*) – List of the roles allowed to use this connection. Defaults to empty (no roles), if contains a "*" any role can use this connection.
- **root_rotation_statements** (*list*) – Specifies the database statements to be executed to rotate the root user's credentials.

**Returns** The response of the request.

**Return type** requests.Response

**create_role**(*name*, *db_name*, *creation_statements*, *default_ttl=None*, *max_ttl=None*, *revocation_statements=None*, *rollback_statements=None*, *renew_statements=None*, *mount_point='database'*)

This endpoint creates or updates a role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the database role to manage.
- **db_name** (*str | unicode*) – The name of the database connection to use for this role.
- **creation_statements** (*list*) – Specifies the database statements executed to create and configure a user.
- **default_ttl** (*int*) – Specifies the TTL for the leases associated with this role.
- **max_ttl** (*int*) – Specifies the maximum TTL for the leases associated with this role.
- **revocation_statements** (*list*) – Specifies the database statements to be executed to revoke a user.
- **rollback_statements** (*list*) – Specifies the database statements to be executed to rollback a create operation in the event of an error.
- **renew_statements** (*list*) – Specifies the database statements to be executed to renew a user.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_static_role**(*name*, *db_name*, *username*, *rotation_statements*, *rotation_period=86400*, *mount_point='database'*)

This endpoint creates or updates a static role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to create.
- **db_name** (*str | unicode*) – The name of the database connection to use for this role.
- **username** (*str | unicode*) – Specifies the database username that the Vault role *name* above corresponds to.

- **rotation_statements** (*list*) – Specifies the database statements to be executed to rotate the password for the configured database user. Not every plugin type will support this functionality. See the plugin's API page for more information on support and formatting for this parameter.
- **rotation_period** (*int*) – Specifies the amount of time Vault should wait before rotating the password. The minimum is 5 seconds.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_connection**(*name*, *mount_point='database'*)
This endpoint deletes a connection.

**Parameters** **name** (*str | unicode*) – Specifies the name of the connection to delete.

**Returns** The response of the request.

**Return type** requests.Response

**delete_role**(*name*, *mount_point='database'*)
This endpoint deletes the role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to delete.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_static_role**(*name*, *mount_point='database'*)
This endpoint deletes the static role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to delete.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**generate_credentials**(*name*, *mount_point='database'*)
This endpoint generates a new set of dynamic credentials based on the named role.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to create credentials against
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**get_static_credentials**(*name*, *mount_point='database'*)
This endpoint returns the current credentials based on the named static role.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to create credentials against

- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_connections**(*mount_point='database'*)
This endpoint returns a list of available connections.

**Returns** The response of the request.

**Return type** requests.Response

**list_roles**(*mount_point='database'*)
This endpoint returns a list of available roles.

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_static_roles**(*mount_point='database'*)
This endpoint returns a list of available static roles.

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_connection**(*name*, *mount_point='database'*)
This endpoint returns the configuration settings for a connection.

**Parameters** **name** (*str | unicode*) – Specifies the name of the connection to read.

**Returns** The response of the request.

**Return type** requests.Response

**read_role**(*name*, *mount_point='database'*)
This endpoint queries the role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**reset_connection**(*name*, *mount_point='database'*)
This endpoint closes a connection and it's underlying plugin and restarts it with the configuration stored in the barrier.

**Parameters** **name** (*str | unicode*) – Specifies the name of the connection to reset.

**Returns** The response of the request.

**Return type** requests.Response

**rotate_root_credentials**(*name*, *mount_point='database'*)
This endpoint is used to rotate the root superuser credentials stored for the database connection. This user must have permissions to update its own password.

**Parameters** **name** (*str | unicode*) – Specifies the name of the connection to rotate.

**Returns** The response of the request.

**Return type** requests.Response

**class** hvac.api.secrets_engines.**Gcp**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Google Cloud Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/gcp/index.html **Methods**

| | |
|---|---|
| *configure*([credentials, ttl, max_ttl, ...]) | Configure shared information for the Gcp secrets engine. |
| *create_or_update_roleset*(name, project, bindings) | Create a roleset or update an existing roleset. |
| *delete_roleset*(name[, mount_point]) | Delete an existing roleset by the given name. |
| *generate_oauth2_access_token*(roleset[, ...]) | Generate an OAuth2 token with the scopes defined on the roleset. |
| *generate_service_account_key*(roleset[, ...]) | Generate Secret (IAM Service Account Creds): Service Account Key |
| *list_rolesets*([mount_point]) | List configured rolesets. |
| *read_config*([mount_point]) | Read the configured shared information for the Gcp secrets engine. |
| *read_roleset*(name[, mount_point]) | Read a roleset. |
| *rotate_roleset_account*(name[, mount_point]) | Rotate the service account this roleset uses to generate secrets. |
| *rotate_roleset_account_key*(name[, mount_point]) | Rotate the service account key this roleset uses to generate access tokens. |

**configure**(*credentials=None*, *ttl=None*, *max_ttl=None*, *mount_point='gcp'*)

Configure shared information for the Gcp secrets engine.

**Supported methods:** POST: /{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **credentials** (*str | unicode*) – JSON credentials (either file contents or '@path/to/file') See docs for alternative ways to pass in to this parameter, as well as the required permissions.
- **ttl** (*int | str*) – – Specifies default config TTL for long-lived credentials (i.e. service account keys). Accepts integer number of seconds or Go duration format string.
- **max_ttl** (*int | str*) – Specifies the maximum config TTL for long-lived credentials (i.e. service account keys). Accepts integer number of seconds or Go duration format string.**
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_or_update_roleset**(*name*, *project*, *bindings*, *secret_type=None*, *token_scopes=None*, *mount_point='gcp'*)

Create a roleset or update an existing roleset.

**See roleset docs for the GCP secrets backend to learn more about what happens when you create or update a** roleset.

**Supported methods:** POST: /{mount_point}/roleset/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role. Cannot be updated.
- **project** (*str | unicode*) – Name of the GCP project that this roleset's service account will belong to. Cannot be updated.
- **bindings** (*str | unicode*) – Bindings configuration string (expects HCL or JSON format in raw or base64-encoded string)
- **secret_type** (*str | unicode*) – Cannot be updated.
- **token_scopes** (*list[str]*) – List of OAuth scopes to assign to access_token secrets generated under this role set (access_token role sets only)
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_roleset**(*name*, *mount_point='gcp'*)

Delete an existing roleset by the given name.

**Supported methods:** DELETE: /{mount_point}/roleset/{name} Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**generate_oauth2_access_token**(*roleset*, *mount_point='gcp'*)

Generate an OAuth2 token with the scopes defined on the roleset.

This OAuth access token can be used in GCP API calls, e.g. curl -H "Authorization: Bearer $TOKEN" ...

**Supported methods:** GET: /{mount_point}/token/{roleset}. Produces: 200 application/json

**Parameters**

- **roleset** (*str | unicode*) – Name of an roleset with secret type access_token to generate access_token under.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**generate_service_account_key**(*roleset*, *key_algorithm='KEY_ALG_RSA_2048'*, *key_type='TYPE_GOOGLE_CREDENTIALS_FILE'*, *method='POST'*, *mount_point='gcp'*)

Generate Secret (IAM Service Account Creds): Service Account Key

**If using GET ('read'), the optional parameters will be set to their defaults. Use POST if you want to specify** different values for these params.

**Parameters**

- **roleset** (*str | unicode*) – Name of an roleset with secret type service_account_key to generate key under.
- **key_algorithm** (*str | unicode*) – Key algorithm used to generate key. Defaults to 2k RSA key You probably should not choose other values (i.e. 1k),
- **key_type** (*str | unicode*) – Private key type to generate. Defaults to JSON credentials file.
- **method** (*str | unicode*) – Supported methods: POST: /{mount_point}/key/{roleset}. Produces: 200 application/json GET: /{mount_point}/key/{roleset}. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_rolesets**(*mount_point='gcp'*)

List configured rolesets.

**Supported methods:** LIST: /{mount_point}/rolesets. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_config**(*mount_point='gcp'*)

Read the configured shared information for the Gcp secrets engine.

Credentials will be omitted from returned data.

**Supported methods:** GET: /{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_roleset**(*name*, *mount_point='gcp'*)

Read a roleset.

**Supported methods:** GET: /{mount_point}/roleset/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**rotate_roleset_account**(*name*, *mount_point='gcp'*)
Rotate the service account this roleset uses to generate secrets.

**This also replaces the key access_token roleset. This can be used to invalidate old secrets generated by the** roleset or fix issues if a roleset's service account (and/or keys) was changed outside of Vault (i.e. through GCP APIs/cloud console).

**Supported methods:** POST: /{mount_point}/roleset/{name}/rotate. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**rotate_roleset_account_key**(*name*, *mount_point='gcp'*)
Rotate the service account key this roleset uses to generate access tokens.

This does not recreate the roleset service account.

**Supported methods:** POST: /{mount_point}/roleset/{name}/rotate-key. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**class** hvac.api.secrets_engines.**Identity**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

Identity Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/identity/entity.html **Methods**

| | |
|---|---|
| *configure_tokens_backend*([issuer, mount_point]) | Update configurations for OIDC-compliant identity tokens issued by Vault. |
| *create_named_key*(name[, rotation_period, ...]) | Create or update a named key which is used by a role to sign tokens. |
| *create_or_update_entity*(name[, entity_id, ...]) | Create or update an Entity. |
| *create_or_update_entity_alias*(name, ...[, ...]) | Create a new alias for an entity. |
| *create_or_update_entity_by_name*(name[, ...]) | Create or update an entity by a given name. |
| *create_or_update_group*(name[, group_id, ...]) | Create or update a Group. |
| *create_or_update_group_alias*(name[, ...]) | Creates or update a group alias. |

continues on next page

Table 37 – continued from previous page

| | |
|---|---|
| create_or_update_group_by_name(name[, ...]) | Create or update a group by its name. |
| create_or_update_role(name, key[, template, ...]) | Create or update a role. |
| delete_entity(entity_id[, mount_point]) | Delete an entity and all its associated aliases. |
| delete_entity_alias(alias_id[, mount_point]) | Delete a entity alias. |
| delete_entity_by_name(name[, mount_point]) | Delete an entity and all its associated aliases, given the entity name. |
| delete_group(group_id[, mount_point]) | Delete a group. |
| delete_group_alias(entity_id[, mount_point]) | Delete a group alias. |
| delete_group_by_name(name[, mount_point]) | Delete a group, given its name. |
| delete_named_key(name[, mount_point]) | Delete a named key. |
| delete_role(name[, mount_point]) | Deletes a role. |
| generate_signed_id_token(name[, mount_point]) | Generate a signed ID (OIDC) token. |
| introspect_signed_id_token(token[, ...]) | Verify the authenticity and active state of a signed ID token. |
| list_entities([method, mount_point]) | List available entities entities by their identifiers. |
| list_entities_by_name([method, mount_point]) | List available entities by their names. |
| list_entity_aliases([method, mount_point]) | List available entity aliases by their identifiers. |
| list_group_aliases([method, mount_point]) | List available group aliases by their identifiers. |
| list_groups([method, mount_point]) | List available groups by their identifiers. |
| list_groups_by_name([method, mount_point]) | List available groups by their names. |
| list_named_keys([mount_point]) | List all named keys. |
| list_roles([mount_point]) | This endpoint will list all signing keys. |
| lookup_entity([name, entity_id, alias_id, ...]) | Query an entity based on the given criteria. |
| lookup_group([name, group_id, alias_id, ...]) | Query a group based on the given criteria. |
| merge_entities(from_entity_ids, to_entity_id) | Merge many entities into one entity. |
| read_active_public_keys([mount_point]) | Retrieve the public portion of named keys. |
| read_entity(entity_id[, mount_point]) | Query an entity by its identifier. |
| read_entity_alias(alias_id[, mount_point]) | Query the entity alias by its identifier. |
| read_entity_by_name(name[, mount_point]) | Query an entity by its name. |
| read_group(group_id[, mount_point]) | Query the group by its identifier. |
| read_group_alias(alias_id[, mount_point]) | Query the group alias by its identifier. |
| read_group_by_name(name[, mount_point]) | Query a group by its name. |
| read_named_key(name[, mount_point]) | Query a named key and returns its configurations. |
| read_role(name[, mount_point]) | Query a role and returns its configuration. |
| read_tokens_backend_configuration([mount_point]) | Query default identity tokens configurations. |
| read_well_known_configurations([mount_point]) | Retrieve a set of claims about the identity tokens' configuration. |
| rotate_named_key(name, verification_ttl[, ...]) | Rotate a named key. |
| update_entity(entity_id[, name, metadata, ...]) | Update an existing entity. |

continues on next page

Table 37 – continued from previous page

| | |
|---|---|
| update_entity_alias(alias_id, name, ...[, ...]) | Update an existing entity alias. |
| update_group(group_id, name[, group_type, ...]) | Update an existing group. |
| update_group_alias(entity_id, name[, ...]) | Update an existing group alias. |
| validate_member_id_params_for_group_type(...) | Determine whether member ID parameters can be sent with a group create / update request. |

**configure_tokens_backend**(*issuer=None*, *mount_point='identity'*)

Update configurations for OIDC-compliant identity tokens issued by Vault.

**Supported methods:** POST: {mount_point}/oidc/config.

**Parameters**

- **issuer** (*str | unicode*) – Issuer URL to be used in the iss claim of the token. If not set, Vault's api_addr will be used. The issuer is a case sensitive URL using the https scheme that contains scheme, host, and optionally, port number and path components, but no query or fragment components.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The a dict or the response of the configure_tokens_backend request. dict returned when messages are included in the response body.

**Return type** requests.Response

**create_named_key**(*name*, *rotation_period='24h'*, *verification_ttl='24h'*, *allowed_client_ids=None*, *algorithm='RS256'*, *mount_point='identity'*)

Create or update a named key which is used by a role to sign tokens.

**Supported methods:** POST: {mount_point}/oidc/key/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the named key.
- **rotation_period** (*str | unicode*) – How often to generate a new signing key. Can be specified as a number of seconds or as a time string like "30m" or "6h".
- **verification_ttl** (*str | unicode*) – Controls how long the public portion of a signing key will be available for verification after being rotated.
- **allowed_client_ids** (*list*) – List of role client ids allowed to use this key for signing. If empty, no roles are allowed. If "*", all roles are allowed.
- **algorithm** (*str | unicode*) – Signing algorithm to use. Allowed values are: RS256 (default), RS384, RS512, ES256, ES384, ES512, EdDSA.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_a_named_key request.

**Return type** dict

**create_or_update_entity**(*name*, *entity_id=None*, *metadata=None*, *policies=None*, *disabled=None*, *mount_point='identity'*)

Create or update an Entity.

**Supported methods:** POST: /{mount_point}/entity. Produces: 200 application/json

**Parameters**

- **entity_id** (*str | unicode*) – ID of the entity. If set, updates the corresponding existing entity.
- **name** (*str | unicode*) – Name of the entity.
- **metadata** (*dict*) – Metadata to be associated with the entity.
- **policies** (*str | unicode*) – Policies to be tied to the entity.
- **disabled** (*bool*) – Whether the entity is disabled. Disabled entities' associated tokens cannot be used, but are not revoked.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response for creates, the generic response object for updates, of the request.

**Return type** dict | requests.Response

**create_or_update_entity_alias**(*name*, *canonical_id*, *mount_accessor*, *alias_id=None*, *mount_point='identity'*)

Create a new alias for an entity.

**Supported methods:** POST: /{mount_point}/entity-alias. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the alias. Name should be the identifier of the client in the authentication source. For example, if the alias belongs to userpass backend, the name should be a valid username within userpass backend. If alias belongs to GitHub, it should be the GitHub username.
- **alias_id** (*str | unicode*) – ID of the entity alias. If set, updates the corresponding entity alias.
- **canonical_id** (*str | unicode*) – Entity ID to which this alias belongs to.
- **mount_accessor** (*str | unicode*) – Accessor of the mount to which the alias should belong to.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**create_or_update_entity_by_name**(*name*, *metadata=None*, *policies=None*, *disabled=None*, *mount_point='identity'*)

Create or update an entity by a given name.

**Supported methods:** POST: /{mount_point}/entity/name/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the entity.
- **metadata** (*dict*) – Metadata to be associated with the entity.
- **policies** (*str | unicode*) – Policies to be tied to the entity.
- **disabled** (*bool*) – Whether the entity is disabled. Disabled entities' associated tokens cannot be used, but are not revoked.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response for creates, the generic response of the request for updates.

**Return type** requests.Response | dict

**create_or_update_group**(*name*, *group_id=None*, *group_type='internal'*, *metadata=None*, *policies=None*, *member_group_ids=None*, *member_entity_ids=None*, *mount_point='identity'*)

Create or update a Group.

**Supported methods:** POST: /{mount_point}/group. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the group.
- **group_id** (*str | unicode*) – ID of the group. If set, updates the corresponding existing group.
- **group_type** (*str | unicode*) – Type of the group, internal or external. Defaults to internal.
- **metadata** (*dict*) – Metadata to be associated with the group.
- **policies** (*str | unicode*) – Policies to be tied to the group.
- **member_group_ids** (*str | unicode*) – Group IDs to be assigned as group members.
- **member_entity_ids** (*str | unicode*) – Entity IDs to be assigned as group members.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response where available, otherwise the generic response object, of the request.

**Return type** dict | requests.Response

**create_or_update_group_alias**(*name*, *alias_id=None*, *mount_accessor=None*, *canonical_id=None*, *mount_point='identity'*)

Creates or update a group alias.

**Supported methods:** POST: /{mount_point}/group-alias. Produces: 200 application/json

**Parameters**

- **alias_id** (*str | unicode*) – ID of the group alias. If set, updates the corresponding existing group alias.
- **name** (*str | unicode*) – Name of the group alias.
- **mount_accessor** (*str | unicode*) – Mount accessor to which this alias belongs to
- **canonical_id** (*str | unicode*) – ID of the group to which this is an alias.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**create_or_update_group_by_name**(*name*, *group_type='internal'*, *metadata=None*, *policies=None*, *member_group_ids=None*, *member_entity_ids=None*, *mount_point='identity'*)

Create or update a group by its name.

**Supported methods:** POST: /{mount_point}/group/name/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the group.
- **group_type** (*str | unicode*) – Type of the group, internal or external. Defaults to internal.
- **metadata** (*dict*) – Metadata to be associated with the group.
- **policies** (*str | unicode*) – Policies to be tied to the group.
- **member_group_ids** (*str | unicode*) – Group IDs to be assigned as group members.
- **member_entity_ids** (*str | unicode*) – Entity IDs to be assigned as group members.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**create_or_update_role**(*name*, *key*, *template=None*, *client_id=None*, *ttl='24h'*, *mount_point='identity'*)

Create or update a role.

ID tokens are generated against a role and signed against a named key.

**Supported methods:** POST: {mount_point}/oidc/role/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **key** (*str | unicode*) – A configured named key, the key must already exist.
- **template** (*str | unicode*) – The template string to use for generating tokens. This may be in stringified JSON or base64 format.
- **client_id** (*str | unicode*) – Optional client ID. A random ID will be generated if left unset.
- **ttl** (*str | unicode*) – TTL of the tokens generated against the role. Can be specified as a number of seconds or as a time string like "30m" or "6h".
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_or_update_a_role request.

**Return type** dict

**delete_entity**(*entity_id*, *mount_point='identity'*)

Delete an entity and all its associated aliases.

**Supported methods:** DELETE: /{mount_point}/entity/id/:id. Produces: 204 (empty body)

**Parameters**

- **entity_id** (*str*) – Identifier of the entity.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_entity_alias**(*alias_id*, *mount_point='identity'*)
Delete a entity alias.

**Supported methods:** DELETE: /{mount_point}/entity-alias/id/{alias_id}. Produces: 204 (empty body)

**Parameters**

- **alias_id** (*str | unicode*) – Identifier of the entity.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_entity_by_name**(*name*, *mount_point='identity'*)
Delete an entity and all its associated aliases, given the entity name.

**Supported methods:** DELETE: /{mount_point}/entity/name/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the entity.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_group**(*group_id*, *mount_point='identity'*)
Delete a group.

**Supported methods:** DELETE: /{mount_point}/group/id/{id}. Produces: 204 (empty body)

**Parameters**

- **group_id** (*str | unicode*) – Identifier of the entity.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_group_alias**(*entity_id*, *mount_point='identity'*)
Delete a group alias.

**Supported methods:** DELETE: /{mount_point}/group-alias/id/{id}. Produces: 204 (empty body)

**Parameters**

- **entity_id** (*str | unicode*) – ID of the group alias.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_group_by_name**(*name*, *mount_point='identity'*)
Delete a group, given its name.

**Supported methods:** DELETE: /{mount_point}/group/name/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Name of the group.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_named_key**(*name*, *mount_point='identity'*)
Delete a named key.

**Supported methods:** DELETE: {mount_point}/oidc/key/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the key.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the delete_a_named_key request.

**Return type** dict

**delete_role**(*name*, *mount_point='identity'*)
Deletes a role.

**Supported methods:** DELETE: {mount_point}/oidc/role/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the delete_a_role request.

**Return type** dict

**generate_signed_id_token**(*name*, *mount_point='identity'*)
Generate a signed ID (OIDC) token.

**Supported methods:** GET: {mount_point}/oidc/token/:name.

**Parameters**

- **name** (*str | unicode*) – The name of the role against which to generate a signed ID token
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the generate_a_signed_id_token request.

**Return type** dict

**introspect_signed_id_token**(*token*, *client_id=None*, *mount_point='identity'*)
Verify the authenticity and active state of a signed ID token.

**Supported methods:** POST: {mount_point}/oidc/introspect.

**Parameters**

- **token** (*str | unicode*) – A signed OIDC compliant ID token
- **client_id** (*str | unicode*) – Specifying the client ID optimizes validation time
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the introspect_a_signed_id_token request.

**Return type** dict

**list_entities**(*method='LIST'*, *mount_point='identity'*)
List available entities entities by their identifiers.

**Parameters**

- **method** (*str | unicode*) – Supported methods: LIST: /{mount_point}/entity/id. Produces: 200 application/json GET: /{mount_point}/entity/id?list=true. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_entities_by_name**(*method='LIST'*, *mount_point='identity'*)
List available entities by their names.

**Parameters**

- **method** (*str | unicode*) – Supported methods: LIST: /{mount_point}/entity/name. Produces: 200 application/json GET: /{mount_point}/entity/name?list=true. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_entity_aliases**(*method='LIST'*, *mount_point='identity'*)
List available entity aliases by their identifiers.

**Parameters**

- **method** (*str | unicode*) – Supported methods: LIST: /{mount_point}/entity-alias/id. Produces: 200 application/json GET: /{mount_point}/entity-alias/id?list=true. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The the JSON response of the request.

**Return type** dict

**list_group_aliases**(*method='LIST'*, *mount_point='identity'*)
List available group aliases by their identifiers.

**Parameters**

- **method** (*str | unicode*) – Supported methods: LIST: /{mount_point}/group-alias/id. Produces: 200 application/json GET: /{mount_point}/group-alias/id?list=true. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The "data" key from the JSON response of the request.

**Return type** dict

**list_groups**(*method='LIST'*, *mount_point='identity'*)
List available groups by their identifiers.

**Parameters**

- **method** (*str | unicode*) – Supported methods: LIST: /{mount_point}/group/id. Produces: 200 application/json GET: /{mount_point}/group/id?list=true. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_groups_by_name**(*method='LIST'*, *mount_point='identity'*)
List available groups by their names.

**Parameters**

- **method** (*str | unicode*) – Supported methods: LIST: /{mount_point}/group/name. Produces: 200 application/json GET: /{mount_point}/group/name?list=true. Produces: 200 application/json
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_named_keys**(*mount_point='identity'*)
List all named keys.

**Supported methods:** LIST: {mount_point}/oidc/key.

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the list_named_keys request.

**Return type** dict

**list_roles**(*mount_point='identity'*)
This endpoint will list all signing keys.

**Supported methods:** LIST: {mount_point}/oidc/role.

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the list_roles request.

**Return type** dict

**lookup_entity**(*name=None*, *entity_id=None*, *alias_id=None*, *alias_name=None*, *alias_mount_accessor=None*, *mount_point='identity'*)

Query an entity based on the given criteria.

The criteria can be name, id, alias_id, or a combination of alias_name and alias_mount_accessor.

**Supported methods:** POST: /{mount_point}/lookup/entity. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the entity.
- **entity_id** (*str | unicode*) – ID of the entity.
- **alias_id** (*str | unicode*) – ID of the alias.
- **alias_name** (*str | unicode*) – Name of the alias. This should be supplied in conjunction with alias_mount_accessor.
- **alias_mount_accessor** (*str | unicode*) – Accessor of the mount to which the alias belongs to. This should be supplied in conjunction with alias_name.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request if a entity / entity alias is found in the lookup, None otherwise.

**Return type** dict | None

**lookup_group**(*name=None*, *group_id=None*, *alias_id=None*, *alias_name=None*, *alias_mount_accessor=None*, *mount_point='identity'*)

Query a group based on the given criteria.

The criteria can be name, id, alias_id, or a combination of alias_name and alias_mount_accessor.

**Supported methods:** POST: /{mount_point}/lookup/group. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the group.
- **group_id** (*str | unicode*) – ID of the group.
- **alias_id** (*str | unicode*) – ID of the alias.
- **alias_name** (*str | unicode*) – Name of the alias. This should be supplied in conjunction with alias_mount_accessor.
- **alias_mount_accessor** (*str | unicode*) – Accessor of the mount to which the alias belongs to. This should be supplied in conjunction with alias_name.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request if a group / group alias is found in the lookup, None otherwise.

**Return type** dict | None

**merge_entities**(*from_entity_ids*, *to_entity_id*, *force=None*, *mount_point='identity'*)

Merge many entities into one entity.

**Supported methods:** POST: /{mount_point}/entity/merge. Produces: 204 (empty body)

**Parameters**

- **from_entity_ids** (*array*) – Entity IDs which needs to get merged.
- **to_entity_id** (*str | unicode*) – Entity ID into which all the other entities need to get merged.
- **force** (*bool*) – Setting this will follow the 'mine' strategy for merging MFA secrets. If there are secrets of the same type both in entities that are merged from and in entity into which all others are getting merged, secrets in the destination will be unaltered. If not set, this API will throw an error containing all the conflicts.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**read_active_public_keys**(*mount_point='identity'*)
Retrieve the public portion of named keys.

Clients can use this to validate the authenticity of an identity token.

**Supported methods:** GET: {mount_point}/oidc/.well-known/openid-configuration.

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the read_active_public_keys request.

**Return type** dict

**read_entity**(*entity_id*, *mount_point='identity'*)
Query an entity by its identifier.

**Supported methods:** GET: /auth/{mount_point}/entity/id/{id}. Produces: 200 application/json

**Parameters**

- **entity_id** (*str*) – Identifier of the entity.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_entity_alias**(*alias_id*, *mount_point='identity'*)
Query the entity alias by its identifier.

**Supported methods:** GET: /{mount_point}/entity-alias/id/{id}. Produces: 200 application/json

**Parameters**

- **alias_id** (*str | unicode*) – Identifier of entity alias.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_entity_by_name**(*name*, *mount_point='identity'*)
Query an entity by its name.

**Supported methods:** GET: /{mount_point}/entity/name/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the entity.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**read_group**(*group_id*, *mount_point='identity'*)
Query the group by its identifier.

**Supported methods:** GET: /{mount_point}/group/id/{id}. Produces: 200 application/json

**Parameters**

- **group_id** (*str | unicode*) – Identifier of the group.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**read_group_alias**(*alias_id*, *mount_point='identity'*)
Query the group alias by its identifier.

**Supported methods:** GET: /{mount_point}/group-alias/id/:id. Produces: 200 application/json

**Parameters**

- **alias_id** (*str | unicode*) – ID of the group alias.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_group_by_name**(*name*, *mount_point='identity'*)
Query a group by its name.

**Supported methods:** GET: /{mount_point}/group/name/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the group.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_named_key**(*name*, *mount_point='identity'*)
Query a named key and returns its configurations.

**Supported methods:** GET: {mount_point}/oidc/key/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the key.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the read_a_named_key request.

**Return type** dict

**read_role**(*name*, *mount_point='identity'*)
Query a role and returns its configuration.

**Supported methods:** GET: {mount_point}/oidc/role/:name.

**Parameters**

- **name** (*str | unicode*) – Name of the role.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the read_a_role request.

**Return type** dict

**read_tokens_backend_configuration**(*mount_point='identity'*)
Query vault identity tokens configurations.

**Supported methods:** GET: {mount_point}/oidc/config.

**Returns** The response of the read_tokens_backend_configuration request.

**Return type** dict

**read_well_known_configurations**(*mount_point='identity'*)
Retrieve a set of claims about the identity tokens' configuration.

The response is a compliant OpenID Provider Configuration Response.

**Supported methods:** GET: {mount_point}/oidc/.well-known/openid-configuration.

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the read_well_known_configurations request.

**Return type** dict

**rotate_named_key**(*name*, *verification_ttl*, *mount_point='identity'*)
Rotate a named key.

**Supported methods:** POST: {mount_point}/oidc/key/:name/rotate.

**Parameters**

- **name** (*str | unicode*) – Name of the key to be rotated.
- **verification_ttl** (*str | unicode*) – Controls how long the public portion of the key will be available for verification after being rotated. Setting verification_ttl here will override the verification_ttl set on the key.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the rotate_a_named_key request.

**Return type** dict

**update_entity**(*entity_id*, *name=None*, *metadata=None*, *policies=None*, *disabled=None*, *mount_point='identity'*)

Update an existing entity.

**Supported methods:** POST: /{mount_point}/entity/id/{id}. Produces: 200 application/json

**Parameters**

- **entity_id** (*str | unicode*) – Identifier of the entity.
- **name** (*str | unicode*) – Name of the entity.
- **metadata** (*dict*) – Metadata to be associated with the entity.
- **policies** (*str | unicode*) – Policies to be tied to the entity.
- **disabled** (*bool*) – Whether the entity is disabled. Disabled entities' associated tokens cannot be used, but are not revoked.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response where available, otherwise the generic response object, of the request.

**Return type** dict | requests.Response

**update_entity_alias**(*alias_id*, *name*, *canonical_id*, *mount_accessor*, *mount_point='identity'*)

Update an existing entity alias.

**Supported methods:** POST: /{mount_point}/entity-alias/id/{id}. Produces: 200 application/json

**Parameters**

- **alias_id** (*str | unicode*) – Identifier of the entity alias.
- **name** (*str | unicode*) – Name of the alias. Name should be the identifier of the client in the authentication source. For example, if the alias belongs to userpass backend, the name should be a valid username within userpass backend. If alias belongs to GitHub, it should be the GitHub username.
- **canonical_id** (*str | unicode*) – Entity ID to which this alias belongs to.
- **mount_accessor** (*str | unicode*) – Accessor of the mount to which the alias should belong to.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response where available, otherwise the generic response object, of the request.

**Return type** dict | requests.Response

**update_group**(*group_id*, *name*, *group_type='internal'*, *metadata=None*, *policies=None*, *member_group_ids=None*, *member_entity_ids=None*, *mount_point='identity'*)

Update an existing group.

**Supported methods:** POST: /{mount_point}/group/id/{id}. Produces: 200 application/json

**Parameters**

- **group_id** (*str | unicode*) – Identifier of the entity.
- **name** (*str | unicode*) – Name of the group.

- **group_type** (*str | unicode*) – Type of the group, internal or external. Defaults to internal.
- **metadata** (*dict*) – Metadata to be associated with the group.
- **policies** (*str | unicode*) – Policies to be tied to the group.
- **member_group_ids** (*str | unicode*) – Group IDs to be assigned as group members.
- **member_entity_ids** (*str | unicode*) – Entity IDs to be assigned as group members.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response where available, otherwise the generic response object, of the request.

**Return type** dict | requests.Response

**update_group_alias**(*entity_id*, *name*, *mount_accessor=None*, *canonical_id=None*, *mount_point='identity'*)

Update an existing group alias.

**Supported methods:** POST: /{mount_point}/group-alias/id/{id}. Produces: 200 application/json

**Parameters**

- **entity_id** (*str | unicode*) – ID of the group alias.
- **name** (*str | unicode*) – Name of the group alias.
- **mount_accessor** (*str | unicode*) – Mount accessor to which this alias belongs toMount accessor to which this alias belongs to.
- **canonical_id** (*str | unicode*) – ID of the group to which this is an alias.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**static validate_member_id_params_for_group_type**(*group_type*, *params*, *member_group_ids*, *member_entity_ids*)

Determine whether member ID parameters can be sent with a group create / update request.

These parameters are only allowed for the internal group type. If they're set for an external group type, Vault returns a "error" response.

**Parameters**

- **group_type** (*str | unicode*) – Type of the group, internal or external
- **params** (*dict*) – Params dict to conditionally add the member entity/group ID's to.
- **member_group_ids** (*str | unicode*) – Group IDs to be assigned as group members.
- **member_entity_ids** (*str | unicode*) – Entity IDs to be assigned as group members.

**Returns** Params dict with conditionally added member entity/group ID's.

**Return type** dict

**class** hvac.api.secrets_engines.**Kv**(*adapter*, *default_kv_version='2'*)
Bases: hvac.api.vault_api_base.VaultApiBase

Class containing methods for the key/value secrets_engines backend API routes. Reference: https://www.vaultproject.io/docs/secrets/kv/index.html **Methods**

| | |
|---|---|
| *__init__*(adapter[, default_kv_version]) | Create a new Kv instance. |

**Attributes**

| | |
|---|---|
| *allowed_kv_versions* | Built-in mutable sequence. |
| *default_kv_version* | |
| *v1* | Accessor for kv version 1 class / method. |
| *v2* | Accessor for kv version 2 class / method. |

**__init__**(*adapter*, *default_kv_version='2'*)
Create a new Kv instance.

**Parameters**

- **adapter** (hvac.adapters.Adapter) – Instance of *hvac.adapters.Adapter*; used for performing HTTP requests.
- **default_kv_version** (*str | unicode*) – KV version number (e.g., '1') to use as the default when accessing attributes/methods under this class.

**allowed_kv_versions = ['1', '2']**

**property default_kv_version**

**property v1**
Accessor for kv version 1 class / method. Provided via the hvac.api.secrets_engines.kv_v1.KvV1 class.

**Returns** This Kv instance's associated KvV1 instance.

**Return type** hvac.api.secrets_engines.kv_v1.KvV1

**property v2**
Accessor for kv version 2 class / method. Provided via the hvac.api.secrets_engines.kv_v2.KvV2 class.

**Returns** This Kv instance's associated KvV2 instance.

**Return type** hvac.api.secrets_engines.kv_v2.KvV2

**class** hvac.api.secrets_engines.**KvV1**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

KV Secrets Engine - Version 1 (API).

Reference: https://www.vaultproject.io/api/secrets/kv/kv-v1.html **Methods**

| | |
|---|---|
| *create_or_update_secret*(path, secret[, ...]) | Store a secret at the specified location. |
| *delete_secret*(path[, mount_point]) | Delete the secret at the specified location. |
| *list_secrets*(path[, mount_point]) | Return a list of key names at the specified location. |
| *read_secret*(path[, mount_point]) | Retrieve the secret at the specified location. |

**create_or_update_secret**(*path*, *secret*, *method=None*, *mount_point='secret'*)
Store a secret at the specified location.

If the value does not yet exist, the calling token must have an ACL policy granting the create capability. If the value already exists, the calling token must have an ACL policy granting the update capability.

**Supported methods:** POST: /{mount_point}/{path}. Produces: 204 (empty body) PUT: /{mount_point}/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secrets to create/update. This is specified as part of the URL.
- **secret** (*dict*) – Specifies keys, paired with associated values, to be held at the given location. Multiple key/value pairs can be specified, and all will be returned on a read operation. A key called ttl will trigger some special behavior. See the Vault KV secrets engine documentation for details.
- **method** (*str | unicode*) – Optional parameter to explicitly request a POST (create) or PUT (update) request to the selected kv secret engine. If no argument is provided for this parameter, hvac attempts to intelligently determine which method is appropriate.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the create_or_update_secret request.

**Return type** requests.Response

**delete_secret**(*path*, *mount_point='secret'*)
Delete the secret at the specified location.

**Supported methods:** DELETE: /{mount_point}/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to delete. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the delete_secret request.

**Return type** requests.Response

**list_secrets**(*path*, *mount_point='secret'*)
Return a list of key names at the specified location.

Folders are suffixed with /. The input must be a folder; list on a file will not return a value. Note that no policy-based filtering is performed on keys; do not encode sensitive information in key names. The values themselves are not accessible via this command.

**Supported methods:** LIST: /{mount_point}/{path}. Produces: 200 application/json

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secrets to list. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the list_secrets request.

**Return type** dict

**read_secret**(*path*, *mount_point='secret'*)
Retrieve the secret at the specified location.

**Supported methods:** GET: /{mount_point}/{path}. Produces: 200 application/json

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to read. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the read_secret request.

**Return type** dict

**class** hvac.api.secrets_engines.**KvV2**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

KV Secrets Engine - Version 2 (API).

Reference: https://www.vaultproject.io/api/secret/kv/kv-v2.html **Methods**

| | |
|---|---|
| *configure*([max_versions, cas_required, ...]) | Configure backend level settings that are applied to every key in the key-value store. |
| *create_or_update_secret*(path, secret[, cas, ...]) | Create a new version of a secret at the specified location. |
| *delete_latest_version_of_secret*(path[, ...]) | Issue a soft delete of the secret's latest version at the specified location. |
| *delete_metadata_and_all_versions*(path[, ...]) | Delete (permanently) the key metadata and all version data for the specified key. |
| *delete_secret_versions*(path, versions[, ...]) | Issue a soft delete of the specified versions of the secret. |
| *destroy_secret_versions*(path, versions[, ...]) | Permanently remove the specified version data and numbers for the provided path from the key-value store. |
| *list_secrets*(path[, mount_point]) | Return a list of key names at the specified location. |
| *patch*(path, secret[, mount_point]) | Set or update data in the KV store without overwriting. |
| *read_configuration*([mount_point]) | Read the KV Version 2 configuration. |
| *read_secret*(path[, mount_point]) | |
| *read_secret_metadata*(path[, mount_point]) | Retrieve the metadata and versions for the secret at the specified path. |
| *read_secret_version*(path[, version, mount_point]) | Retrieve the secret at the specified location. |
| *undelete_secret_versions*(path, versions[, ...]) | Undelete the data for the provided version and path in the key-value store. |
| *update_metadata*(path[, max_versions, ...]) | Updates the max_versions of cas_required setting on an existing path. |

**configure**(*max_versions=10*, *cas_required=None*, *delete_version_after='0s'*, *mount_point='secret'*)
Configure backend level settings that are applied to every key in the key-value store.

**Supported methods:** POST: /{mount_point}/config. Produces: 204 (empty body)

**Parameters**

- **max_versions** (*int*) – The number of versions to keep per key. This value applies to all keys, but a key's metadata setting can overwrite this value. Once a key has more than the configured allowed versions the oldest version will be permanently deleted. Defaults to 10.
- **cas_required** (*bool*) – If true all keys will require the cas parameter to be set on all write requests.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.
- **delete_version_after** (*str*) – Specifies the length of time before a version is deleted. Accepts Go duration format string. Defaults to "0s" (i.e., disabled).

**Returns** The response of the request.

**Return type** requests.Response

**create_or_update_secret**(*path*, *secret*, *cas=None*, *mount_point='secret'*)

Create a new version of a secret at the specified location.

If the value does not yet exist, the calling token must have an ACL policy granting the create capability. If the value already exists, the calling token must have an ACL policy granting the update capability.

**Supported methods:** POST: /{mount_point}/data/{path}. Produces: 200 application/json

**Parameters**

- **path** (*str | unicode*) – Path
- **cas** (*int*) – Set the "cas" value to use a Check-And-Set operation. If not set the write will be allowed. If set to 0 a write will only be allowed if the key doesn't exist. If the index is non-zero the write will only be allowed if the key's current version matches the version specified in the cas parameter.
- **secret** (*dict*) – The contents of the "secret" dict will be stored and returned on read.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**delete_latest_version_of_secret**(*path*, *mount_point='secret'*)

Issue a soft delete of the secret's latest version at the specified location.

This marks the version as deleted and will stop it from being returned from reads, but the underlying data will not be removed. A delete can be undone using the undelete path.

**Supported methods:** DELETE: /{mount_point}/data/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to delete. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_metadata_and_all_versions**(*path*, *mount_point='secret'*)
Delete (permanently) the key metadata and all version data for the specified key.

All version history will be removed.

**Supported methods:** DELETE: /{mount_point}/metadata/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to delete. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**delete_secret_versions**(*path*, *versions*, *mount_point='secret'*)
Issue a soft delete of the specified versions of the secret.

This marks the versions as deleted and will stop them from being returned from reads, but the underlying data will not be removed. A delete can be undone using the undelete path.

**Supported methods:** POST: /{mount_point}/delete/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to delete. This is specified as part of the URL.
- **versions** (*int*) – The versions to be deleted. The versioned data will not be deleted, but it will no longer be returned in normal get requests.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**destroy_secret_versions**(*path*, *versions*, *mount_point='secret'*)
Permanently remove the specified version data and numbers for the provided path from the key-value store.

**Supported methods:** POST: /{mount_point}/destroy/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to destroy. This is specified as part of the URL.
- **versions** (*list of int*) – The versions to destroy. Their data will be permanently deleted.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**list_secrets**(*path*, *mount_point='secret'*)
Return a list of key names at the specified location.

Folders are suffixed with /. The input must be a folder; list on a file will not return a value. Note that no policy-based filtering is performed on keys; do not encode sensitive information in key names. The values themselves are not accessible via this command.

**Supported methods:** LIST: /{mount_point}/metadata/{path}. Produces: 200 application/json

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secrets to list. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**patch**(*path*, *secret*, *mount_point='secret'*)

Set or update data in the KV store without overwriting.

**Parameters**

- **path** (*str | unicode*) – Path
- **secret** (*dict*) – The contents of the "secret" dict will be stored and returned on read.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the create_or_update_secret request.

**Return type** dict

**read_configuration**(*mount_point='secret'*)

Read the KV Version 2 configuration.

**Supported methods:** GET: /auth/{mount_point}/config. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_secret**(*path*, *mount_point='secret'*)

**read_secret_metadata**(*path*, *mount_point='secret'*)

Retrieve the metadata and versions for the secret at the specified path.

**Supported methods:** GET: /{mount_point}/metadata/{path}. Produces: 200 application/json

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to read. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_secret_version**(*path*, *version=None*, *mount_point='secret'*)

Retrieve the secret at the specified location.

**Supported methods:** GET: /{mount_point}/data/{path}. Produces: 200 application/json

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to read. This is specified as part of the URL.
- **version** (*int*) – Specifies the version to return. If not set the latest version is returned.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**undelete_secret_versions**(*path*, *versions*, *mount_point='secret'*)

Undelete the data for the provided version and path in the key-value store.

This restores the data, allowing it to be returned on get requests.

**Supported methods:** POST: /{mount_point}/undelete/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path of the secret to undelete. This is specified as part of the URL.
- **versions** (*list of int*) – The versions to undelete. The versions will be restored and their data will be returned on normal get requests.
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**update_metadata**(*path*, *max_versions=None*, *cas_required=None*, *delete_version_after='0s'*, *mount_point='secret'*)

Updates the max_versions of cas_required setting on an existing path.

**Supported methods:** POST: /{mount_point}/metadata/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Path
- **max_versions** (*int*) – The number of versions to keep per key. If not set, the backend's configured max version is used. Once a key has more than the configured allowed versions the oldest version will be permanently deleted.
- **cas_required** (*bool*) – If true the key will require the cas parameter to be set on all write requests. If false, the backend's configuration will be used.
- **delete_version_after** (*str*) – Specifies the length of time before a version is deleted. Accepts Go duration format string. Defaults to "0s" (i.e., disabled).
- **mount_point** (*str | unicode*) – The "path" the secret engine was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**class** hvac.api.secrets_engines.**Pki**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Pki Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/pki/index.html **Methods**

| | |
|---|---|
| *create_or_update_role*(name[, extra_params, ...]) | Create/Update Role. |
| *delete_role*(name[, mount_point]) | Delete Role. |
| *delete_root*([mount_point]) | Delete Root. |
| *generate_certificate*(name, common_name[, ...]) | Generate Certificate. |
| *generate_intermediate*(type, common_name[, ...]) | Generate Intermediate. |
| *generate_root*(type, common_name[, ...]) | Generate Root. |
| *list_certificates*([mount_point]) | List Certificates. |
| *list_roles*([mount_point]) | List Roles. |
| *read_ca_certificate*([mount_point]) | Read CA Certificate. |
| *read_ca_certificate_chain*([mount_point]) | Read CA Certificate Chain. |
| *read_certificate*(serial[, mount_point]) | Read Certificate. |
| *read_crl*([mount_point]) | Read CRL. |
| *read_crl_configuration*([mount_point]) | Read CRL Configuration. |
| *read_role*(name[, mount_point]) | Read Role. |
| *read_urls*([mount_point]) | Read URLs. |
| *revoke_certificate*(serial_number[, mount_point]) | Revoke Certificate. |
| *rotate_crl*([mount_point]) | Rotate CRLs. |
| *set_crl_configuration*([expiry, disable, ...]) | Set CRL Configuration. |
| *set_signed_intermediate*(certificate[, ...]) | Set Signed Intermediate. |
| *set_urls*(params[, mount_point]) | Set URLs. |
| *sign_certificate*(name, csr, common_name[, ...]) | Sign Certificate. |
| *sign_intermediate*(csr, common_name[, ...]) | Sign Intermediate. |
| *sign_self_issued*(certificate[, mount_point]) | Sign Self-Issued. |
| *sign_verbatim*(csr[, name, extra_params, ...]) | Sign Verbatim. |
| *submit_ca_information*(pem_bundle[, mount_point]) | Submit CA Information. |
| *tidy*([extra_params, mount_point]) | Tidy. |

**create_or_update_role**(*name*, *extra_params=None*, *mount_point='pki'*)

Create/Update Role.

Creates or updates the role definition.

**Supported methods:** POST: /{mount_point}/roles/{name}. Produces: 200 application/json

**Parameters**

- **name** – The name of the role to create.
- **extra_params** – A dictionary with extra parameters.
- **mount_point** – The "path" the method/backend was mounted on.

**Name name** str | unicode

**Name extra_params** dict

**Name mount_point** str | unicode

**Returns** The JSON response of the request.

**Rname** requests.Response

**delete_role**(*name*, *mount_point='pki'*)
Delete Role.

Deletes the role definition.

**Supported methods:** DELETE: /{mount_point}/roles/{name}. Produces: 200 application/json

**Parameters**

- **name** – The name of the role to delete.
- **mount_point** – The "path" the method/backend was mounted on.

**Name name** str | unicode

**Name mount_point** str | unicode

**Returns** The JSON response of the request.

**Return type** requests.Response

**delete_root**(*mount_point='pki'*)
Delete Root.

Deletes the current CA key.

**Supported methods:** DELETE: /{mount_point}/root. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**generate_certificate**(*name*, *common_name*, *extra_params=None*, *mount_point='pki'*, *wrap_ttl=None*)
Generate Certificate.

Generates a new set of credentials (private key and certificate) based on the role named in the endpoint.

**Supported methods:** POST: /{mount_point}/issue/{name}. Produces: 200 application/json

**Parameters**

- **name** – The name of the role to create the certificate against.
- **common_name** – The requested CN for the certificate.
- **extra_params** – A dictionary with extra parameters.
- **mount_point** – The "path" the method/backend was mounted on.
- **wrap_ttl** (*str | unicode*) – Specifies response wrapping token creation with duration. IE: '15s', '20m', '25h'.

**Name name** str | unicode

**Name common_name** str | unicode

**Name extra_params** dict

**Name mount_point** str | unicode

**Returns** The JSON response of the request.

**Return type** requests.Response

**generate_intermediate**(*type*, *common_name*, *extra_params=None*, *mount_point='pki'*, *wrap_ttl=None*)

Generate Intermediate.

Generates a new private key and a CSR for signing.

**Supported methods:** POST: /{mount_point}/intermediate/generate/{type}. Produces: 200 application/json

**Parameters**

- **type** (*str | unicode*) – Specifies the type to create. *exported* (private key also exported) or *internal*.
- **common_name** (*str | unicode*) – Specifies the requested CN for the certificate.
- **extra_params** (*dict*) – Dictionary with extra parameters.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.
- **wrap_ttl** (*str | unicode*) – Specifies response wrapping token creation with duration. IE: '15s', '20m', '25h'.

**Returns** The JSON response of the request.

**Return type** requests.Response

**generate_root**(*type*, *common_name*, *extra_params=None*, *mount_point='pki'*, *wrap_ttl=None*)

Generate Root.

Generates a new self-signed CA certificate and private key.

**Supported methods:** POST: /{mount_point}/root/generate/{type}. Produces: 200 application/json

**Parameters**

- **type** (*str | unicode*) – Specifies the type to create. *exported* (private key also exported) or *internal*.
- **common_name** (*str | unicode*) – The requested CN for the certificate.
- **extra_params** (*dict*) – A dictionary with extra parameters.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.
- **wrap_ttl** (*str | unicode*) – Specifies response wrapping token creation with duration. IE: '15s', '20m', '25h'.

**Returns** The JSON response of the request.

**Return type** requests.Response

**list_certificates**(*mount_point='pki'*)

List Certificates.

The list of the current certificates by serial number only.

**Supported methods:** LIST: /{mount_point}/certs. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_roles**(*mount_point='pki'*)
List Roles.

Get a list of available roles.

**Supported methods:** LIST: /{mount_point}/roles. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_ca_certificate**(*mount_point='pki'*)
Read CA Certificate.

Retrieves the CA certificate in raw DER-encoded form.

**Supported methods:** GET: /{mount_point}/ca/pem. Produces: String

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The certificate as pem.

**Return type** str

**read_ca_certificate_chain**(*mount_point='pki'*)
Read CA Certificate Chain.

Retrieves the CA certificate chain, including the CA in PEM format.

**Supported methods:** GET: /{mount_point}/ca_chain. Produces: String

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The certificate chain as pem.

**Return type** str

**read_certificate**(*serial*, *mount_point='pki'*)
Read Certificate.

Retrieves one of a selection of certificates.

**Supported methods:** GET: /{mount_point}/cert/{serial}. Produces: 200 application/json

**Parameters**

- **serial** (*str | unicode*) – the serial of the key to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_crl**(*mount_point='pki'*)
Read CRL.

Retrieves the current CRL in PEM format. This endpoint is an unauthenticated.

**Supported methods:** GET: /{mount_point}/crl/pem. Produces: 200 application/pkix-crl

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The content of the request e.g. CRL string representation.

**Return type** str

**read_crl_configuration**(*mount_point='pki'*)
Read CRL Configuration.

Getting the duration for which the generated CRL should be marked valid.

**Supported methods:** GET: /{mount_point}/config/crl. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_role**(*name*, *mount_point='pki'*)
Read Role.

Queries the role definition.

**Supported methods:** GET: /{mount_point}/roles/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – The name of the role to read.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_urls**(*mount_point='pki'*)
Read URLs.

Fetches the URLs to be encoded in generated certificates.

**Supported methods:** GET: /{mount_point}/config/urls. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**revoke_certificate**(*serial_number*, *mount_point='pki'*)
Revoke Certificate.

Revokes a certificate using its serial number.

**Supported methods:** POST: /{mount_point}/revoke. Produces: 200 application/json

**Parameters**

- **serial_number** – The serial number of the certificate to revoke.
- **mount_point** – The "path" the method/backend was mounted on.

**Name serial_number** str | unicode

**Name mount_point** str | unicode

**Returns** The JSON response of the request.

**Return type** requests.Response

**rotate_crl**(*mount_point='pki'*)
Rotate CRLs.

Forces a rotation of the CRL.

**Supported methods:** GET: /{mount_point}/crl/rotate. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**set_crl_configuration**(*expiry=None*, *disable=None*, *extra_params=None*, *mount_point='pki'*)
Set CRL Configuration.

Setting the duration for which the generated CRL should be marked valid. If the CRL is disabled, it will return a signed but zero-length CRL for any request. If enabled, it will re-build the CRL.

**Supported methods:** POST: /{mount_point}/config/crl. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**set_signed_intermediate**(*certificate*, *mount_point='pki'*)
Set Signed Intermediate.

Allows submitting the signed CA certificate corresponding to a private key generated via "Generate Intermediate"

**Supported methods:** POST: /{mount_point}/intermediate/set-signed. Produces: 200 application/json

**Parameters**

- **certificate** (*str | unicode*) – Specifies the certificate in PEM format.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**set_urls**(*params*, *mount_point='pki'*)
Set URLs.

Setting the issuing certificate endpoints, CRL distribution points, and OCSP server endpoints that will be encoded into issued certificates. You can update any of the values at any time without affecting the other existing values. To remove the values, simply use a blank string as the parameter.

**Supported methods:** POST: /{mount_point}/config/urls. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**sign_certificate**(*name*, *csr*, *common_name*, *extra_params=None*, *mount_point='pki'*)
Sign Certificate.

Signs a new certificate based upon the provided CSR and the supplied parameters.

**Supported methods:** POST: /{mount_point}/sign/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – The role to sign the certificate.
- **csr** (*str | unicode*) – The PEM-encoded CSR.
- **common_name** (*str | unicode*) – The requested CN for the certificate. If the CN is allowed by role policy, it will be issued.
- **extra_params** (*dict*) – A dictionary with extra parameters.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**sign_intermediate**(*csr*, *common_name*, *extra_params=None*, *mount_point='pki'*)
Sign Intermediate.

Issue a certificate with appropriate values for acting as an intermediate CA.

**Supported methods:** POST: /{mount_point}/root/sign-intermediate. Produces: 200 application/json

**Parameters**

- **csr** (*str | unicode*) – The PEM-encoded CSR.
- **common_name** (*str | unicode*) – The requested CN for the certificate.
- **extra_params** (*dict*) – Dictionary with extra parameters.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**sign_self_issued**(*certificate*, *mount_point='pki'*)
Sign Self-Issued.

Sign a self-issued certificate.

**Supported methods:** POST: /{mount_point}/root/sign-self-issued. Produces: 200 application/json

**Parameters**

- **certificate** (*str | unicode*) – The PEM-encoded self-issued certificate.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**sign_verbatim**(*csr*, *name=False*, *extra_params=None*, *mount_point='pki'*)
Sign Verbatim.

Signs a new certificate based upon the provided CSR.

**Supported methods:** POST: /{mount_point}/sign-verbatim. Produces: 200 application/json

**Parameters**

- **csr** (*str | unicode*) – The PEM-encoded CSR.
- **name** (*str | unicode*) – Specifies a role.
- **extra_params** (*dict*) – A dictionary with extra parameters.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**submit_ca_information**(*pem_bundle*, *mount_point='pki'*)
Submit CA Information.

Submitting the CA information for the backend.

**Supported methods:** POST: /{mount_point}/config/ca. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**tidy**(*extra_params=None*, *mount_point='pki'*)
Tidy.

Allows tidying up the storage backend and/or CRL by removing certificates that have expired and are past a certain buffer period beyond their expiration time.

**Supported methods:** POST: /{mount_point}/tidy. Produces: 200 application/json

**Parameters**

- **extra_params** (*dict*) – A dictionary with extra parameters.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** requests.Response

**class** hvac.api.secrets_engines.**RabbitMQ**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

RabbitMQ Secrets Engine (API). Reference: https://www.vaultproject.io/api/secret/rabbitmq/index.html **Methods**

| | |
|---|---|
| *configure*([connection_uri, username, ...]) | Configure shared information for the rabbitmq secrets engine. |
| *configure_lease*(ttl, max_ttl[, mount_point]) | This endpoint configures the lease settings for generated credentials. |
| *create_role*(name[, tags, vhosts, ...]) | This endpoint creates or updates the role definition. |
| *delete_role*(name[, mount_point]) | This endpoint deletes the role definition. |
| *generate_credentials*(name[, mount_point]) | This endpoint generates a new set of dynamic credentials based on the named role. |
| *read_role*(name[, mount_point]) | This endpoint queries the role definition. |

**configure**(*connection_uri=''*, *username=''*, *password=''*, *verify_connection=True*, *mount_point='rabbitmq'*)
Configure shared information for the rabbitmq secrets engine.

**Supported methods:** POST: /{mount_point}/config/connection. Produces: 204 (empty body)

**Parameters**

- **connection_uri** (*str | unicode*) – Specifies the RabbitMQ connection URI.
- **username** (*str | unicode*) – Specifies the RabbitMQ management administrator username.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: rabbitmq).

**Password** Specifies the RabbitMQ management administrator password.

**Verify_connection** Specifies whether to verify connection URI, username, and password.

**Returns** The response of the request.

**Return type** requests.Response

**configure_lease**(*ttl*, *max_ttl*, *mount_point='rabbitmq'*)
This endpoint configures the lease settings for generated credentials.

**Parameters**

- **ttl** (*int*) – Specifies the lease ttl provided in seconds.
- **max_ttl** (*int*) – Specifies the maximum ttl provided in seconds.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: rabbitmq).

**Returns** The JSON response of the request.

**Return type** requests.Response

**create_role**(*name*, *tags=''*, *vhosts=''*, *vhost_topics=''*, *mount_point='rabbitmq'*)
This endpoint creates or updates the role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to create.
- **tags** (*str | unicode*) – Specifies a comma-separated RabbitMQ management tags.
- **vhosts** (*str | unicode*) – pecifies a map of virtual hosts to permissions.
- **vhost_topics** (*str | unicode*) – Specifies a map of virtual hosts and exchanges to topic permissions.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: rabbitmq).

**Returns** The JSON response of the request.

**Return type** requests.Response

**delete_role**(*name*, *mount_point='rabbitmq'*)

This endpoint deletes the role definition. Even if the role does not exist, this endpoint will still return a successful response.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to delete.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: rabbitmq).

**Returns** The response of the request.

**Return type** requests.Response

**generate_credentials**(*name*, *mount_point='rabbitmq'*)

This endpoint generates a new set of dynamic credentials based on the named role.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to create credentials against.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: rabbitmq).

**Returns** The response of the request.

**Return type** requests.Response

**read_role**(*name*, *mount_point='rabbitmq'*)

This endpoint queries the role definition.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to read.
- **mount_point** (*str | unicode*) – Specifies the place where the secrets engine will be accessible (default: rabbitmq).

**Returns** The JSON response of the request.

**Return type** requests.Response

**class** hvac.api.secrets_engines.**SecretsEngines**(*adapter*)

Bases: hvac.api.vault_api_category.VaultApiCategory

Secrets Engines. **Attributes**

| | |
|---|---|
| *implemented_classes* | Built-in mutable sequence. |
| *unimplemented_classes* | Built-in mutable sequence. |

**implemented_classes = [<class 'hvac.api.secrets_engines.aws.Aws'>, <class 'hvac.api.se**

**unimplemented_classes = ['AliCloud', 'Azure', 'GcpKms', 'Nomad', 'Ssh', 'TOTP', 'Cassa**

**class** hvac.api.secrets_engines.**Transform**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Transform Secrets Engine (API).

Reference: https://www.vaultproject.io/api-docs/secret/transform **Methods**

| | |
|---|---|
| *check_tokenization*(role_name, value, ...[, ...]) | Determine if a provided plaintext value has an valid, unexpired tokenized value. |
| *create_or_update_alphabet*(name, alphabet[, ...]) | Create or update an alphabet with the given name. |
| *create_or_update_fpe_transformation*(name, ...) | Creates or update an FPE transformation with the given name. |
| *create_or_update_masking_transformation*(name, ...) | Creates or update a masking transformation with the given name. |
| *create_or_update_role*(name, transformations) | Creates or update the role with the given name. |
| *create_or_update_template*(name, ...[, ...]) | Creates or update a template with the given name. |
| *create_or_update_tokenization_store*(name, ...) | Create or update a storage configuration for use with tokenization. |
| *create_or_update_tokenization_transformation*(name, ...) | This endpoint creates or updates a tokenization transformation with the given name. |
| *create_or_update_transformation*(name, ...[, ...]) | Create or update a transformation with the given name. |
| *decode*(role_name[, value, transformation, ...]) | Decode the provided value using a named role. |
| *delete_alphabet*(name[, mount_point]) | Delete an existing alphabet by the given name. |
| *delete_role*(name[, mount_point]) | Delete an existing role by the given name. |
| *delete_template*(name[, mount_point]) | Delete an existing template by the given name. |
| *delete_transformation*(name[, mount_point]) | Delete an existing transformation by the given name. |
| *encode*(role_name[, value, transformation, ...]) | Encode the provided value using a named role. |
| *export_decoded_tokenization_state*(name[, ...]) | Start or continue retrieving an export of tokenization state, including the tokens and their decoded values. |
| *list_alphabets*([mount_point]) | List all existing alphabets in the secrets engine. |
| *list_roles*([mount_point]) | List all existing roles in the secrets engine. |
| *list_templates*([mount_point]) | List all existing templates in the secrets engine. |
| *list_tokenization_key_configuration*([...]) | List all tokenization keys. |
| *list_transformations*([mount_point]) | List all existing transformations in the secrets engine. |
| *read_alphabet*(name[, mount_point]) | Queries an existing alphabet by the given name. |
| *read_role*(name[, mount_point]) | Query an existing role by the given name. |
| *read_template*(name[, mount_point]) | Query an existing template by the given name. |
| *read_tokenization_key_configuration*(..., ...]) | Read tokenization key configuration for a particular transform. |
| *read_transformation*(name[, mount_point]) | Query an existing transformation by the given name. |

continues on next page

Table 45 – continued from previous page

| | |
|---|---|
| *restore_tokenization_state*(name, values[, ...]) | This endpoint restores previously snapshotted tokenization state values to the underlying store(s) of a tokenization transform. |
| *retrieve_token_metadata*(role_name, value, ...) | This endpoint retrieves metadata for a tokenized value using a named role. |
| *rotate_tokenization_key*(transform_name[, ...]) | Rotate the version of the named key. |
| *snapshot_tokenization_state*(name[, limit, ...]) | This endpoint starts or continues retrieving a snapshot of the stored state of a tokenization transform. |
| *trim_tokenization_key_version*(...[, mount_point]) | Trim older key versions setting a minimum version for the keyring. |
| *update_tokenization_key_config*(...[, ...]) | Allow the minimum key version to be set for decode operations. |
| *validate_token*(role_name, value, transformation) | Determine if a provided tokenized value is valid and unexpired. |

**check_tokenization**(*role_name*, *value*, *transformation*, *batch_input=None*, *mount_point='transform'*)

Determine if a provided plaintext value has an valid, unexpired tokenized value. Note that this cannot return the token, just confirm that a tokenized value exists. This endpoint is only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/tokenized/:role_name.

**Parameters**

- **role_name** (*str*) – the role name to use for this operation. This is specified as part of the URL.
- **value** (*str*) – the token to test for whether it has a valid tokenization.
- **transformation** (*str*) – the transformation within the role that should be used for this decode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.
- **batch_input** (*list*) – a list of items to be decoded in a single batch. When this parameter is set, the 'value' parameter is ignored. Instead, the aforementioned parameters should be provided within each object in the list.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the check_tokenization request.

**Return type** requests.Response

**create_or_update_alphabet**(*name*, *alphabet*, *mount_point='transform'*)

Create or update an alphabet with the given name.

If an alphabet with the name does not exist, it will be created. If the alphabet exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/alphabet/:name.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the transformation alphabet to create.
- **alphabet** (*str | unicode*) – the set of characters that can exist within the provided value and the encoded or decoded value for a FPE transformation.

- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the create_or_update_alphabet request.

**Return type** requests.Response

**create_or_update_fpe_transformation**(*name*, *template*, *tweak_source='supplied'*, *allowed_roles=None*, *mount_point='transform'*)

Creates or update an FPE transformation with the given name.

If a transformation with the name does not exist, it will be created. If the transformation exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/transformations/fpe/:name.

**Parameters**

- **name** (*str*) – The name of the transformation to create or update. This is part of the request URL.
- **template** (*str*) – The template name to use for matching value on encode and decode operations when using this transformation.
- **tweak_source** (*str*) – Specifies the source of where the tweak value comes from. Valid sources are: supplied, generated, and internal.
- **allowed_roles** (*list*) – A list of allowed roles that this transformation can be assigned to. A role using this transformation must exist in this list in order for encode and decode operations to properly function.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_or_update_fpe_transformation request.

**Return type** requests.Response

**create_or_update_masking_transformation**(*name*, *template*, *masking_character='*'*, *allowed_roles=None*, *mount_point='transform'*)

Creates or update a masking transformation with the given name. If a transformation with the name does not exist, it will be created. If the transformation exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/transformations/masking/:name.

**Parameters**

- **name** (*str*) – The name of the transformation to create or update. This is part of the request URL.
- **template** (*str*) – The template name to use for matching value on encode and decode operations when using this transformation.
- **masking_character** (*str*) – The character to use for masking. If multiple characters are provided, only the first one is used and the rest is ignored. Only used when the type is masking.
- **allowed_roles** (*list*) – A list of allowed roles that this transformation can be assigned to. A role using this transformation must exist in this list in order for encode and decode operations to properly function.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_or_update_masking_transformation request.

**Return type** requests.Response

**create_or_update_role**(*name*, *transformations*, *mount_point='transform'*)

Creates or update the role with the given name.

If a role with the name does not exist, it will be created. If the role exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/role/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the role to create. This is part of the request URL.
- **transformations** (*list*) – Specifies the transformations that can be used with this role. At least one transformation is required.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the create_or_update_role request.

**Return type** requests.Response

**create_or_update_template**(*name*, *template_type*, *pattern*, *alphabet*, *mount_point='transform'*)

Creates or update a template with the given name.

If a template with the name does not exist, it will be created. If the template exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/template/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the template to create.
- **template_type** (*str | unicode*) – Specifies the type of pattern matching to perform. The ony type currently supported by this backend is regex.
- **pattern** (*str | unicode*) – the pattern used to match a particular value. For regex type matching, capture group determines the set of character that should be matched against. Any matches outside of capture groups are retained post-transformation.
- **alphabet** (*str | unicode*) – the name of the alphabet to use when this template is used for FPE encoding and decoding operations.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the create_or_update_template request.

**Return type** requests.Response

**create_or_update_tokenization_store**(*name*, *driver*, *connection_string*, *username=None*, *password=None*, *type='sql'*, *supported_transformations=None*, *schema='public'*, *max_open_connections=4*, *max_idle_connections=4*, *max_connection_lifetime=0*, *mount_point='transform'*)

Create or update a storage configuration for use with tokenization. The database user configured here should only have permission to SELECT, INSERT, and UPDATE rows in the tables.

**Supported methods:** POST: /{mount_point}/store/:name.

**Parameters**

- **name** (*str*) – The name of the store to create or update.
- **type** (*str*) – Specifies the type of store. Currently only *sql* is supported.
- **driver** (*str*) – Specifies the database driver to use, and thus which SQL database type. Currently the supported options are *postgres* or *mysql*
- **supported_transformations** (*list(str)*) – The types of transformations this store can host. Currently only *tokenization* is supported.
- **connection_string** (*str*) – database connection string with template slots for username and password that Vault will use for locating and connecting to a database. Each database driver type has a different syntax for its connection strings.
- **username** (*str*) – username value to use when connecting to the database.
- **password** (*str*) – password value to use when connecting to the database.
- **schema** (*str*) – schema within the database to expect tokenization state tables.
- **max_open_connections** (*int*) – maximum number of connections to the database at any given time.
- **max_idle_connections** (*int*) – maximum number of idle connections to the database at any given time.
- **max_connection_lifetime** (*duration*) – means no limit.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_or_update_tokenization_store request.

**Return type** requests.Response

**create_or_update_tokenization_transformation**(*name*, *max_ttl=0*, *mapping_mode='default'*, *allowed_roles=None*, *stores=None*, *mount_point='transform'*)

This endpoint creates or updates a tokenization transformation with the given name. If a transformation with the name does not exist, it will be created. If the transformation exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/:name.

**Parameters**

- **max_ttl** (*str*) – The maximum TTL of a token. If 0 or unspecified, tokens may have no expiration.
- **mapping_mode** (*str*) – Specifies the mapping mode for stored tokenization values.
  - *default* is strongly recommended for highest security
  - *exportable* exportable allows for all plaintexts to be decoded via the export-decoded endpoint in an emergency.
- **allowed_roles** (*list*) – aAlist of allowed roles that this transformation can be assigned to. A role using this transformation must exist in this list in order for encode and decode operations to properly function.

- **stores** (*list*) – list of tokenization stores to use for tokenization state. Vault's internal storage is used by default.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the create_or_update_tokenization_transformation request.

**Return type** requests.Response

**create_or_update_transformation**(*name*, *transform_type*, *template*, *tweak_source='supplied'*, *masking_character='*'*, *allowed_roles=None*, *mount_point='transform'*)

Create or update a transformation with the given name.

If a transformation with the name does not exist, it will be created. If the transformation exists, it will be updated with the new attributes.

**Supported methods:** POST: /{mount_point}/transformation/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the transformation to create or update. This is part of the request URL.
- **transform_type** (*str | unicode*) – Specifies the type of transformation to perform. The types currently supported by this backend are fpe and masking. This value cannot be modified by an update operation after creation.
- **template** (*str | unicode*) – the template name to use for matching value on encode and decode operations when using this transformation.
- **tweak_source** (*str | unicode*) – Only used when the type is FPE.
- **masking_character** (*str | unicode*) – the character to use for masking. If multiple characters are provided, only the first one is used and the rest is ignored. Only used when the type is masking.
- **allowed_roles** (*list*) – a list of allowed roles that this transformation can be assigned to. A role using this transformation must exist in this list in order for encode and decode operations to properly function.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the create_or_update_ation request.

**Return type** requests.Response

**decode**(*role_name*, *value=None*, *transformation=None*, *tweak=None*, *batch_input=None*, *mount_point='transform'*)

Decode the provided value using a named role.

**Supported methods:** POST: /{mount_point}/decode/:role_name.

**Parameters**

- **role_name** (*str | unicode*) – the role name to use for this operation. This is specified as part of the URL.
- **value** (*str | unicode*) – the value to be decoded.

- **transformation** (*str | unicode*) – the transformation within the role that should be used for this decode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.
- **tweak** (*str | unicode*) – the tweak source.
- **batch_input** (*array<object>*) – a list of items to be decoded in a single batch. When this parameter is set, the 'value', 'transformation' and 'tweak' parameters are ignored. Instead, the aforementioned parameters should be provided within each object in the list.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the decode request.

**Return type** requests.Response

**delete_alphabet**(*name*, *mount_point='transform'*)

Delete an existing alphabet by the given name.

**Supported methods:** DELETE: /{mount_point}/alphabet/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the alphabet to delete. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the delete_alphabet request.

**Return type** requests.Response

**delete_role**(*name*, *mount_point='transform'*)

Delete an existing role by the given name.

**Supported methods:** DELETE: /{mount_point}/role/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the role to delete. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the delete_role request.

**Return type** requests.Response

**delete_template**(*name*, *mount_point='transform'*)

Delete an existing template by the given name.

**Supported methods:** DELETE: /{mount_point}/template/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the template to delete. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the delete_template request.

**Return type** requests.Response

**delete_transformation**(*name*, *mount_point='transform'*)
Delete an existing transformation by the given name.

**Supported methods:** DELETE: /{mount_point}/transformation/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the transformation to delete. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the delete_ation request.

**Return type** requests.Response

**encode**(*role_name*, *value=None*, *transformation=None*, *tweak=None*, *batch_input=None*, *mount_point='transform'*)
Encode the provided value using a named role.

**Supported methods:** POST: /{mount_point}/encode/:role_name.

**Parameters**

- **role_name** (*str | unicode*) – the role name to use for this operation. This is specified as part of the URL.
- **value** (*str | unicode*) – the value to be encoded.
- **transformation** (*str | unicode*) – the transformation within the role that should be used for this encode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.
- **tweak** (*str | unicode*) – the tweak source.
- **batch_input** (*list*) – a list of items to be encoded in a single batch. When this parameter is set, the 'value', 'transformation' and 'tweak' parameters are ignored. Instead, the aforementioned parameters should be provided within each object in the list.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the encode request.

**Return type** requests.Response

**export_decoded_tokenization_state**(*name*, *limit=1000*, *continuation=''*, *mount_point='transform'*)
Start or continue retrieving an export of tokenization state, including the tokens and their decoded values. This call is only supported on tokenization stores configured with the exportable mapping mode. Refer to the Tokenization documentation for when to use the exportable mapping mode. Decoded values are in Base64 representation.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/export-decoded/:name.

**Parameters**

- **name** (*str*) – the name of the transformation to export.
- **limit** (*int*) – maximum number of tokenized value states to return on this call.

- **continuation** (*str*) – absent or empty, a new export is started. If present, the export should continue at the next available value.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the export_decoded_tokenization_state request.

**Return type** requests.Response

**list_alphabets**(*mount_point='transform'*)
List all existing alphabets in the secrets engine.

**Supported methods:** LIST: /{mount_point}/alphabet.

**Parameters** **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the list_alphabets request.

**Return type** requests.Response

**list_roles**(*mount_point='transform'*)
List all existing roles in the secrets engine.

**Supported methods:** LIST: /{mount_point}/role.

**Parameters** **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the list_roles request.

**Return type** requests.Response

**list_templates**(*mount_point='transform'*)
List all existing templates in the secrets engine.

**Supported methods:** LIST: /{mount_point}/transformation.

**Parameters** **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the list_template request.

**Return type** requests.Response

**list_tokenization_key_configuration**(*mount_point='transform'*)
List all tokenization keys. Only valid for tokenization transformations.

**Supported methods:** LIST: /{mount_point}/tokenization/keys/.

**Parameters** **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the list_tokenization_key_configuration request.

**Return type** requests.Response

**list_transformations**(*mount_point='transform'*)
List all existing transformations in the secrets engine.

**Supported methods:** LIST: /{mount_point}/transformation.

**Parameters** **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the list_ation request.

**Return type** requests.Response

**read_alphabet**(*name*, *mount_point='transform'*)
Queries an existing alphabet by the given name.

**Supported methods:** GET: /{mount_point}/alphabet/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the alphabet to delete. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the read_alphabet request.

**Return type** requests.Response

**read_role**(*name*, *mount_point='transform'*)
Query an existing role by the given name.

**Supported methods:** GET: /{mount_point}/role/:name.

**Parameters**

- **name** (*str | unicode*) – the name of the role to read. This is part of the request URL.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the read_role request.

**Return type** requests.Response

**read_template**(*name*, *mount_point='transform'*)
Query an existing template by the given name.

**Supported methods:** GET: /{mount_point}/template/:name.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to read.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the read_template request.

**Return type** requests.Response

**read_tokenization_key_configuration**(*transform_name*, *mount_point='transform'*)
Read tokenization key configuration for a particular transform. Only valid for tokenization transformations.

**Supported methods:** GET: /{mount_point}/tokenization/keys/:{mount_point}_name.

**Parameters**

- **transform_name** (*str*) – the transform name to use for this operation. This is specified as part of the URL.

- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the read_tokenization_key_configuration request.

**Return type** requests.Response

**read_transformation**(*name*, *mount_point='transform'*)

Query an existing transformation by the given name.

**Supported methods:** GET: /{mount_point}/transformation/:name.

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the role to read.
- **mount_point** (*str | unicode*) – The "path" the secrets engine was mounted on.

**Returns** The response of the read_ation request.

**Return type** requests.Response

**restore_tokenization_state**(*name*, *values*, *mount_point='transform'*)

This endpoint restores previously snapshotted tokenization state values to the underlying store(s) of a tokenization transform. Calls to this endpoint are idempotent, so multiple outputs from a snapshot run can be applied via restore in any order and duplicates will not cause a problem.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/restore/:name.

**Parameters**

- **name** (*str*) – the name of the transformation to restore.
- **values** (*str*) – number of tokenization state values from a previous snapshot call.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the restore_tokenization_state request.

**Return type** requests.Response

**retrieve_token_metadata**(*role_name*, *value*, *transformation*, *batch_input=None*, *mount_point='transform'*)

This endpoint retrieves metadata for a tokenized value using a named role. Only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/metadata/:role_name.

**Parameters**

- **role_name** (*str*) – the role name to use for this operation. This is specified as part of the URL.
- **value** (*str*) – the token for which to retrieve metadata.
- **transformation** (*str*) – the transformation within the role that should be used for this decode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.
- **batch_input** (*list*) – a list of items to be decoded in a single batch. When this parameter is set, the 'value' parameter is ignored. Instead, the aforementioned parameters should be provided within each object in the list.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the retrieve_token_metadata request.

**Return type** requests.Response

**rotate_tokenization_key**(*transform_name*, *mount_point='transform'*)

Rotate the version of the named key. After rotation, new requests will be encoded with the new version of the key.

**Supported methods:** POST: /{mount_point}/tokenization/keys/{transform_name}/rotate.

**Parameters**

- **transform_name** (*str*) – the transform name to use for this operation. This is specified as part of the URL.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the rotate_tokenization_key request.

**Return type** requests.Response

**snapshot_tokenization_state**(*name*, *limit=1000*, *continuation=''*, *mount_point='transform'*)

This endpoint starts or continues retrieving a snapshot of the stored state of a tokenization transform. This state is protected as it is in the underlying store, and so is safe for storage or transport. Snapshots may be used for backup purposes or to migrate from one store to another. If more than one store is configured for a tokenization transform, the snapshot data contains the contents of the first store.

**Supported methods:** POST: /{mount_point}/transformations/tokenization/snapshot/:name.

**Parameters**

- **name** (*str*) – the name of the transformation to snapshot.
- **limit** (*int*) – maximum number of tokenized value states to return on this call.
- **continuation** (*str*) – absent or empty, a new snapshot is started. If present, the snapshot should continue at the next available value.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the snapshot_tokenization_state request.

**Return type** requests.Response

**trim_tokenization_key_version**(*transform_name*, *min_available_version*, *mount_point='transform'*)

Trim older key versions setting a minimum version for the keyring. Once trimmed, previous versions of the key cannot be recovered.

**Supported methods:** POST: /{mount_point}/tokenization/keys/{transform_name}/trim.

**Parameters**

- **transform_name** (*str*) – the transform name to use for this operation. This is specified as part of the URL.
- **min_available_version** (*int*) –
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the trim_tokenization_key_version request.

**Return type** requests.Response

**update_tokenization_key_config**(*transform_name*, *min_decryption_version*, *mount_point='transform'*)

Allow the minimum key version to be set for decode operations. Only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/tokenization/keys/{transform_name}/config.

**Parameters**

- **transform_name** (*str*) – the transform name to use for this operation. This is specified as part of the URL.
- **min_decryption_version** (*int*) – the minimum key version that vault can use to decode values for the corresponding transform.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the update_tokenization_key_config request.

**Return type** requests.Response

**validate_token**(*role_name*, *value*, *transformation*, *batch_input=None*, *mount_point='transform'*)

Determine if a provided tokenized value is valid and unexpired. Only valid for tokenization transformations.

**Supported methods:** POST: /{mount_point}/validate/:role_name.

**Parameters**

- **role_name** (*str*) – the role name to use for this operation. This is specified as part of the URL.
- **value** (*str*) – the token for which to check validity.
- **transformation** (*str*) – the transformation within the role that should be used for this decode operation. If a single transformation exists for role, this parameter may be skipped and will be inferred. If multiple transformations exist, one must be specified.
- **batch_input** (*list*) – a list of items to be decoded in a single batch. When this parameter is set, the 'value' parameter is ignored. Instead, the aforementioned parameters should be provided within each object in the list.
- **mount_point** (*str*) – The "path" the method/backend was mounted on.

**Returns** The response of the validate_token request.

**Return type** requests.Response

**class** hvac.api.secrets_engines.**Transit**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Transit Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/transit/index.html **Methods**

| | |
|---|---|
| *backup_key*(name[, mount_point]) | Return a plaintext backup of a named key. |
| *create_key*(name[, convergent_encryption, ...]) | Create a new named encryption key of the specified type. |
| *decrypt_data*(name, ciphertext[, context, ...]) | Decrypt the provided ciphertext using the named key. |
| *delete_key*(name[, mount_point]) | Delete a named encryption key. |
| *encrypt_data*(name, plaintext[, context, ...]) | Encrypt the provided plaintext using the named key. |

continues on next page

Table 46 – continued from previous page

| | |
|---|---|
| export_key(name, key_type[, version, ...]) | Return the named key. |
| generate_data_key(name, key_type[, context, ...]) | Generates a new high-entropy key and the value encrypted with the named key. |
| generate_hmac(name, hash_input[, ...]) | Return the digest of given data using the specified hash algorithm and the named key. |
| generate_random_bytes([n_bytes, ...]) | Return high-quality random bytes of the specified length. |
| hash_data(hash_input[, algorithm, ...]) | Return the cryptographic hash of given data using the specified algorithm. |
| list_keys([mount_point]) | List keys (if there are any). |
| read_key(name[, mount_point]) | Read information about a named encryption key. |
| restore_key(backup[, name, force, mount_point]) | Restore the backup as a named key. |
| rewrap_data(name, ciphertext[, context, ...]) | Rewrap the provided ciphertext using the latest version of the named key. |
| rotate_key(name[, mount_point]) | Rotate the version of the named key. |
| sign_data(name, hash_input[, key_version, ...]) | Return the cryptographic signature of the given data using the named key and the specified hash algorithm. |
| trim_key(name, min_version[, mount_point]) | Trims older key versions setting a minimum version for the keyring. |
| update_key_configuration(name[, ...]) | Tune configuration values for a given key. |
| verify_signed_data(name, hash_input[, ...]) | Return whether the provided signature is valid for the given data. |

**backup_key**(*name*, *mount_point='transit'*)

Return a plaintext backup of a named key.

The backup contains all the configuration data and keys of all the versions along with the HMAC key. The response from this endpoint can be used with the /restore endpoint to restore the key.

**Supported methods:** GET: /{mount_point}/backup/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Name of the key.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**create_key**(*name*, *convergent_encryption=None*, *derived=None*, *exportable=None*, *allow_plaintext_backup=None*, *key_type=None*, *mount_point='transit'*)

Create a new named encryption key of the specified type.

The values set here cannot be changed after key creation.

**Supported methods:** POST: /{mount_point}/keys/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to create. This is specified as part of the URL.

- **convergent_encryption** (*bool*) – If enabled, the key will support convergent encryption, where the same plaintext creates the same ciphertext. This requires derived to be set to true. When enabled, each encryption(/decryption/rewrap/datakey) operation will derive a nonce value rather than randomly generate it.
- **derived** (*bool*) – Specifies if key derivation is to be used. If enabled, all encrypt/decrypt requests to this named key must provide a context which is used for key derivation.
- **exportable** (*bool*) – Enables keys to be exportable. This allows for all the valid keys in the key ring to be exported. Once set, this cannot be disabled.
- **allow_plaintext_backup** (*bool*) – If set, enables taking backup of named key in the plaintext format. Once set, this cannot be disabled.
- **key_type** (*str | unicode*) – Specifies the type of key to create. The currently-supported types are:
  - **aes256-gcm96**: AES-256 wrapped with GCM using a 96-bit nonce size AEAD
  - **chacha20-poly1305**: ChaCha20-Poly1305 AEAD (symmetric, supports derivation and convergent encryption)
  - **ed25519**: ED25519 (asymmetric, supports derivation).
  - **ecdsa-p256**: ECDSA using the P-256 elliptic curve (asymmetric)
  - **ecdsa-p384**: ECDSA using the P-384 elliptic curve (asymmetric)
  - **ecdsa-p521**: ECDSA using the P-521 elliptic curve (asymmetric)
  - **rsa-2048**: RSA with bit size of 2048 (asymmetric)
  - **rsa-3072**: RSA with bit size of 3072 (asymmetric)
  - **rsa-4096**: RSA with bit size of 4096 (asymmetric)
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**decrypt_data**(*name*, *ciphertext*, *context=None*, *nonce=None*, *batch_input=None*, *mount_point='transit'*)

Decrypt the provided ciphertext using the named key.

**Supported methods:** POST: /{mount_point}/decrypt/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to decrypt against. This is specified as part of the URL.
- **ciphertext** (*str | unicode*) – the ciphertext to decrypt.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled.
- **nonce** (*str | unicode*) – Specifies a base64 encoded nonce value used during encryption. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+.

- **batch_input** (*List[dict]*) – Specifies a list of items to be decrypted in a single batch. When this parameter is set, if the parameters 'ciphertext', 'context' and 'nonce' are also set, they will be ignored. Format for the input goes like this: [dict(context="b64_context", ciphertext="b64_plaintext"), ...]
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**delete_key**(*name*, *mount_point='transit'*)

Delete a named encryption key.

It will no longer be possible to decrypt any data encrypted with the named key. Because this is a potentially catastrophic operation, the deletion_allowed tunable must be set in the key's /config endpoint.

**Supported methods:** DELETE: /{mount_point}/keys/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to delete. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**encrypt_data**(*name*, *plaintext*, *context=None*, *key_version=None*, *nonce=None*, *batch_input=None*, *type=None*, *convergent_encryption=None*, *mount_point='transit'*)

Encrypt the provided plaintext using the named key.

This path supports the create and update policy capabilities as follows: if the user has the create capability for this endpoint in their policies, and the key does not exist, it will be upserted with default values (whether the key requires derivation depends on whether the context parameter is empty or not). If the user only has update capability and the key does not exist, an error will be returned.

**Supported methods:** POST: /{mount_point}/encrypt/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to encrypt against. This is specified as part of the URL.
- **plaintext** (*str | unicode*) – Specifies base64 encoded plaintext to be encoded.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled for this key.
- **key_version** (*int*) – Specifies the version of the key to use for encryption. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **nonce** (*str | unicode*) – Specifies the base64 encoded nonce value. This must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+. The value must be exactly 96 bits (12 bytes) long and the user must ensure that for any given context (and thus, any given encryption key) this nonce value is never reused.

- **batch_input** (*List[dict]*) – Specifies a list of items to be encrypted in a single batch. When this parameter is set, if the parameters 'plaintext', 'context' and 'nonce' are also set, they will be ignored. The format for the input is: [dict(context="b64_context", plaintext="b64_plaintext"), ...]
- **type** (*str | unicode*) – This parameter is required when encryption key is expected to be created. When performing an upsert operation, the type of key to create.
- **convergent_encryption** (*str | unicode*) – This parameter will only be used when a key is expected to be created. Whether to support convergent encryption. This is only supported when using a key with key derivation enabled and will require all requests to carry both a context and 96-bit (12-byte) nonce. The given nonce will be used in place of a randomly generated nonce. As a result, when the same context and nonce are supplied, the same ciphertext is generated. It is very important when using this mode that you ensure that all nonces are unique for a given context. Failing to do so will severely impact the ciphertext's security.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**export_key**(*name*, *key_type*, *version=None*, *mount_point='transit'*)

Return the named key.

The keys object shows the value of the key for each version. If version is specified, the specific version will be returned. If latest is provided as the version, the current key will be provided. Depending on the type of key, different information may be returned. The key must be exportable to support this operation and the version must still be valid.

**Supported methods:** GET: /{mount_point}/export/{key_type}/{name}(/{version}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to read information about. This is specified as part of the URL.
- **key_type** (*str | unicode*) – Specifies the type of the key to export. This is specified as part of the URL. Valid values are: encryption-key signing-key hmac-key
- **version** (*str | unicode*) – Specifies the version of the key to read. If omitted, all versions of the key will be returned. If the version is set to latest, the current key will be returned.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**generate_data_key**(*name*, *key_type*, *context=None*, *nonce=None*, *bits=None*, *mount_point='transit'*)

Generates a new high-entropy key and the value encrypted with the named key.

Optionally return the plaintext of the key as well. Whether plaintext is returned depends on the path; as a result, you can use Vault ACL policies to control whether a user is allowed to retrieve the plaintext value of a key. This is useful if you want an untrusted user or operation to generate keys that are then made available to trusted users.

**Supported methods:** POST: /{mount_point}/datakey/{key_type}/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to use to encrypt the datakey. This is specified as part of the URL.
- **key_type** (*str | unicode*) – Specifies the type of key to generate. If plaintext, the plaintext key will be returned along with the ciphertext. If wrapped, only the ciphertext value will be returned. This is specified as part of the URL.
- **context** (*str | unicode*) – Specifies the key derivation context, provided as a base64-encoded string. This must be provided if derivation is enabled.
- **nonce** (*str | unicode*) – Specifies a nonce value, provided as base64 encoded. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+. The value must be exactly 96 bits (12 bytes) long and the user must ensure that for any given context (and thus, any given encryption key) this nonce value is never reused.
- **bits** (*int*) – Specifies the number of bits in the desired key. Can be 128, 256, or 512.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**generate_hmac**(*name*, *hash_input*, *key_version=None*, *algorithm=None*, *mount_point='transit'*)

Return the digest of given data using the specified hash algorithm and the named key.

The key can be of any type supported by transit; the raw key will be marshaled into bytes to be used for the HMAC function. If the key is of a type that supports rotation, the latest (current) version will be used.

**Supported methods:** POST: /{mount_point}/hmac/{name}(/{algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to generate hmac against. This is specified as part of the URL.
- **hash_input** – Specifies the base64 encoded input data.
- **key_version** (*int*) – Specifies the version of the key to use for the operation. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**generate_random_bytes**(*n_bytes=None*, *output_format=None*, *mount_point='transit'*)

Return high-quality random bytes of the specified length.

**Supported methods:** POST: /{mount_point}/random(/{bytes}). Produces: 200 application/json

**Parameters**

- **n_bytes** (*int*) – Specifies the number of bytes to return. This value can be specified either in the request body, or as a part of the URL.
- **output_format** (*str | unicode*) – Specifies the output encoding. Valid options are hex or base64.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**hash_data**(*hash_input*, *algorithm=None*, *output_format=None*, *mount_point='transit'*)

Return the cryptographic hash of given data using the specified algorithm.

**Supported methods:** POST: /{mount_point}/hash(/{algorithm}). Produces: 200 application/json

**Parameters**

- **hash_input** (*str | unicode*) – Specifies the base64 encoded input data.
- **algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **output_format** (*str | unicode*) – Specifies the output encoding. This can be either hex or base64.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**list_keys**(*mount_point='transit'*)

List keys (if there are any).

Only the key names are returned (not the actual keys themselves).

An exception is thrown if there are no keys.

**Supported methods:** LIST: /{mount_point}/keys. Produces: 200 application/json

**Parameters** **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**read_key**(*name*, *mount_point='transit'*)

Read information about a named encryption key.

The keys object shows the creation time of each key version; the values are not the keys themselves. Depending on the type of key, different information may be returned, e.g. an asymmetric key will return its public key in a standard format for the type.

**Supported methods:** GET: /{mount_point}/keys/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to read. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the read_key request.

**Return type** dict

**restore_key**(*backup*, *name=None*, *force=None*, *mount_point='transit'*)

Restore the backup as a named key.

This will restore the key configurations and all the versions of the named key along with HMAC keys. The input to this endpoint should be the output of /backup endpoint. For safety, by default the backend will refuse to restore to an existing key. If you want to reuse a key name, it is recommended you delete the key before restoring. It is a good idea to attempt restoring to a different key name first to verify that the operation successfully completes.

**Supported methods:** POST: /{mount_point}/restore(/name). Produces: 204 (empty body)

**Parameters**

- **backup** (*str | unicode*) – Backed up key data to be restored. This should be the output from the /backup endpoint.
- **name** (*str | unicode*) – If set, this will be the name of the restored key.
- **force** (*bool*) – If set, force the restore to proceed even if a key by this name already exists.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**rewrap_data**(*name*, *ciphertext*, *context=None*, *key_version=None*, *nonce=None*, *batch_input=None*, *mount_point='transit'*)

Rewrap the provided ciphertext using the latest version of the named key.

Because this never returns plaintext, it is possible to delegate this functionality to untrusted users or scripts.

**Supported methods:** POST: /{mount_point}/rewrap/{name}. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to re-encrypt against. This is specified as part of the URL.
- **ciphertext** (*str | unicode*) – Specifies the ciphertext to re-encrypt.
- **context** (*str | unicode*) – Specifies the base64 encoded context for key derivation. This is required if key derivation is enabled.
- **key_version** (*int*) – Specifies the version of the key to use for the operation. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **nonce** (*str | unicode*) – Specifies a base64 encoded nonce value used during encryption. Must be provided if convergent encryption is enabled for this key and the key was generated with Vault 0.6.1. Not required for keys created in 0.6.2+.

- **batch_input** (*List[dict]*) – Specifies a list of items to be decrypted in a single batch. When this parameter is set, if the parameters 'ciphertext', 'context' and 'nonce' are also set, they will be ignored. Format for the input goes like this: [dict(context="b64_context", ciphertext="b64_plaintext"), ...]
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**rotate_key**(*name*, *mount_point='transit'*)

Rotate the version of the named key.

After rotation, new plaintext requests will be encrypted with the new version of the key. To upgrade ciphertext to be encrypted with the latest version of the key, use the rewrap endpoint. This is only supported with keys that support encryption and decryption operations.

**Supported methods:** POST: /{mount_point}/keys/{name}/rotate. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to read information about. This is specified as part of the URL.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**sign_data**(*name*, *hash_input*, *key_version=None*, *hash_algorithm=None*, *context=None*, *prehashed=None*, *signature_algorithm=None*, *marshaling_algorithm=None*, *mount_point='transit'*)

Return the cryptographic signature of the given data using the named key and the specified hash algorithm.

The key must be of a type that supports signing.

**Supported methods:** POST: /{mount_point}/sign/{name}(/{hash_algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to use for signing. This is specified as part of the URL.
- **hash_input** (*str | unicode*) – Specifies the base64 encoded input data.
- **key_version** (*int*) – Specifies the version of the key to use for signing. If not set, uses the latest version. Must be greater than or equal to the key's min_encryption_version, if set.
- **hash_algorithm** (*str | unicode*) – Specifies the hash algorithm to use for supporting key types (notably, not including ed25519 which specifies its own hash algorithm). This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **context** (*str | unicode*) – Base64 encoded context for key derivation. Required if key derivation is enabled; currently only available with ed25519 keys.
- **prehashed** (*bool*) – Set to true when the input is already hashed. If the key type is rsa-2048 or rsa-4096, then the algorithm used to hash the input should be indicated by the hash_algorithm parameter. Just as the value to sign should be the base64-encoded

representation of the exact binary data you want signed, when set, input is expected to be base64-encoded binary hashed data, not hex-formatted. (As an example, on the command line, you could generate a suitable input via openssl dgst -sha256 -binary | base64.)

- **signature_algorithm** (*str | unicode*) – When using a RSA key, specifies the RSA signature algorithm to use for signing. Supported signature types are: pss, pkcs1v15
- **marshaling_algorithm** (*str | unicode*) – Specifies the way in which the signature should be marshaled. This currently only applies to ECDSA keys. Supported types are: asn1, jws
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

**trim_key**(*name*, *min_version*, *mount_point='transit'*)

Trims older key versions setting a minimum version for the keyring.

Once trimmed, previous versions of the key cannot be recovered.

**Supported methods:** POST: /{mount_point}/keys/{name}/trim. Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the key to be trimmed.
- **min_version** (*int*) – The minimum version for the key ring. All versions before this version will be permanently deleted. This value can at most be equal to the lesser of min_decryption_version and min_encryption_version. This is not allowed to be set when either min_encryption_version or min_decryption_version is set to zero.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** dict

**update_key_configuration**(*name*, *min_decryption_version=None*, *min_encryption_version=None*, *deletion_allowed=None*, *exportable=None*, *allow_plaintext_backup=None*, *mount_point='transit'*)

Tune configuration values for a given key.

These values are returned during a read operation on the named key.

**Supported methods:** POST: /{mount_point}/keys/{name}/config. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key to update configuration for.
- **min_decryption_version** (*int*) – Specifies the minimum version of ciphertext allowed to be decrypted. Adjusting this as part of a key rotation policy can prevent old copies of ciphertext from being decrypted, should they fall into the wrong hands. For signatures, this value controls the minimum version of signature that can be verified against. For HMACs, this controls the minimum version of a key allowed to be used as the key for verification.

- **min_encryption_version** (*int*) – Specifies the minimum version of the key that can be used to encrypt plaintext, sign payloads, or generate HMACs. Must be 0 (which will use the latest version) or a value greater or equal to min_decryption_version.
- **deletion_allowed** (*bool*) – Specifies if the key is allowed to be deleted.
- **exportable** (*bool*) – Enables keys to be exportable. This allows for all the valid keys in the key ring to be exported. Once set, this cannot be disabled.
- **allow_plaintext_backup** (*bool*) – If set, enables taking backup of named key in the plaintext format. Once set, this cannot be disabled.
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The response of the request.

**Return type** requests.Response

**verify_signed_data**(*name*, *hash_input*, *signature=None*, *hmac=None*, *hash_algorithm=None*, *context=None*, *prehashed=None*, *signature_algorithm=None*, *marshaling_algorithm=None*, *mount_point='transit'*)

Return whether the provided signature is valid for the given data.

**Supported methods:** POST: /{mount_point}/verify/{name}(/{hash_algorithm}). Produces: 200 application/json

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the encryption key that was used to generate the signature or HMAC.
- **hash_input** – Specifies the base64 encoded input data.
- **signature** (*str | unicode*) – Specifies the signature output from the /transit/sign function. Either this must be supplied or hmac must be supplied.
- **hmac** (*str | unicode*) – Specifies the signature output from the /transit/hmac function. Either this must be supplied or signature must be supplied.
- **hash_algorithm** (*str | unicode*) – Specifies the hash algorithm to use. This can also be specified as part of the URL. Currently-supported algorithms are: sha2-224, sha2-256, sha2-384, sha2-512
- **context** (*str | unicode*) – Base64 encoded context for key derivation. Required if key derivation is enabled; currently only available with ed25519 keys.
- **prehashed** (*bool*) – Set to true when the input is already hashed. If the key type is rsa-2048 or rsa-4096, then the algorithm used to hash the input should be indicated by the hash_algorithm parameter.
- **signature_algorithm** (*str | unicode*) – When using a RSA key, specifies the RSA signature algorithm to use for signature verification. Supported signature types are: pss, pkcs1v15
- **marshaling_algorithm** (*str | unicode*) – Specifies the way in which the signature should be marshaled. This currently only applies to ECDSA keys. Supported types are: asn1, jws
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

**Returns** The JSON response of the request.

**Return type** dict

## 4.5 hvac.api.system_backend

Collection of Vault system backend API endpoint classes.

**Classes**

| | |
|---|---|
| *Audit*(adapter) | |
| *Auth*(adapter) | |
| *Capabilities*(adapter) | |
| *Health*(adapter) | . |
| *Init*(adapter) | |
| *Key*(adapter) | |
| *Leader*(adapter) | |
| *Lease*(adapter) | |
| *Mount*(adapter) | |
| *Namespace*(adapter) | |
| *Policy*(adapter) | |
| *Raft*(adapter) | Raft cluster-related system backend methods. |
| *Seal*(adapter) | |
| *SystemBackend*(adapter) | |
| *SystemBackendMixin*(adapter) | Base class for System Backend API endpoints. |
| *Wrapping*(adapter) | |

**class** hvac.api.system_backend.**Audit**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *calculate_hash*(path, input_to_hash) | Hash the given input data with the specified audit device's hash function and salt. |
| *disable_audit_device*(path) | Disable the audit device at the given path. |
| *enable_audit_device*(device_type[, ...]) | Enable a new audit device at the supplied path. |
| *list_enabled_audit_devices*() | List enabled audit devices. |

**calculate_hash**(*path*, *input_to_hash*)

Hash the given input data with the specified audit device's hash function and salt.

This endpoint can be used to discover whether a given plaintext string (the input parameter) appears in the audit log in obfuscated form.

**Supported methods:** POST: /sys/audit-hash/{path}. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – The path of the audit device to generate hashes for. This is part of the request URL.
- **input_to_hash** (*str | unicode*) – The input string to hash.

**Returns** The JSON response of the request.

**Return type** requests.Response

**disable_audit_device**(*path*)

Disable the audit device at the given path.

**Supported methods:** DELETE: /sys/audit/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – The path of the audit device to delete. This is part of the request URL.

**Returns** The response of the request.

**Return type** requests.Response

**enable_audit_device**(*device_type*, *description=None*, *options=None*, *path=None*, *local=None*)
Enable a new audit device at the supplied path.

The path can be a single word name or a more complex, nested path.

**Supported methods:** PUT: /sys/audit/{path}. Produces: 204 (empty body)

**Parameters**

- **device_type** (*str | unicode*) – Specifies the type of the audit device.
- **description** (*str | unicode*) – Human-friendly description of the audit device.
- **options** (*str | unicode*) – Configuration options to pass to the audit device itself. This is dependent on the audit device type.
- **path** (*str | unicode*) – Specifies the path in which to enable the audit device. This is part of the request URL.
- **local** (*bool*) – Specifies if the audit device is a local only.

**Returns** The response of the request.

**Return type** requests.Response

**list_enabled_audit_devices**()
List enabled audit devices.

It does not list all available audit devices. This endpoint requires sudo capability in addition to any path-specific capabilities.

**Supported methods:** GET: /sys/audit. Produces: 200 application/json

**Returns** JSON response of the request.

**Return type** dict

**class** hvac.api.system_backend.**Auth**(*adapter*)
Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *disable_auth_method*(path) | Disable the auth method at the given auth path. |
| *enable_auth_method*(method_type[, ...]) | Enable a new auth method. |
| *list_auth_methods*() | List all enabled auth methods. |
| *read_auth_method_tuning*(path) | Read the given auth path's configuration. |
| *tune_auth_method*(path[, default_lease_ttl, ...]) | Tune configuration parameters for a given auth path. |

**disable_auth_method**(*path*)
Disable the auth method at the given auth path.

**Supported methods:** DELETE: /sys/auth/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.

**Returns** The response of the request.

**Return type** requests.Response

**enable_auth_method**(*method_type*, *description=None*, *config=None*, *plugin_name=None*, *local=False*, *path=None*, ***kwargs*)

Enable a new auth method.

After enabling, the auth method can be accessed and configured via the auth path specified as part of the URL. This auth path will be nested under the auth prefix.

**Supported methods:** POST: /sys/auth/{path}. Produces: 204 (empty body)

**Parameters**

- **method_type** (*str | unicode*) – The name of the authentication method type, such as "github" or "token".
- **description** (*str | unicode*) – A human-friendly description of the auth method.
- **config** (*dict*) – Configuration options for this auth method. These are the possible values:
  - **default_lease_ttl**: The default lease duration, specified as a string duration like "5s" or "30m".
  - **max_lease_ttl**: The maximum lease duration, specified as a string duration like "5s" or "30m".
  - **audit_non_hmac_request_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
  - **audit_non_hmac_response_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
  - **listing_visibility**: Speficies whether to show this mount in the UI-specific listing endpoint.
  - **passthrough_request_headers**: Comma-separated list of headers to whitelist and pass from the request to the backend.
- **plugin_name** (*str | unicode*) – The name of the auth plugin to use based from the name in the plugin catalog. Applies only to plugin methods.
- **local** (*bool*) – <Vault enterprise only> Specifies if the auth method is a local only. Local auth methods are not replicated nor (if a secondary) removed by replication.
- **path** (*str | unicode*) – The path to mount the method on. If not provided, defaults to the value of the "method_type" argument.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

**list_auth_methods**()

List all enabled auth methods.

**Supported methods:** GET: /sys/auth. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**read_auth_method_tuning**(*path*)
Read the given auth path's configuration.

This endpoint requires sudo capability on the final path, but the same functionality can be achieved without sudo via sys/mounts/auth/[auth-path]/tune.

**Supported methods:** GET: /sys/auth/{path}/tune. Produces: 200 application/json

**Parameters** **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.

**Returns** The JSON response of the request.

**Return type** dict

**tune_auth_method**(*path*, *default_lease_ttl=None*, *max_lease_ttl=None*, *description=None*, *audit_non_hmac_request_keys=None*, *audit_non_hmac_response_keys=None*, *listing_visibility=None*, *passthrough_request_headers=None*, ***kwargs*)
Tune configuration parameters for a given auth path.

This endpoint requires sudo capability on the final path, but the same functionality can be achieved without sudo via sys/mounts/auth/[auth-path]/tune.

**Supported methods:** POST: /sys/auth/{path}/tune. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – The path the method was mounted on. If not provided, defaults to the value of the "method_type" argument.
- **default_lease_ttl** (*int*) – Specifies the default time-to-live. If set on a specific auth path, this overrides the global default.
- **max_lease_ttl** (*int*) – The maximum time-to-live. If set on a specific auth path, this overrides the global default.
- **description** (*str | unicode*) – Specifies the description of the mount. This overrides the current stored value, if any.
- **audit_non_hmac_request_keys** (*array*) – Specifies the list of keys that will not be HMAC'd by audit devices in the request data object.
- **audit_non_hmac_response_keys** (*list*) – Specifies the list of keys that will not be HMAC'd by audit devices in the response data object.
- **listing_visibility** (*list*) – Specifies whether to show this mount in the UI-specific listing endpoint. Valid values are "unauth" or "".
- **passthrough_request_headers** (*list*) – List of headers to whitelist and pass from the request to the backend.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

**class** hvac.api.system_backend.**Capabilities**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *get_capabilities*(paths[, token, accessor]) | Get the capabilities associated with a token. |

**get_capabilities**(*paths*, *token=None*, *accessor=None*)

Get the capabilities associated with a token.

**Supported methods:** POST: /sys/capabilities-self. Produces: 200 application/json POST: /sys/capabilities. Produces: 200 application/json POST: /sys/capabilities-accessor. Produces: 200 application/json

**Parameters**

- **paths** (*List[str]*) – Paths on which capabilities are being queried.
- **token** (*str*) – Token for which capabilities are being queried.
- **accessor** (*str*) – Accessor of the token for which capabilities are being queried.

**Returns** The JSON response of the request.

**Return type** dict

**class** hvac.api.system_backend.**Health**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin

**Methods**

| | |
|---|---|
| *read_health_status*([standby_ok, ...]) | Read the health status of Vault. |

.

Reference: https://www.vaultproject.io/api-docs/system/health

**read_health_status**(*standby_ok=None*, *active_code=None*, *standby_code=None*, *dr_secondary_code=None*, *performance_standby_code=None*, *sealed_code=None*, *uninit_code=None*, *method='HEAD'*)

Read the health status of Vault.

This matches the semantics of a Consul HTTP health check and provides a simple way to monitor the health of a Vault instance.

**Parameters**

- **standby_ok** (*bool*) – Specifies if being a standby should still return the active status code instead of the standby status code. This is useful when Vault is behind a non-configurable load balance that just wants a 200-level response.
- **active_code** (*int*) – The status code that should be returned for an active node.
- **standby_code** (*int*) – Specifies the status code that should be returned for a standby node.
- **dr_secondary_code** (*int*) – Specifies the status code that should be returned for a DR secondary node.
- **performance_standby_code** (*int*) – Specifies the status code that should be returned for a performance standby node.

- **sealed_code** (*int*) – Specifies the status code that should be returned for a sealed node.
- **uninit_code** (*int*) – Specifies the status code that should be returned for a uninitialized node.
- **method** (*str | unicode*) – Supported methods: HEAD: /sys/health. Produces: 000 (empty body) GET: /sys/health. Produces: 000 application/json

**Returns** The JSON response of the request.

**Return type** requests.Response

**class** hvac.api.system_backend.**Init**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *initialize*([secret_shares, ...]) | Initialize a new Vault. |
| *is_initialized*() | Determine is Vault is initialized or not. |
| *read_init_status*() | Read the initialization status of Vault. |

**initialize**(*secret_shares=5*, *secret_threshold=3*, *pgp_keys=None*, *root_token_pgp_key=None*, *stored_shares=None*, *recovery_shares=None*, *recovery_threshold=None*, *recovery_pgp_keys=None*)

Initialize a new Vault.

The Vault must not have been previously initialized. The recovery options, as well as the stored shares option, are only available when using Vault HSM.

**Supported methods:** PUT: /sys/init. Produces: 200 application/json

**Parameters**

- **secret_shares** (*int*) – The number of shares to split the master key into.
- **secret_threshold** (*int*) – Specifies the number of shares required to reconstruct the master key. This must be less than or equal secret_shares. If using Vault HSM with auto-unsealing, this value must be the same as secret_shares.
- **pgp_keys** (*list*) – List of PGP public keys used to encrypt the output unseal keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as secret_shares.
- **root_token_pgp_key** (*str | unicode*) – Specifies a PGP public key used to encrypt the initial root token. The key must be base64-encoded from its original binary representation.
- **stored_shares** (*int*) – <enterprise only> Specifies the number of shares that should be encrypted by the HSM and stored for auto-unsealing. Currently must be the same as secret_shares.
- **recovery_shares** (*int*) – <enterprise only> Specifies the number of shares to split the recovery key into.
- **recovery_threshold** (*int*) – <enterprise only> Specifies the number of shares required to reconstruct the recovery key. This must be less than or equal to recovery_shares.
- **recovery_pgp_keys** (*list*) – <enterprise only> Specifies an array of PGP public keys used to encrypt the output recovery keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as recovery_shares.

**Returns** The JSON response of the request.

**Return type** dict

**is_initialized**()
Determine is Vault is initialized or not.

**Returns** True if Vault is initialized, False otherwise.

**Return type** bool

**read_init_status**()
Read the initialization status of Vault.

**Supported methods:** GET: /sys/init. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**class** hvac.api.system_backend.**Key**(*adapter*)
Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *cancel_rekey*([recovery_key]) | Cancel any in-progress rekey. |
| *cancel_rekey_verify*() | Cancel any in-progress rekey verification. |
| *cancel_root_generation*() | Cancel any in-progress root generation attempt. |
| *generate_root*(key, nonce) | Enter a single master key share to progress the root generation attempt. |
| *get_encryption_key_status*() | Read information about the current encryption key used by Vault. |
| *read_backup_keys*([recovery_key]) | Retrieve the backup copy of PGP-encrypted unseal keys. |
| *read_rekey_progress*([recovery_key]) | Read the configuration and progress of the current rekey attempt. |
| *read_rekey_verify_progress*() | Read the configuration and progress of the current rekey verify attempt. |
| *read_root_generation_progress*() | Read the configuration and process of the current root generation attempt. |
| *rekey*(key[, nonce, recovery_key]) | Enter a single recovery key share to progress the rekey of the Vault. |
| *rekey_multi*(keys[, nonce, recovery_key]) | Enter multiple recovery key shares to progress the rekey of the Vault. |
| *rekey_verify*(key, nonce) | Enter a single new recovery key share to progress the rekey verification of the Vault. |
| *rekey_verify_multi*(keys, nonce) | Enter multiple new recovery key shares to progress the rekey verification of the Vault. |
| *rotate_encryption_key*() | Trigger a rotation of the backend encryption key. |
| *start_rekey*([secret_shares, ...]) | Initializes a new rekey attempt. |
| *start_root_token_generation*([otp, pgp_key]) | Initialize a new root generation attempt. |

**cancel_rekey**(*recovery_key=False*)
Cancel any in-progress rekey.

This clears the rekey settings as well as any progress made. This must be called to change the parameters

of the rekey.

Note: Verification is still a part of a rekey. If rekeying is canceled during the verification flow, the current unseal keys remain valid.

**Supported methods:** DELETE: /sys/rekey/init. Produces: 204 (empty body) DELETE: /sys/rekey-recovery-key/init. Produces: 204 (empty body)

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The response of the request.

**Return type** requests.Response

**cancel_rekey_verify**()

Cancel any in-progress rekey verification. This clears any progress made and resets the nonce. Unlike cancel_rekey, this only resets the current verification operation, not the entire rekey atttempt. The return value is the same as GET along with the new nonce.

**Supported methods:** DELETE: /sys/rekey/verify. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

**cancel_root_generation**()

Cancel any in-progress root generation attempt.

This clears any progress made. This must be called to change the OTP or PGP key being used.

**Supported methods:** DELETE: /sys/generate-root/attempt. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** request.Response

**generate_root**(*key*, *nonce*)

Enter a single master key share to progress the root generation attempt.

If the threshold number of master key shares is reached, Vault will complete the root generation and issue the new token. Otherwise, this API must be called multiple times until that threshold is met. The attempt nonce must be provided with each call.

**Supported methods:** PUT: /sys/generate-root/update. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single master key share.
- **nonce** (*str | unicode*) – The nonce of the attempt.

**Returns** The JSON response of the request.

**Return type** dict

**get_encryption_key_status**()

Read information about the current encryption key used by Vault.

**Supported methods:** GET: /sys/key-status. Produces: 200 application/json

**Returns** JSON response with information regarding the current encryption key used by Vault.

**Return type** dict

**read_backup_keys**(*recovery_key=False*)
Retrieve the backup copy of PGP-encrypted unseal keys.

The returned value is the nonce of the rekey operation and a map of PGP key fingerprint to hex-encoded PGP-encrypted key.

**Supported methods:** PUT: /sys/rekey/backup. Produces: 200 application/json PUT: /sys/rekey-recovery-key/backup. Produces: 200 application/json

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** dict

**read_rekey_progress**(*recovery_key=False*)
Read the configuration and progress of the current rekey attempt.

**Supported methods:** GET: /sys/rekey-recovery-key/init. Produces: 200 application/json GET: /sys/rekey/init. Produces: 200 application/json

**Parameters** **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** requests.Response

**read_rekey_verify_progress**()
Read the configuration and progress of the current rekey verify attempt.

**Supported methods:** GET: /sys/rekey/verify. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** requests.Response

**read_root_generation_progress**()
Read the configuration and process of the current root generation attempt.

**Supported methods:** GET: /sys/generate-root/attempt. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**rekey**(*key*, *nonce=None*, *recovery_key=False*)
Enter a single recovery key share to progress the rekey of the Vault.

If the threshold number of recovery key shares is reached, Vault will complete the rekey. Otherwise, this API must be called multiple times until that threshold is met. The rekey nonce operation must be provided with each call.

**Supported methods:** PUT: /sys/rekey/update. Produces: 200 application/json PUT: /sys/rekey-recovery-key/update. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single recovery share key.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey operation.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON response of the request.

**Return type** dict

**rekey_multi**(*keys*, *nonce=None*, *recovery_key=False*)

Enter multiple recovery key shares to progress the rekey of the Vault.

If the threshold number of recovery key shares is reached, Vault will complete the rekey.

**Parameters**

- **keys** (*list*) – Specifies multiple recovery share keys.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey operation.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The last response of the rekey request.

**Return type** response.Request

**rekey_verify**(*key*, *nonce*)

Enter a single new recovery key share to progress the rekey verification of the Vault. If the threshold number of new recovery key shares is reached, Vault will complete the rekey. Otherwise, this API must be called multiple times until that threshold is met. The rekey verification nonce must be provided with each call.

**Supported methods:** PUT: /sys/rekey/verify. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies multiple recovery share keys.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey verify operation.

**Returns** The JSON response of the request.

**Return type** dict

**rekey_verify_multi**(*keys*, *nonce*)

Enter multiple new recovery key shares to progress the rekey verification of the Vault. If the threshold number of new recovery key shares is reached, Vault will complete the rekey. Otherwise, this API must be called multiple times until that threshold is met. The rekey verification nonce must be provided with each call.

**Supported methods:** PUT: /sys/rekey/verify. Produces: 200 application/json

**Parameters**

- **keys** (*list*) – Specifies multiple recovery share keys.
- **nonce** (*str | unicode*) – Specifies the nonce of the rekey verify operation.

**Returns** The JSON response of the request.

**Return type** dict

**rotate_encryption_key**()
Trigger a rotation of the backend encryption key.

This is the key that is used to encrypt data written to the storage backend, and is not provided to operators. This operation is done online. Future values are encrypted with the new key, while old values are decrypted with previous encryption keys.

This path requires sudo capability in addition to update.

**Supported methods:** PUT: /sys/rorate. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

**start_rekey**(*secret_shares=5*, *secret_threshold=3*, *pgp_keys=None*, *backup=False*, *require_verification=False*, *recovery_key=False*)
Initializes a new rekey attempt.

Only a single recovery key rekeyattempt can take place at a time, and changing the parameters of a rekey requires canceling and starting a new rekey, which will also provide a new nonce.

**Supported methods:** PUT: /sys/rekey/init. Produces: 204 (empty body) PUT: /sys/rekey-recovery-key/init. Produces: 204 (empty body)

**Parameters**

- **secret_shares** (*int*) – Specifies the number of shares to split the master key into.
- **secret_threshold** (*int*) – Specifies the number of shares required to reconstruct the master key. This must be less than or equal to secret_shares.
- **pgp_keys** (*list*) – Specifies an array of PGP public keys used to encrypt the output unseal keys. Ordering is preserved. The keys must be base64-encoded from their original binary representation. The size of this array must be the same as secret_shares.
- **backup** (*bool*) – Specifies if using PGP-encrypted keys, whether Vault should also store a plaintext backup of the PGP-encrypted keys at core/unseal-keys-backup in the physical storage backend. These can then be retrieved and removed via the sys/rekey/backup endpoint.
- **require_verification** (*bool*) – This turns on verification functionality. When verification is turned on, after successful authorization with the current unseal keys, the new unseal keys are returned but the master key is not actually rotated. The new keys must be provided to authorize the actual rotation of the master key. This ensures that the new keys have been successfully saved and protects against a risk of the keys being lost after rotation but before they can be persisted. This can be used with without pgp_keys, and when used with it, it allows ensuring that the returned keys can be successfully decrypted before committing to the new shares, which the backup functionality does not provide.
- **recovery_key** (*bool*) – If true, send requests to "rekey-recovery-key" instead of "rekey" api path.

**Returns** The JSON dict of the response.

**Return type** dict | request.Response

**start_root_token_generation**(*otp=None*, *pgp_key=None*)
Initialize a new root generation attempt.

Only a single root generation attempt can take place at a time. One (and only one) of otp or pgp_key are required.

**Supported methods:** PUT: /sys/generate-root/attempt. Produces: 200 application/json

**Parameters**

- **otp** (*str | unicode*) – Specifies a base64-encoded 16-byte value. The raw bytes of the token will be XOR'd with this value before being returned to the final unseal key provider.
- **pgp_key** (*str | unicode*) – Specifies a base64-encoded PGP public key. The raw bytes of the token will be encrypted with this value before being returned to the final unseal key provider.

**Returns** The JSON response of the request.

**Return type** dict

**class** hvac.api.system_backend.**Leader**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *read_leader_status*() | Read the high availability status and current leader instance of Vault. |
| *step_down*() | Force the node to give up active status. |

**read_leader_status**()

Read the high availability status and current leader instance of Vault.

**Supported methods:** GET: /sys/leader. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**step_down**()

Force the node to give up active status.

If the node does not have active status, this endpoint does nothing. Note that the node will sleep for ten seconds before attempting to grab the active lock again, but if no standby nodes grab the active lock in the interim, the same node may become the active node again. Requires a token with root policy or sudo capability on the path.

**Returns** The JSON response of the request.

**Return type** dict

**class** hvac.api.system_backend.**Lease**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *list_leases*(prefix) | Retrieve a list of lease ids. |
| *read_lease*(lease_id) | Retrieve lease metadata. |
| *renew_lease*(lease_id[, increment]) | Renew a lease, requesting to extend the lease. |
| *revoke_force*(prefix) | Revoke all secrets or tokens generated under a given prefix immediately. |
| *revoke_lease*(lease_id) | Revoke a lease immediately. |

continues on next page

Table 55 – continued from previous page

| | |
|---|---|
| revoke_prefix(prefix) | Revoke all secrets (via a lease ID prefix) or tokens (via the tokens' path property) generated under a given prefix immediately. |

**list_leases**(*prefix*)
Retrieve a list of lease ids.

**Supported methods:** LIST: /sys/leases/lookup/{prefix}. Produces: 200 application/json

**Parameters** **prefix** (*str | unicode*) – Lease prefix to filter list by.

**Returns** The JSON response of the request.

**Return type** dict

**read_lease**(*lease_id*)
Retrieve lease metadata.

**Supported methods:** PUT: /sys/leases/lookup. Produces: 200 application/json

**Parameters** **lease_id** (*str | unicode*) – the ID of the lease to lookup.

**Returns** Parsed JSON response from the leases PUT request

**Return type** dict.

**renew_lease**(*lease_id*, *increment=None*)
Renew a lease, requesting to extend the lease.

**Supported methods:** PUT: /sys/leases/renew. Produces: 200 application/json

**Parameters**

- **lease_id** (*str | unicode*) – The ID of the lease to extend.
- **increment** (*int*) – The requested amount of time (in seconds) to extend the lease.

**Returns** The JSON response of the request

**Return type** dict

**revoke_force**(*prefix*)
Revoke all secrets or tokens generated under a given prefix immediately.

Unlike revoke_prefix, this path ignores backend errors encountered during revocation. This is potentially very dangerous and should only be used in specific emergency situations where errors in the backend or the connected backend service prevent normal revocation.

**Supported methods:** PUT: /sys/leases/revoke-force/{prefix}. Produces: 204 (empty body)

**Parameters** **prefix** (*str | unicode*) – The prefix to revoke.

**Returns** The response of the request.

**Return type** requests.Response

**revoke_lease**(*lease_id*)
Revoke a lease immediately.

**Supported methods:** PUT: /sys/leases/revoke. Produces: 204 (empty body)

**Parameters** **lease_id** (*str | unicode*) – Specifies the ID of the lease to revoke.

**Returns** The response of the request.

**Return type** requests.Response

**revoke_prefix**(*prefix*)

Revoke all secrets (via a lease ID prefix) or tokens (via the tokens' path property) generated under a given prefix immediately.

This requires sudo capability and access to it should be tightly controlled as it can be used to revoke very large numbers of secrets/tokens at once.

**Supported methods:** PUT: /sys/leases/revoke-prefix/{prefix}. Produces: 204 (empty body)

**Parameters** **prefix** (*str | unicode*) – The prefix to revoke.

**Returns** The response of the request.

**Return type** requests.Response

**class** hvac.api.system_backend.**Mount**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *disable_secrets_engine*(path) | Disable the mount point specified by the provided path. |
| *enable_secrets_engine*(backend_type[, path, ...]) | Enable a new secrets engine at the given path. |
| *list_mounted_secrets_engines*() | Lists all the mounted secrets engines. |
| *move_backend*(from_path, to_path) | Move an already-mounted backend to a new mount point. |
| *read_mount_configuration*(path) | Read the given mount's configuration. |
| *retrieve_mount_option*(mount_point, option_name) | |
| *tune_mount_configuration*(path[, ...]) | Tune configuration parameters for a given mount point. |

**disable_secrets_engine**(*path*)

Disable the mount point specified by the provided path.

**Supported methods:** DELETE: /sys/mounts/{path}. Produces: 204 (empty body)

**Parameters** **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.

**Returns** The response of the request.

**Return type** requests.Response

**enable_secrets_engine**(*backend_type*, *path=None*, *description=None*, *config=None*, *plugin_name=None*, *options=None*, *local=False*, *seal_wrap=False*, ***kwargs*)

Enable a new secrets engine at the given path.

**Supported methods:** POST: /sys/mounts/{path}. Produces: 204 (empty body)

**Parameters**

- **backend_type** (*str | unicode*) – The name of the backend type, such as "github" or "token".
- **path** (*str | unicode*) – The path to mount the method on. If not provided, defaults to the value of the "backend_type" argument.
- **description** (*str | unicode*) – A human-friendly description of the mount.
- **config** (*dict*) – Configuration options for this mount. These are the possible values:
  - **default_lease_ttl**: The default lease duration, specified as a string duration like "5s" or "30m".
  - **max_lease_ttl**: The maximum lease duration, specified as a string duration like "5s" or "30m".
  - **force_no_cache**: Disable caching.
  - **plugin_name**: The name of the plugin in the plugin catalog to use.
  - **audit_non_hmac_request_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
  - **audit_non_hmac_response_keys**: Comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
  - **listing_visibility**: Specifies whether to show this mount in the UI-specific listing endpoint. ("unauth" or "hidden")
  - **passthrough_request_headers**: Comma-separated list of headers to whitelist and pass from the request to the backend.
- **options** (*dict*) – Specifies mount type specific options that are passed to the backend.
  - **version**: <KV> The version of the KV to mount. Set to "2" for mount KV v2.
- **plugin_name** (*str | unicode*) – Specifies the name of the plugin to use based from the name in the plugin catalog. Applies only to plugin backends.
- **local** (*bool*) – <Vault enterprise only> Specifies if the auth method is a local only. Local auth methods are not replicated nor (if a secondary) removed by replication.
- **seal_wrap** (*bool*) – <Vault enterprise only> Enable seal wrapping for the mount.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response of the request.

**Return type** requests.Response

**list_mounted_secrets_engines**()

Lists all the mounted secrets engines.

**Supported methods:** POST: /sys/mounts. Produces: 200 application/json

**Returns** JSON response of the request.

**Return type** dict

**move_backend**(*from_path*, *to_path*)

Move an already-mounted backend to a new mount point.

**Supported methods:** POST: /sys/remount. Produces: 204 (empty body)

**Parameters**

- **from_path** (*str | unicode*) – Specifies the previous mount point.
- **to_path** (*str | unicode*) – Specifies the new destination mount point.

**Returns** The response of the request.

**Return type** requests.Response

**read_mount_configuration**(*path*)
Read the given mount's configuration.

Unlike the mounts endpoint, this will return the current time in seconds for each TTL, which may be the system default or a mount-specific value.

**Supported methods:** GET: /sys/mounts/{path}/tune. Produces: 200 application/json

**Parameters** **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.

**Returns** The JSON response of the request.

**Return type** requests.Response

**retrieve_mount_option**(*mount_point*, *option_name*, *default_value=None*)

**tune_mount_configuration**(*path*, *default_lease_ttl=None*, *max_lease_ttl=None*, *description=None*, *audit_non_hmac_request_keys=None*, *audit_non_hmac_response_keys=None*, *listing_visibility=None*, *passthrough_request_headers=None*, *options=None*, *force_no_cache=None*, ***kwargs*)
Tune configuration parameters for a given mount point.

**Supported methods:** POST: /sys/mounts/{path}/tune. Produces: 204 (empty body)

**Parameters**

- **path** (*str | unicode*) – Specifies the path where the secrets engine will be mounted. This is specified as part of the URL.
- **mount_point** (*str*) – The path the associated secret backend is mounted
- **description** (*str*) – Specifies the description of the mount. This overrides the current stored value, if any.
- **default_lease_ttl** (*int*) – Default time-to-live. This overrides the global default. A value of 0 is equivalent to the system default TTL
- **max_lease_ttl** (*int*) – Maximum time-to-live. This overrides the global default. A value of 0 are equivalent and set to the system max TTL.
- **audit_non_hmac_request_keys** (*list*) – Specifies the comma-separated list of keys that will not be HMAC'd by audit devices in the request data object.
- **audit_non_hmac_response_keys** (*list*) – Specifies the comma-separated list of keys that will not be HMAC'd by audit devices in the response data object.
- **listing_visibility** (*str*) – Speficies whether to show this mount in the UI-specific listing endpoint. Valid values are "unauth" or "".
- **passthrough_request_headers** (*str*) – Comma-separated list of headers to whitelist and pass from the request to the backend.

- **options** (*dict*) – Specifies mount type specific options that are passed to the backend.
  - **version**: <KV> The version of the KV to mount. Set to "2" for mount KV v2.
- **force_no_cache** (*bool*) – Disable caching.
- **kwargs** (*dict*) – All dicts are accepted and passed to vault. See your specific secret engine for details on which extra key-word arguments you might want to pass.

**Returns** The response from the request.

**Return type** request.Response

**class** hvac.api.system_backend.**Namespace**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *create_namespace*(path) | Create a namespace at the given path. |
| *delete_namespace*(path) | Delete a namespaces. |
| *list_namespaces*() | Lists all the namespaces. |

**create_namespace**(*path*)

Create a namespace at the given path.

**Supported methods:** POST: /sys/namespaces/{path}. Produces: 200 application/json

**Returns** The response of the request.

**Return type** requests.Response

**delete_namespace**(*path*)

Delete a namespaces. You cannot delete a namespace with existing child namespaces.

**Supported methods:** DELETE: /sys/namespaces. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

**list_namespaces**()

Lists all the namespaces.

**Supported methods:** LIST: /sys/namespaces. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**class** hvac.api.system_backend.**Policy**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *create_or_update_policy*(name, policy[, ...]) | Add a new or update an existing policy. |
| *delete_policy*(name) | Delete the policy with the given name. |
| *list_policies*() | List all configured policies. |
| *read_policy*(name) | Retrieve the policy body for the named policy. |

**create_or_update_policy**(*name*, *policy*, *pretty_print=True*)
Add a new or update an existing policy.

Once a policy is updated, it takes effect immediately to all associated users.

**Supported methods:** PUT: /sys/policy/{name}. Produces: 204 (empty body)

**Parameters**

- **name** (*str | unicode*) – Specifies the name of the policy to create.
- **policy** (*str | unicode | dict*) – Specifies the policy document.
- **pretty_print** (*bool*) – If True, and provided a dict for the policy argument, send the policy JSON to Vault with "pretty" formatting.

**Returns** The response of the request.

**Return type** requests.Response

**delete_policy**(*name*)
Delete the policy with the given name.

This will immediately affect all users associated with this policy.

**Supported methods:** DELETE: /sys/policy/{name}. Produces: 204 (empty body)

**Parameters** **name** (*str | unicode*) – Specifies the name of the policy to delete.

**Returns** The response of the request.

**Return type** requests.Response

**list_policies**()
List all configured policies.

**Supported methods:** GET: /sys/policy. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**read_policy**(*name*)
Retrieve the policy body for the named policy.

**Supported methods:** GET: /sys/policy/{name}. Produces: 200 application/json

**Parameters** **name** (*str | unicode*) – The name of the policy to retrieve.

**Returns** The response of the request

**Return type** dict

**class** hvac.api.system_backend.**Raft**(*adapter*)
Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin

Raft cluster-related system backend methods.

When using Shamir seal, as soon as the Vault server is brought up, this API should be invoked instead of sys/init. This API completes in 2 phases. Once this is invoked, the joining node will receive a challenge from the Raft's leader node. This challenge can be answered by the joining node only after a successful unseal. Hence, the joining node should be unsealed using the unseal keys of the Raft's leader node.

Reference: https://www.vaultproject.io/api-docs/system/storage/raft **Methods**

| | |
|---|---|
| *force_restore_raft_snapshot*(snapshot) | Installs the provided snapshot, returning the cluster to the state defined in it. |
| *join_raft_cluster*(leader_api_addr[, retry, ...]) | Join a new server node to the Raft cluster. |
| *read_raft_config*() | Read the details of all the nodes in the raft cluster. |
| *remove_raft_node*(server_id) | Remove a node from the raft cluster. |
| *restore_raft_snapshot*(snapshot) | Install the provided snapshot, returning the cluster to the state defined in it. |
| *take_raft_snapshot*() | Returns a snapshot of the current state of the raft cluster. |

**force_restore_raft_snapshot**(*snapshot*)

Installs the provided snapshot, returning the cluster to the state defined in it.

This is same as writing to /sys/storage/raft/snapshot except that this bypasses checks ensuring the Autounseal or shamir keys are consistent with the snapshot data.

**Supported methods:** POST: /sys/storage/raft/snapshot-force.

**Parameters** **snapshot** (*bytes*) – Previously created raft snapshot / binary data.

**Returns** The response of the force_restore_raft_snapshot request.

**Return type** requests.Response

**join_raft_cluster**(*leader_api_addr*, *retry=False*, *leader_ca_cert=None*, *leader_client_cert=None*, *leader_client_key=None*)

Join a new server node to the Raft cluster.

When using Shamir seal, as soon as the Vault server is brought up, this API should be invoked instead of sys/init. This API completes in 2 phases. Once this is invoked, the joining node will receive a challenge from the Raft's leader node. This challenge can be answered by the joining node only after a successful unseal. Hence, the joining node should be unsealed using the unseal keys of the Raft's leader node.

**Supported methods:** POST: /sys/storage/raft/join.

**Parameters**

- **leader_api_addr** (*str | unicode*) – Address of the leader node in the Raft cluster to which this node is trying to join.
- **retry** (*bool*) – Retry joining the Raft cluster in case of failures.
- **leader_ca_cert** (*str | unicode*) – CA certificate used to communicate with Raft's leader node.
- **leader_client_cert** (*str | unicode*) – Client certificate used to communicate with Raft's leader node.
- **leader_client_key** (*str | unicode*) – Client key used to communicate with Raft's leader node.

**Returns** The response of the join_raft_cluster request.

**Return type** requests.Response

**read_raft_config**()

Read the details of all the nodes in the raft cluster.

**Supported methods:** GET: /sys/storage/raft/configuration.

**Returns** The response of the read_raft_config request.

**Return type** requests.Response

**remove_raft_node**(*server_id*)
Remove a node from the raft cluster.

**Supported methods:** POST: /sys/storage/raft/remove-peer.

**Parameters** **server_id** (*str*) – The ID of the node to remove.

**Returns** The response of the remove_raft_node request.

**Return type** requests.Response

**restore_raft_snapshot**(*snapshot*)
Install the provided snapshot, returning the cluster to the state defined in it.

**Supported methods:** POST: /sys/storage/raft/snapshot.

**Parameters** **snapshot** (*bytes*) – Previously created raft snapshot / binary data.

**Returns** The response of the restore_raft_snapshot request.

**Return type** requests.Response

**take_raft_snapshot**()
Returns a snapshot of the current state of the raft cluster.

The snapshot is returned as binary data and should be redirected to a file.

**Supported methods:** GET: /sys/storage/raft/snapshot.

**Returns** The response of the s request.

**Return type** requests.Response

**class** hvac.api.system_backend.**Seal**(*adapter*)
Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *is_sealed*() | Determine if Vault is sealed. |
| *read_seal_status*() | Read the seal status of the Vault. |
| *seal*() | Seal the Vault. |
| *submit_unseal_key*([key, reset, migrate]) | Enter a single master key share to progress the unsealing of the Vault. |
| *submit_unseal_keys*(keys[, migrate]) | Enter multiple master key share to progress the unsealing of the Vault. |

**is_sealed**()
Determine if Vault is sealed.

**Returns** True if Vault is seal, False otherwise.

**Return type** bool

**read_seal_status**()
Read the seal status of the Vault.

This is an unauthenticated endpoint.

**Supported methods:** GET: /sys/seal-status. Produces: 200 application/json

**Returns** The JSON response of the request.

**Return type** dict

**seal**()
Seal the Vault.

In HA mode, only an active node can be sealed. Standby nodes should be restarted to get the same effect. Requires a token with root policy or sudo capability on the path.

**Supported methods:** PUT: /sys/seal. Produces: 204 (empty body)

**Returns** The response of the request.

**Return type** requests.Response

**submit_unseal_key**(*key=None*, *reset=False*, *migrate=False*)
Enter a single master key share to progress the unsealing of the Vault.

If the threshold number of master key shares is reached, Vault will attempt to unseal the Vault. Otherwise, this API must be called multiple times until that threshold is met.

Either the key or reset parameter must be provided; if both are provided, reset takes precedence.

**Supported methods:** PUT: /sys/unseal. Produces: 200 application/json

**Parameters**

- **key** (*str | unicode*) – Specifies a single master key share. This is required unless reset is true.
- **reset** (*bool*) – Specifies if previously-provided unseal keys are discarded and the unseal process is reset.
- **migrate** – Available in 1.0 Beta - Used to migrate the seal from shamir to autoseal or autoseal to shamir. Must be provided on all unseal key calls.

**Type** migrate: bool

**Returns** The JSON response of the request.

**Return type** dict

**submit_unseal_keys**(*keys*, *migrate=False*)
Enter multiple master key share to progress the unsealing of the Vault.

**Parameters**

- **keys** (*List[str]*) – List of master key shares.
- **migrate** – Available in 1.0 Beta - Used to migrate the seal from shamir to autoseal or autoseal to shamir. Must be provided on all unseal key calls.

**Type** migrate: bool

**Returns** The JSON response of the last unseal request.

**Return type** dict

**class** hvac.api.system_backend.**SystemBackend**(*adapter*)

Bases: hvac.api.vault_api_category.VaultApiCategory, hvac.api.system_backend.audit.Audit, hvac.api.system_backend.auth.Auth, hvac.api.system_backend.capabilities.Capabilities, hvac.api.system_backend.health.Health, hvac.api.system_backend.init.Init, hvac.api.system_backend.key.Key, hvac.api.system_backend.leader.Leader, hvac.api.system_backend.lease.Lease, hvac.api.system_backend.mount.Mount, hvac.api.system_backend.namespace.Namespace, hvac.api.system_backend.policy.Policy, hvac.api.system_backend.raft.Raft, hvac.api.system_backend.seal.Seal, hvac.api.system_backend.wrapping.Wrapping **Methods**

| | |
|---|---|
| *__init__*(adapter) | API Category class constructor. |

**Attributes**

| | |
|---|---|
| *implemented_classes* | Built-in mutable sequence. |
| *unimplemented_classes* | Built-in mutable sequence. |

**__init__**(*adapter*)

API Category class constructor.

**Parameters** **adapter** (hvac.adapters.Adapter) – Instance of *hvac.adapters.Adapter*; used for performing HTTP requests.

**implemented_classes = [<class 'hvac.api.system_backend.audit.Audit'>, <class 'hvac.api**

**unimplemented_classes = []**

**class** hvac.api.system_backend.**SystemBackendMixin**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Base class for System Backend API endpoints.

**class** hvac.api.system_backend.**Wrapping**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin **Methods**

| | |
|---|---|
| *unwrap*([token]) | Return the original response inside the given wrapping token. |

**unwrap**(*token=None*)

Return the original response inside the given wrapping token.

Unlike simply reading cubbyhole/response (which is deprecated), this endpoint provides additional validation checks on the token, returns the original value on the wire rather than a JSON string representation of it, and ensures that the response is properly audit-logged.

**Supported methods:** POST: /sys/wrapping/unwrap. Produces: 200 application/json

**Parameters** **token** (*str | unicode*) – Specifies the wrapping token ID. This is required if the client token is not the wrapping token. Do not use the wrapping token in both locations.

**Returns** The JSON response of the request.

**Return type** dict

## 4.6 hvac.utils

Misc utility functions and constants

**Functions**

| | |
|---|---|
| *comma_delimited_to_list*(list_param) | Convert comma-delimited list / string into a list of strings |
| *deprecated_method*(to_be_removed_in_version) | This is a decorator which can be used to mark methods as deprecated. |
| *format_url*(format_str, *args, **kwargs) | Creates a URL using the specified format after escaping the provided arguments. |
| *generate_method_deprecation_message*(...[, ...]) | Generate a message to be used when warning about the use of deprecated methods. |
| *generate_property_deprecation_message*(...[, ...]) | Generate a message to be used when warning about the use of deprecated properties. |
| *get_token_from_env*() | Get the token from env var, VAULT_TOKEN. |
| *getattr_with_deprecated_properties*(obj, ...) | Helper method to use in the getattr method of a class with deprecated properties. |
| *list_to_comma_delimited*(list_param) | Convert a list of strings into a comma-delimited list / string. |
| *raise_for_error*(method, url, status_code[, ...]) | Helper method to raise exceptions based on the status code of a response received back from Vault. |
| *remove_nones*(params) | Removes None values from optional arguments in a parameter dictionary. |
| *validate_list_of_strings_param*(param_name, ...) | Validate that an argument is a list of strings. |
| *validate_pem_format*(param_name, param_argument) | Validate that an argument is a PEM-formatted public key or certificate |

hvac.utils.**comma_delimited_to_list**(*list_param*)
Convert comma-delimited list / string into a list of strings

**Parameters** **list_param** (*str | unicode*) – Comma-delimited string

**Returns** A list of strings

**Return type** list

hvac.utils.**deprecated_method**(*to_be_removed_in_version*, *new_method=None*)
This is a decorator which can be used to mark methods as deprecated. It will result in a warning being emitted when the function is used.

**Parameters**

- **to_be_removed_in_version** (*str*) – Version of this module the decorated method will be removed in.
- **new_method** (*function*) – Method intended to replace the decorated method. This method's docstrings are included in the decorated method's docstring.

**Returns** Wrapped function that includes a deprecation warning and update docstrings from the replacement method.

**Return type** types.FunctionType

hvac.utils.**format_url**(*format_str*, **args*, ***kwargs*)
Creates a URL using the specified format after escaping the provided arguments.

**Parameters**

- **format_str** (*str*) – The URL containing replacement fields.
- **kwargs** (*dict*) – Positional replacement field values.
- **kwargs** – Named replacement field values.

**Returns** The formatted URL path with escaped replacement fields.

**Return type** str

hvac.utils.**generate_method_deprecation_message**(*to_be_removed_in_version*, *old_method_name*, *method_name=None*, *module_name=None*)

Generate a message to be used when warning about the use of deprecated methods.

**Parameters**

- **to_be_removed_in_version** (*str*) – Version of this module the deprecated method will be removed in.
- **old_method_name** (*str*) – Deprecated method name.
- **method_name** (*str*) – Method intended to replace the deprecated method indicated. This method's docstrings are included in the decorated method's docstring.
- **module_name** (*str*) – Name of the module containing the new method to use.

**Returns** Full deprecation warning message for the indicated method.

**Return type** str

hvac.utils.**generate_property_deprecation_message**(*to_be_removed_in_version*, *old_name*, *new_name*, *new_attribute*, *module_name='Client'*)

Generate a message to be used when warning about the use of deprecated properties.

**Parameters**

- **to_be_removed_in_version** (*str*) – Version of this module the deprecated property will be removed in.
- **old_name** (*str*) – Deprecated property name.
- **new_name** (*str*) – Name of the new property name to use.
- **new_attribute** (*str*) – The new attribute where the new property can be found.
- **module_name** (*str*) – Name of the module containing the new method to use.

**Returns** Full deprecation warning message for the indicated property.

**Return type** str

hvac.utils.**get_token_from_env**()

Get the token from env var, VAULT_TOKEN. If not set, attempt to get the token from, ~/.vault-token

**Returns** The vault token if set, else None

**Return type** str | None

hvac.utils.**getattr_with_deprecated_properties**(*obj*, *item*, *deprecated_properties*)

Helper method to use in the getattr method of a class with deprecated properties.

**Parameters**

- **obj** (*object*) – Instance of the Class containing the deprecated properties in question.
- **item** (*str*) – Name of the attribute being requested.
- **deprecated_properties** (*List[dict]*) – List of deprecated properties. Each item in the list is a dict with at least a "to_be_removed_in_version" and "client_property" key to be used in the displayed deprecation warning.

**Returns** The new property indicated where available.

**Return type** object

hvac.utils.**list_to_comma_delimited**(*list_param*)
Convert a list of strings into a comma-delimited list / string.

**Parameters** **list_param** (*list*) – A list of strings.

**Returns** Comma-delimited string.

**Return type** str

hvac.utils.**raise_for_error**(*method*, *url*, *status_code*, *message=None*, *errors=None*)
Helper method to raise exceptions based on the status code of a response received back from Vault.

**Parameters**

- **method** (*str*) – HTTP method of a request to Vault.
- **url** (*str*) – URL of the endpoint requested in Vault.
- **status_code** (*int*) – Status code received in a response from Vault.
- **message** (*str*) – Optional message to include in a resulting exception.
- **errors** (*list | str*) – Optional errors to include in a resulting exception.

**Raises** hvac.exceptions.InvalidRequest | hvac.exceptions.Unauthorized | hvac.exceptions.Forbidden | hvac.exceptions.InvalidPath | hvac.exceptions.RateLimitExceeded | hvac.exceptions.InternalServerError | hvac.exceptions.VaultNotInitialized | hvac.exceptions.BadGateway | hvac.exceptions.VaultDown | hvac.exceptions.UnexpectedError

hvac.utils.**remove_nones**(*params*)
Removes None values from optional arguments in a parameter dictionary.

**Parameters** **params** (*dict*) – The dictionary of parameters to be filtered.

**Returns** A filtered copy of the parameter dictionary.

**Return type** dict

hvac.utils.**validate_list_of_strings_param**(*param_name*, *param_argument*)
Validate that an argument is a list of strings.

**Parameters**

- **param_name** (*str | unicode*) – The name of the parameter being validated. Used in any resulting exception messages.
- **param_argument** (*list*) – The argument to validate.

**Returns** True if the argument is validated, False otherwise.

**Return type** bool

hvac.utils.**validate_pem_format**(*param_name*, *param_argument*)
Validate that an argument is a PEM-formatted public key or certificate

**Parameters**

- **param_name** (*str | unicode*) – The name of the parameter being validate. Used in any resulting exception messages.
- **param_argument** (*str | unicode*) – The argument to validate

**Returns** True if the argument is validate False otherwise

**Return type** bool

## 4.7 hvac.aws_utils

### Classes

| | |
|---|---|
| *SigV4Auth*(access_key, secret_key[, ...]) | |

### Functions

| | |
|---|---|
| *generate_sigv4_auth_request*([header_value]) | Helper function to prepare a AWS API request to subsequently generate a "AWS Signature Version 4" header. |

**class** hvac.aws_utils.**SigV4Auth**(*access_key*, *secret_key*, *session_token=None*, *region='us-east-1'*)

Bases: object **Methods**

| | |
|---|---|
| *__init__*(access_key, secret_key[, ...]) | Initialize self. |
| *add_auth*(request) | |

**__init__**(*access_key*, *secret_key*, *session_token=None*, *region='us-east-1'*)
Initialize self. See help(type(self)) for accurate signature.

**add_auth**(*request*)

hvac.aws_utils.**generate_sigv4_auth_request**(*header_value=None*)
Helper function to prepare a AWS API request to subsequently generate a "AWS Signature Version 4" header.

**Parameters** **header_value** (*str*) – Vault allows you to require an additional header, X-Vault-AWS-IAM-Server-ID, to be present to mitigate against different types of replay attacks. Depending on the configuration of the AWS auth backend, providing a argument to this optional parameter may be required.

**Returns** A PreparedRequest instance, optionally containing the provided header value under a 'X-Vault-AWS-IAM-Server-ID' header name pointed to AWS's simple token service with action "GetCallerIdentity"

**Return type** requests.PreparedRequest

## 4.8 hvac.adapters

HTTP Client Library Adapters

**Classes**

| | |
|---|---|
| *Adapter*([base_uri, token, cert, verify, ...]) | Abstract base class used when constructing adapters for use with the Client class. |
| *JSONAdapter*([base_uri, token, cert, verify, ...]) | The JSONAdapter adapter class. |
| *RawAdapter*([base_uri, token, cert, verify, ...]) | The RawAdapter adapter class. |
| *Request* | The RawAdapter adapter class. |

**class** hvac.adapters.**Adapter**(*base_uri='http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *namespace=None*, *ignore_exceptions=False*, *strict_http=False*)

Bases: object

Abstract base class used when constructing adapters for use with the Client class. **Methods**

| | |
|---|---|
| *__init__*([base_uri, token, cert, verify, ...]) | Create a new request adapter instance. |
| *auth*(url[, use_token]) | Call to deprecated function 'auth'. |
| *close*() | Close the underlying Requests session. |
| *delete*(url, **kwargs) | Performs a DELETE request. |
| *get*(url, **kwargs) | Performs a GET request. |
| *get_login_token*(response) | Extracts the client token from a login response. |
| *head*(url, **kwargs) | Performs a HEAD request. |
| *list*(url, **kwargs) | Performs a LIST request. |
| *login*(url[, use_token]) | Perform a login request. |
| *post*(url, **kwargs) | Performs a POST request. |
| *put*(url, **kwargs) | Performs a PUT request. |
| *request*(method, url[, headers, raise_exception]) | Main method for routing HTTP requests to the configured Vault base_uri. |
| *urljoin*(*args) | Joins given arguments into a url. |

**__init__**(*base_uri='http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *namespace=None*, *ignore_exceptions=False*, *strict_http=False*)

Create a new request adapter instance.

**Parameters**

- **base_uri** (*str*) – Base URL for the Vault instance being addressed.
- **token** (*str*) – Authentication token to include in requests sent to Vault.
- **cert** (*tuple*) – Certificates for use in requests sent to the Vault instance. This should be a tuple with the certificate and then key.
- **verify** (*Union[bool,str]*) – Either a boolean to indicate whether TLS verification should be performed when sending requests to Vault, or a string pointing at the CA bundle to use for verification. See http://docs.python-requests.org/en/master/user/advanced/#ssl-cert-verification.
- **timeout** (*int*) – The timeout value for requests sent to Vault.

- **proxies** (*dict*) – Proxies to use when preforming requests. See: http://docs.python-requests.org/en/master/user/advanced/#proxies
- **allow_redirects** (*bool*) – Whether to follow redirects when sending requests to Vault.
- **session** (*request.Session*) – Optional session object to use when performing request.
- **namespace** (*str*) – Optional Vault Namespace.
- **ignore_exceptions** (*bool*) – If True, _always_ return the response object for a given request. I.e., don't raise an exception based on response status code, etc.
- **strict_http** (*bool*) – If True, use only standard HTTP verbs in request with additional params, otherwise process as is

**auth**(*url*, *use_token=True*, ***kwargs*)

**Call to deprecated function 'auth'. This method will be removed in version '0.9.0' Please use the 'login' method on tl**
Docstring content from this method's replacement copied below: Perform a login request.

**Associated request is typically to a path prefixed with "/v1/auth") and optionally stores the client token sent**
in the resulting Vault response for use by the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.

**Parameters**

- **url** (*str | unicode*) – Path to send the authentication request to.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the params sent with the request.

**Returns** The response of the auth request.

**Return type** requests.Response

**close**()
Close the underlying Requests session.

**delete**(*url*, ***kwargs*)
Performs a DELETE request.

**Parameters**

- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

**Returns** The response of the request.

**Return type** requests.Response

**get**(*url*, ***kwargs*)
Performs a GET request.

**Parameters**

- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.

- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

**Returns** The response of the request.

**Return type** requests.Response

**abstract get_login_token**(*response*)
Extracts the client token from a login response.

**Parameters** **response** – The response object returned by the login method.

**Returns** A client token.

**Return type** str

**head**(*url*, ***kwargs*)
Performs a HEAD request.

**Parameters**

- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

**Returns** The response of the request.

**Return type** requests.Response

**list**(*url*, ***kwargs*)
Performs a LIST request.

**Parameters**

- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

**Returns** The response of the request.

**Return type** requests.Response

**login**(*url*, *use_token=True*, ***kwargs*)
Perform a login request.

**Associated request is typically to a path prefixed with "/v1/auth") and optionally stores the client token sent** in the resulting Vault response for use by the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.

**Parameters**

- **url** (*str | unicode*) – Path to send the authentication request to.
- **use_token** (*bool*) – if True, uses the token in the response received from the auth request to set the "token" attribute on the the *hvac.adapters.Adapter()* instance under the _adapater Client attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the params sent with the request.

**Returns** The response of the auth request.

**Return type** requests.Response

**post**(*url*, ***kwargs*)
Performs a POST request.

**Parameters**

- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

**Returns** The response of the request.

**Return type** requests.Response

**put**(*url*, ***kwargs*)
Performs a PUT request.

**Parameters**

- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

**Returns** The response of the request.

**Return type** requests.Response

**abstract request**(*method*, *url*, *headers=None*, *raise_exception=True*, ***kwargs*)
Main method for routing HTTP requests to the configured Vault base_uri. Intended to be implement by subclasses.

**Parameters**

- **method** (*str*) – HTTP method to use with the request. E.g., GET, POST, etc.
- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.
- **headers** (*dict*) – Additional headers to include with the request.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.
- **raise_exception** (*bool*) – If True, raise an exception via utils.raise_for_error(). Set this parameter to False to bypass this functionality.

**Returns** The response of the request.

**Return type** requests.Response

**static urljoin**(**args*)
Joins given arguments into a url. Trailing and leading slashes are stripped for each argument.

**Parameters** **args** (*str | unicode*) – Multiple parts of a URL to be combined into one string.

**Returns** Full URL combining all provided arguments

**Return type** str | unicode

**class** hvac.adapters.**JSONAdapter**(*base_uri='http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *namespace=None*, *ignore_exceptions=False*, *strict_http=False*)
Bases: *hvac.adapters.RawAdapter*

The JSONAdapter adapter class. This adapter works just like the RawAdapter adapter except that HTTP 200 responses are returned as JSON dicts. All non-200 responses are returned as Response objects. **Methods**

| | |
|---|---|
| *get_login_token*(response) | Extracts the client token from a login response. |
| *request*(*args, **kwargs) | Main method for routing HTTP requests to the configured Vault base_uri. |

**get_login_token**(*response*)
Extracts the client token from a login response.

**Parameters** **response** (*dict | requests.Response*) – The response object returned by the login method.

**Returns** A client token.

**Return type** str

**request**(**args*, ***kwargs*)
Main method for routing HTTP requests to the configured Vault base_uri.

**Parameters**

- **args** (*list*) – Positional arguments to pass to RawAdapter.request.
- **kwargs** (*dict*) – Keyword arguments to pass to RawAdapter.request.

**Returns** Dict on HTTP 200 with JSON body, otherwise the response object.

**Return type** dict | requests.Response

**class** hvac.adapters.**RawAdapter**(*base_uri='http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *namespace=None*, *ignore_exceptions=False*, *strict_http=False*)
Bases: *hvac.adapters.Adapter*

The RawAdapter adapter class. This adapter adds Vault-specific headers as required and optionally raises exceptions on errors, but always returns Response objects for requests. **Methods**

| | |
|---|---|
| *get_login_token*(response) | Extracts the client token from a login response. |
| *request*(method, url[, headers, raise_exception]) | Main method for routing HTTP requests to the configured Vault base_uri. |

**get_login_token**(*response*)
Extracts the client token from a login response.

**Parameters** **response** (*requests.Response*) – The response object returned by the login method.

**Returns** A client token.

**Return type** str

**request**(*method*, *url*, *headers=None*, *raise_exception=True*, ***kwargs*)
Main method for routing HTTP requests to the configured Vault base_uri.

**Parameters**

- **method** (*str*) – HTTP method to use with the request. E.g., GET, POST, etc.
- **url** (*str | unicode*) – Partial URL path to send the request to. This will be joined to the end of the instance's base_uri attribute.

- **headers** (*dict*) – Additional headers to include with the request.
- **raise_exception** (*bool*) – If True, raise an exception via utils.raise_for_error(). Set this parameter to False to bypass this functionality.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

**Returns** The response of the request.

**Return type** requests.Response

| | |
|---|---|
| get_login_token(response) | Extracts the client token from a login response. |
| request(method, url[, headers, raise_exception]) | Main method for routing HTTP requests to the configured Vault base_uri. |

hvac.adapters.**Request**

**Methods**

alias of *hvac.adapters.RawAdapter*

## 4.9 hvac.exceptions

**Exceptions**

| | |
|---|---|
| *BadGateway*([message, errors, method, url]) | |
| *Forbidden*([message, errors, method, url]) | |
| *InternalServerError*([message, errors, ...]) | |
| *InvalidPath*([message, errors, method, url]) | |
| *InvalidRequest*([message, errors, method, url]) | |
| *ParamValidationError*([message, errors, ...]) | |
| *RateLimitExceeded*([message, errors, method, url]) | |
| *Unauthorized*([message, errors, method, url]) | |
| *UnexpectedError*([message, errors, method, url]) | |
| *VaultDown*([message, errors, method, url]) | |
| *VaultError*([message, errors, method, url]) | |
| *VaultNotInitialized*([message, errors, ...]) | |

**exception** hvac.exceptions.**BadGateway**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**Forbidden**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**InternalServerError**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**InvalidPath**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**InvalidRequest**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**ParamValidationError**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**RateLimitExceeded**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**Unauthorized**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**UnexpectedError**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**VaultDown**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

**exception** hvac.exceptions.**VaultError**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: Exception

**__init__**(*message=None*, *errors=None*, *method=None*, *url=None*)

Initialize self. See help(type(self)) for accurate signature.

**exception** hvac.exceptions.**VaultNotInitialized**(*message=None*, *errors=None*, *method=None*, *url=None*)

Bases: *hvac.exceptions.VaultError*

CHAPTER

FIVE

# CONTRIBUTING

Feel free to open issues and/or pull requests with additional features or improvements! For general questions about contributing to hvac that don't fit in the scope of a GitHub issue, and for any folks are interested in becoming a maintainer of hvac, please feel free to join our gitter chat room for discussions at: gitter.im/hvac/community.

## 5.1 Typical Development Environment Setup

```
virtualenv hvac-env
source hvac-env/bin/activate

git clone https://github.com/hvac/hvac.git
cd hvac
pip install -e .
```

## 5.2 Testing

Integration tests will automatically start a Vault server in the background. Just make sure the latest `vault` binary is available in your `PATH`.

1. Install Vault or execute `tests/scripts/install-vault.sh`. Note: by default this script installs the OSS version of Vault. An enterprise trial version of the Vault binary is available for testing (but has an explicitly limited runtime). To run integration test cases requiring enterprise Vault, you can invoke the installation script with: `install-vault.sh <desired version> 'enterprise'`
2. Install requirements

```
cd hvac
pip install -r requirements.txt
```

1. Run tests: `make test`

## 5.3 Updating Requirements

This project uses pip-tool's `pip-compile` utility to manage its various requirements. Any given requirements file can be manually updated by following the pip-compile comments at the top of the file. Alternatively, the `update-all-requirements` Makefile target can be used to update requirements across the board (this has a dependency on docker being available).

## 5.4 Documentation

### 5.4.1 Adding New Documentation Files

When adding documentation for an entirely new feature / class, it often makes sense to place the documentation in a new `.rst` file. After drafting the new file, be sure to add the file as an entry to at least one table of contents directive (e.g., `toctree`) to ensure it gets rendered and published on https://hvac.readthedocs.io/. As an example, the process for adding a new documentation file for a secrets engine related to Active Directory could involve:

1. Add a new file to `docs/usage/secrets_engines` with a name along the lines of `active_directory.rst`.
2. Update the `toctree` directive within `docs/usage/secrets_engines/index.rst` to add a line for `active_directory`
3. Verify the new file is being included and rendered as expected by running `make html` from the `docs/` subdirectory. You can then view the rendered HTML documentation, in a browser or otherwise, by opening `docs/_build/html/index.html`.

### 5.4.2 Testing Docs

```
cd docs/
pip install -r requirements.txt
make doctest
```

### 5.4.3 Examples

Example code or general guides for methods in this module can be added under docs/usage. Any newly added or updated method in this module will ideally have a corresponding addition to these examples. New usage sections should also be added to the table of contents tracked in docs/usage.rst.

## 5.5 Backwards Compatibility Breaking Changes

Due to the close connection between this module and HashiCorp Vault versions, breaking changes are sometimes required. This can also occur as part of code refactoring to enable improvements in the module generally. In these cases:

- A deprecation notice should be displayed to callers of the module until the minor revision +2. E.g., a notice added in version 0.6.2 could see the marked method / functionality removed in version 0.8.0.
- Breaking changes should be called out in the CHANGELOG.md for the affected version.

## 5.6 Creating / Publishing Releases

- [ ] Checkout the `develop` branch:

```
git checkout develop
git pull
```

- [ ] Update the version number using bumpversion. Releases typically just use the "patch" bumpversion option; but "minor" and "major" are available as needed. This will also add an appropriate git commit for the new version.

```
bumpversion --no-tag {patch|minor|major}
```

- [ ] Pull up the current draft hvac release and use the release-drafter generated release body to update CHANGELOG.md. Then commit the changes:

```
git commit CHANGELOG.md -m "Changelog updates for v$(grep -oP '(?<=current_
↪version = ).*' .bumpversion.cfg)"
```

- [ ] Git push the updated develop branch (`git push`) and open a PR to rebase merge the develop branch into main: https://github.com/hvac/hvac/compare/main...develop. Ensure the PR has the "release" label applied and then merge it.
- [ ] Publish the draft release on GitHub: https://github.com/hvac/hvac/releases. Ensure the tag is set to the release name (e.g., vX.X.X) and the target is the main branch. NOTE: release-drafter sets the release name by default. If performing a minor or major update, these values will need to be manually updated before publishing the draft release subsequently.

CHAPTER

SIX

# CHANGELOG

## 6.1 0.10.13 (May 20th, 2021)

### 6.1.1 Bug Fixes

- Refactor `Cert.login()` Conditional for Python 2.7 Syntax Support. GH-708

## 6.2 0.10.12 (May 19th, 2021)

### 6.2.1 Features

- Add default to `group_type` argument in `update_group` and `create_or_update_group_by_name`. GH-703
- Add Certificate Authentication Methods. GH-691

Thanks to @Tylerlhess, @jeffwecan, @matusf, @mblau-leaffilter and tyhess for their lovely contributions.

## 6.3 0.10.11 (May 7th, 2021)

### 6.3.1 Features

- Expand Transform class to include new(ish) tokenization methods. GH-696
- Add `delete_version_after` KvV2 Param - `configure()` / `update_metadata(). GH-694

### 6.3.2 Miscellaneous

- Bump versions of Vault used in CI workflows. GH-695

Thanks to @jeffwecan for their lovely contributions.

## 6.4 0.10.10 (April 29th, 2021)

### 6.4.1 Features

- AWS Secrets Engine: Add support for iam_tags when creating roles. GH-684
- Add Active Directory generate credential capability. GH-657
- Add `policies` Parameter to Userpass `create_or_update_user()` Method. GH-562
- Add handling of unsupported HTTP methods inside adapter. GH-689
- Add Convenience `read_secret()` Method for KVv2 Class. GH-686

### 6.4.2 Miscellaneous

- Set daemon attribute instead of using setDaemon method that was deprecated in Python 3.10. GH-688

Thanks to @jeffwecan, @mblau-leaffilter, @nicholaswold, @sshishov, @tirkarthi, @tomwerneruk and @vamshide-veloper for their lovely contributions.

## 6.5 0.10.9 (April 2nd, 2021)

### 6.5.1 Bug Fixes

- Send AppRole generate_secret_id Method Metadata Parameter as String GH-689

### 6.5.2 Documentation

- Fix lambda authentication example in aws auth documentation. GH-675
- Docs(secret_engines/pki): Remove 'self' from examples. GH-676

Thanks to @JPoser, @fhemberger, @jeffwecan, @lperdereau and jposer for their lovely contributions.

## 6.6 0.10.8 (February 8th, 2021)

### 6.6.1 Features

- K8s Auth: Allow wildcards for service account and namespace. GH-669
- Add token_type support to create_kubernetes_role. GH-664

## 6.7 0.10.7 (February 1st, 2021)

### 6.7.1 Features

- Support database secrets static roles. GH-662

### 6.7.2 Miscellaneous

- Replace Travis CI w/ GitHub actions. GH-666

Thanks to @jeffwecan, @krish7919 and Krish for their lovely contributions.

## 6.8 0.10.6 (December 14th, 2020)

### 6.8.1 Features

- Enable response wrapping of PKI secrets. GH-649
- Fix OIDC login and add working example. GH-638
- Add rabbitmq vhost_topics parameter. GH-626
- Expand auth_methods module to support AppRole. GH-637

### 6.8.2 Bug Fixes

- Template "auth method not implemented" error message. GH-651
- Fix health.py read_health_status GET method. GH-653
- Fix transit constants for "generate_data_key". GH-632
- Fix PUT method in secrets engine kv_v1 to use PUT instead of POST. GH-629
- Remove Erroneous json() Calls In rabbitmq Class. GH-624

### 6.8.3 Miscellaneous

- Update health.py to match new Vault API query parameters. GH-635
- Remove Consul Secrets Engine create_or_update_role Policy Type Validation. GH-636

Thanks to @Angeall, @JJCella, @briantist, @derBroBro, @discogestalt, @dogfish182, @el-deano, @ghTravis, @godara01, @jeffwecan, @leongyh, @phickey, @tienthanh2509 and @tmcolby for their lovely contributions.

## 6.9 0.10.5 (July 26th, 2020)

### 6.9.1 Features

- Add JWT/OIDC Authentication Method Classes. GH-613
- Add Identity Tokens Methods and Documentation. GH-611
- Add P-521 to list of allowed key types. GH-608
- Add P-384 and RSA-3072 to list of allowed key types. GH-606

### 6.9.2 Bug Fixes

- Options not read by tune_mount_configuration. GH-603

### 6.9.3 Documentation

- Add Autodoc Summaries. GH-612
- Correct Return Type Docstrings Within Transit Class. GH-609
- Transit engine docs for Encrypt Data now refer to encrypt_data. GH-601

### 6.9.4 Miscellaneous

- Update Vault version test matrix / Oldest Support Vault Version. GH-610

Thanks to @akdor1154, @jeffwecan, @ns-jshilkaitis and @trishankatdatadog for their lovely contributions.

## 6.10 0.10.4 (June 16th, 2020)

### 6.10.1 Features

- Extract “renew_self_token” from “renew_token”. GH-598
- Add convenience step_down sys backend method. GH-597

### 6.10.2 Documentation

- Update AWS Auth Docs With Latest Usage . GH-599

Thanks to @jeffwecan, @jm96441n and @pnijhara for their lovely contributions.

## 6.11 0.10.3 (May 24th, 2020)

### 6.11.1 Features

- Add Support For use_token_groups In LDAP Auth Method. GH-591
- Add Raft System Backend Methods. GH-594

Thanks to @finarfin and @jeffwecan for their lovely contributions.

## 6.12 0.10.2 (May 19th, 2020)

### 6.12.1 Features

- Create_role_secret_id: add token_bound_cidrs parameter. GH-585
- Add vault rekey verification methods. GH-586
- Add request data to exception objects. GH-583
- Add marshaling_algorithm to sign/verify params. GH-584
- Add issuer to kubernetes configuration. GH-575

### 6.12.2 Bug Fixes

- Remove json() calls (unneeded following JSONAdapter addition) GH-589

### 6.12.3 Documentation

- Fix format errors in contributing for HTML docs. GH-577

Thanks to @TerryHowe, @and-semakin, @jeffwecan, @jschlyter, @jzck, @mdelaney and @scarabeusiv for their lovely contributions.

## 6.13 0.10.1 (April 7th, 2020)

### 6.13.1 Breaking Changes

- Make returned responses more consistent. GH-537

*Note*: GH-537 changes some methods' return types from None to a request.Response instance. For instance the `client.secrets.identity.lookup_entity` now returns a Response[204] (truthy) value instead of None (falsy) when the lookup returns no results. This change was made to simplify maintenance of response parsing within the hvac code base.

### 6.13.2 Features

- Add support for Transform secrets engine. GH-569

### 6.13.3 Bug Fixes

- Fix "Exception: member entities can't be set manually for external groups". GH-558

Thanks to @jeffwecan, @llamasoft and @msuszko for their lovely contributions.

## 6.14 0.10.0 (February 26th, 2020)

### 6.14.1 Features

- Add a correct endpoint for CRL retrieving . GH-547

### 6.14.2 Documentation

- Fixes close quotes in example usage of read_secret_version. GH-557
- Fixes typo in docs: much -> must. GH-555

### 6.14.3 Miscellaneous

- Don't send optional parameters unless explicitly specified. GH-533

*Note*: GH-533 includes fundamental behavior involving sending parameters to API requests to Vault. Many hvac method parameters that would have been sent with default arguments no longer are included in requests to Vault. Notably, the following behavioral changes should be expected (copied from the related PR comments):

Azure:

- CHANGED: `create_role` parameter `policies` now accepts CSV string or list of strings

Database:

- CHANGED: `create_role` documentation updated to something meaningful

GCP:

- `configure` parameter `google_certs_endpoint` is deprecated
- `create_role` parameter `project_id` is deprecated by `bound_projects` (list)

GitHub:

- `configure` is missing a lot of parameters

LDAP:

- CHANGED: `configure` parameters `user_dn` and `group_dn` made optional
  - Retained argument position to prevent being a breaking change
- CHANGED: `hvac/constants/ldap.py` file removed as it is no longer used

MFA:

- This entire endpoint is deprecated so I didn't bother updating it

Okta:

- CHANGED: `configure` parameter `base_url` default value now differs from API documentation
  - This is likely just a documentation issue
- `register_user`, `read_user`, and `delete_user` duplicate URL parameter `username` in JSON payload
  - I left this one as-is as it doesn't appear to hurt anything
- Ditto for `delete_group`, but `register_group` and `list_group` correctly omit it

PKI:

- CHANGED: `sign_data` and `verify_signed_data` optional parameter `marshaling_algorithm` added

RADIUS:

- `configure` is missing a lot of parameters
- BUG: `register_user` attempted to convert `username` string into a CSV list (?!) for POST data
  - Didn't hurt anything as `username` is extracted from URL path in Vault server
- BUG: `register_user` parameter `policies` never actually passed as parameter

System Backend:

- Auth
  - `enable_auth_method` parameter `plugin_name` is deprecated
  - CHANGED: `enable_audit_device` optional parameter `local` was added
- Init
  - `initialize` provides default for required API parameters `secret_shares` and `secret_threshold`
- Key
  - `start_root_token_generation` parameter `otp` is deprecated

**Misc:**

- There seems to be some discrepancy on how "extra arguments" are accepted:
  - Some methods use only `**kwargs` (e.g. `hvac/api/system_backend/auth.py`)
  - Some use `*args` and `**kwargs` (e.g. `hvac/api/secrets_engines/active_directory.py`)
  - `hvac/api/secrets_engines/pki.py` uses `extra_params={}`
- Most argument names match API parameter names, but some don't
  - Example: `hvac/api/auth_methods/ldap.py` `configure` uses `user_dn` instead of `userdn`
  - Example: `hvac/api/system_backend/auth.py` `configure` uses `method_type` instead of `type`
- Many methods duplicate URL parameters into JSON payload as well
  - This isn't necessary and fortunately Vault ignores the extra parameters
- `ttl`, `max_ttl`, `policies`, `period`, `num_uses` and a few other fields are deprecated as of Vault version 1.2.0

  - https://github.com/hashicorp/vault/blob/master/CHANGELOG.md#120-july-30th-2019

Thanks to @findmyname666, @llamasoft, @moisesguimaraes, @philherbert and Adrian Eib for their lovely contributions.

## 6.15 0.9.6 (November 20th, 2019)

### 6.15.1 Features

- Added userpass auth method. GH-519
- added rabbitmq secrets backend. GH-540
- Quote/Escape all URL placeholders. GH-532

### 6.15.2 Documentation

- Getting Started Guide and LDAP Auth Updates. GH-524

### 6.15.3 Miscellaneous

- Handle bad gateway from Vault. GH-542
- Fix GET/LIST typos. GH-536
- Fix Travis HEAD build + Overhaul install scripts. GH-535
- Improve Integration Test Error Handling. GH-531

Thanks to @DaveDeCaprio, @Dowwie, @drewmullen, @jeffwecan, @llamasoft and @vamshideveloper for their lovely contributions.

## 6.16 0.9.5 (July 19th, 2019)

### 6.16.1 Features

- Add Active Directory Secrets Engine Support. GH-508

### 6.16.2 Documentation

- Include Recently Added Namespace Documentation In Toctree. GH-509

Thanks to @jeffwecan and @vamshideveloper for their lovely contributions.

## 6.17 0.9.4 (July 18th, 2019)

### 6.17.1 Features

- Add delete_namespace Method and Establish Namespace Documentation. GH-500

### 6.17.2 Bug Fixes

- Fix consul configure_access/create_or_update_role Method Return Values. GH-502

### 6.17.3 Documentation

- Fix Database generate_credentials Docstring Params. GH-498

### 6.17.4 Miscellaneous

- Add config for updatedocs app. GH-495
- Add a Codeowners file for automatic reviewer assignments. GH-494

Thanks to @Tylerlhess, @drewmullen and @jeffwecan for their lovely contributions.

## 6.18 0.9.3 (July 7th, 2019)

### 6.18.1 Features

- Add Create and List Namespace System Backend Methods. GH-489
- Expanded Support for AWS Auth Method. GH-482
- Capabilities System Backend Support. GH-476

### 6.18.2 Bug Fixes

- GCP Auth Test Case Updates For Changes in Vault v1.1.1+. GH-487
- Change AWS `generate_credentials` request method to GET. GH-475

### 6.18.3 Documentation

- Numerous Fixes and Doctest Support for Transit Secrets Engine. GH-486

### 6.18.4 Miscellaneous

- Start Using Enterprise (Trial) Version of Vault For Travis CI Builds. GH-478
- Update Travis CI Test Matrix With Latest Vault Version & Drop Python 3.6. GH-488
- Set up release-drafter / *mostly* automated releases. GH-485

Thanks to @donjar, @fhemberger, @jeffwecan, @stevefranks and @stevenmanton for their lovely contributions.

## 6.19 0.9.2 (June 8th, 2019)

BUG FIXES:

- Fix kubernetes auth method list roles method. GH-466
- Enable consul secrets engine. GH-460
- Enable database secrets engine. GH-455
- Many fixes for the database secrets engine. GH-457

IMPROVEMENTS:

- The `enable_auth_method()`, `tune_auth_method()`, `enable_secrets_engine()`, `tune_mount_configuration()` system backend method now take arbitrary `**kwargs` parameters to provide greater support for variations in accepted parameters in the underlying Vault plugins.
- Azure auth params, add `num_uses`, change `bound_location` -> `bound_locations` and `bound_resource_group_names` -> `bound_resource_groups`. GH-452

MISCELLANEOUS:

- The hvac project now has gitter chat enabled. Feel free to check it out for any online discussions related to this module at: gitter.im/hvac/community)! GH-465
- Added Vault agent socket listener usage example under the "advanced usage" documentation section at: hvac.readthedocs.io GH-468

Thanks to @denisvll, @Dudesons, and @drewmullen for their lovely contributions.

## 6.20 0.9.1 (May 25th, 2019)

BUG FIXES:

- Fix Azure list roles GH-448

IMPROVEMENTS:

- Support for the PKI secrets engine. GH-436

MISCELLANEOUS:

- `delete_roleset()` method added to GCP secrets engine support. GH-449

Thanks to @nledez and @drewmullen for their lovely contributions.

## 6.21 0.9.0 (May 23rd, 2019)

BUG FIXES:

- Update path to azure.login() GH-429
- AWS secrets engine generate credentials updated to a post request. GH-430

IMPROVEMENTS:

- Support for the Radius auth method. GH-420
- Support for the Database secrets engine. GH-431
- Add the consul secret engine support GH-432
- Support for the GCP secrets engine. GH-443

MISCELLANEOUS:

- Remove logger call within adapters module GH-445
- Add docs for auth_cubbyhole GH-427

Thanks to @paulcaskey, @stevenmanton, @brad-alexander, @yoyomeng2, @JadeHayes, @Dudesons for their lovely contributions.

## 6.22 0.8.2 (April 4th, 2019)

BUG FIXES:

- Fix priority of client url and VAULT_ADDR environment variable. GH-423
- Update setup.py to only compile hvac package. GH-418

Thanks to @eltoder and @andytumelty for their lovely contributions.

## 6.23 0.8.1 (March 31st, 2019)

BUG FIXES:

- Fix `initialize()` method `recovery_shares` and `recovery_threshold` parameter validation regression. GH-416

## 6.24 0.8.0 (March 29th, 2019)

BACKWARDS COMPATIBILITY NOTICE:

- The `Client()` class constructor now behaves similarly to Vault CLI in that it uses the `VAULT_ADDR` environmental variable for the Client URL when that variable is set. Along the same lines, when no token is passed into the `Client()` constructor, it will attempt to load a token from the `VAULT_TOKEN` environmental variable or the `~/.vault-token` file where available. GH-411

IMPROVEMENTS:

- Support for the Kubernetes auth method. GH-408

BUG FIXES:

- Fix for comparision `recovery_threshold` and `recovery_shares` during initialization. GH-398
- Fix request method for AWS secrets engine `generate_credentials()` method. GH-403
- Fix request parameter (`n_bytes` -> `bytes`) for Transit secrets engine `generate_random_bytes()` method. GH-377

Thanks to @engstrom, @viralpoetry, @bootswithdefer, @steved, @kserrano, @spbsoluble, @uepoch, @singuliere, @frgaudet, @jsporna, & @mrsiesta for their lovely contributions.

## 6.25 0.7.2 (January 1st, 2019)

IMPROVEMENTS:

- Support for the AWS secrets engine. GH-370

BUG FIXES:

- Fixes for intermittent test case failures. GH-361 & GH-364

MISCELLANEOUS:

- Travis CI builds now run against Python 3.7 (along side the previously tested 2.7 and 3.6 versions). GH-360
- Documentation build test case added. GH-366
- Module version now managed by the `bumpversion` utility exclusively. GH-369

## 6.26 0.7.1 (December 19th, 2018)

IMPROVEMENTS:

- Support for the Okta auth method. GH-341

BUG FIXES:

- Simplify redirect handling in `Adapter` class to fix issues following location headers with fully qualified URLs. Note: hvac now converts `//` to `/` within any paths. GH-348
- Fixed a bug where entity and group member IDs were not being passed in to Identity secrets engine group creation / updates. GH-346
- Ensure all types of responses for the `read_health_status()` system backend method can be retrieved without exceptions being raised. GH-347
- Fix `read_seal_status()` in `Client` class's `seal_status` property. GH-354

DOCUMENTATION UPDATES:

- Example GCP auth method `login()` call with google-api-python-client usage added: Example with google-api-python-client Usage. GH-350

MISCELLANEOUS:

- Note: Starting after release 0.7.0, `develop` is the main integration branch for the hvac project. The `main` branch is now intended to capture the state of the most recent release.
- Test cases for hvac are no longer included in the release artifacts published to PyPi. GH-334
- The `create_or_update_policy` system backend method now supports a "pretty_print" argument for different JSON formatting. This allows create more viewable policy documents when retrieve existing policies (e.g., from within the Vault UI interface). GH-342

- Explicit support for Vault v0.8.3 dropped. CI/CD tests updated to run against Vault v1.0.0. GH-344

## 6.27 0.7.0 (November 1st, 2018)

DEPRECATION NOTICES:

- All auth method classes are now accessible under the `auth` property on the `hvac.Client` class. GH-310. (E.g. the `github`, `ldap`, and `mfa` Client properties' methods are now accessible under `Client.auth.github`, etc.)
- All secrets engines classes are now accessible under the `secrets` property on the `hvac.Client` class. GH-311 (E.g. the `kv`, Client property's methods are now accessible under `Client.secrets.kv`)
- All system backend classes are now accessible under the `sys` property on the `hvac.Client` class. GH-314 ([GH-314] through [GH-325]) (E.g. methods such as `enable_secret_backend()` under the Client class are now accessible under `Client.sys.enable_secrets_engine()`, etc.)

IMPROVEMENTS:

- Support for Vault Namespaces. GH-268
- Support for the Identity secrets engine. GH-269
- Support for the GCP auth method. GH-240
- Support for the Azure auth method. GH-286
- Support for the Azure secrets engine. GH-287
- Expanded Transit secrets engine support. GH-303

Thanks to @tiny-dancer, @jacquat, @deejay1, @MJ111, @jasonarewhy, and @alexandernst for their lovely contributions.

## 6.28 0.6.4 (September 5th, 2018)

IMPROVEMENTS:

- New KV secret engine-related classes added. See the KV documentation under hvac's readthedocs.io site for usage / examples. GH-257 / GH-260

MISCELLANEOUS:

- Language classifiers are now being included with the distribution. GH-247
- Token no longer being sent in URL path for the `Client.renew_token` method. GH-250
- Support for the response structure in newer versions of Vault within the `Client.get_policy` method. GH-254
- `config` and `plugin_name` parameters added to the `Client.enable_auth_backend` method. GH-253

Thanks to @ijl, @rastut, @seuf, @downeast for their lovely contributions.

## 6.29 0.6.3 (August 8th, 2018)

DEPRECATION NOTICES:

- The `auth_github()` method within the `hvac.Client` class has been marked as deprecated and will be removed in hvac v0.8.0 (or later). Please update any callers of this method to use the `hvac.Client.github.login()` instead.
- The `auth_ldap()` method within the `hvac.Client` class has been marked as deprecated and will be removed in hvac v0.8.0 (or later). Please update any callers of this method to use the `hvac.Client.ldap.login()` instead.

IMPROVEMENTS:

- New Github auth method class added. See the Github documentation for usage / examples. GH-242
- New Ldap auth method class added. See the Ldap documentation for usage / examples. GH-244
- New Mfa auth method class added. See the documentation for usage / examples. GH-255
- `auth_aws_iam()` method updated to include "region" parameter for deployments in different AWS regions. GH-243

DOCUMENTATION UPDATES:

- Additional guidance for how to configure hvac's `Client` class to leverage self-signed certificates / private CA bundles has been added at: Making Use of Private CA. GH-230
- Docstring for `verify` `Client` parameter corrected and expanded. GH-238

MISCELLANEOUS:

- Automated PyPi deploys via travis-ci removed. GH-226
- Repository transferred to the new "hvac" GitHub organization; thanks @ianunruh! GH-227
- Codecov (automatic code coverage reports) added. GH-229 / GH-228
- Tests subdirectory reorganized; now broken up by integration versus unit tests with subdirectories matching the module path for the code under test. GH-236

Thanks to @otakup0pe, @FabianFrank, @andrewheald for their lovely contributions.

## 6.30 0.6.2 (July 19th, 2018)

BACKWARDS COMPATIBILITY NOTICE:

- With the newly added `hvac.adapters.Request` class, request kwargs can no longer be directly modified via the `_kwargs` attribute on the `Client` class. If runtime modifications to this dictionary are required, callers either need to explicitly pass in a new `adapter` instance with the desired settings via the `adapter` propery on the `Client` class *or* access the `_kwargs` property via the `adapter` property on the `Client` class.

See the Advanced Usage section of this module's documentation for additional details.

IMPROVEMENTS:

- sphinx documentation and readthedocs.io project added. GH-222
- README.md included in setuptools metadata. GH-222
- All `tune_secret_backend()` parameters now accepted. GH-215
- Add `read_lease()` method GH-218

- Added adapter module with `Request` class to abstract HTTP requests away from the `Client` class. GH-223

Thanks to @bbayszczak, @jvanbrunschot-coolblue for their lovely contributions.

## 6.31 0.6.1 (July 5th, 2018)

IMPROVEMENTS:

- Update `unwrap()` method to match current Vault versions [GH-149]
- Initial support for Kubernetes authentication backend [GH-210]
- Initial support for Google Cloud Platform (GCP) authentication backend [GH-206]
- Update enable_secret_backend function to support kv version 2 [GH-201]

BUG FIXES:

- Change URL parsing to allow for routes in the base Vault address (e.g., `https://example.com/vault`) [GH-212].

Thanks to @mracter, @cdsf, @SiN, @seanmalloy, for their lovely contributions.

## 6.32 0.6.0 (June 14, 2018)

BACKWARDS COMPATIBILITY NOTICE:

- Token revocation now sends the token in the request payload. Requires Vault >0.6.5
- Various methods have new and/or re-ordered keyword arguments. Code calling these methods with positional arguments may need to be modified.

IMPROVEMENTS:

- Ensure mount_point Parameter for All AWS EC2 Methods [GH-195]
- Add Methods for Auth Backend Tuning [GH-193]
- Customizable approle path / mount_point [GH-190]
- Add more methods for the userpass backend [GH-175]
- Add transit signature_algorithm parameter [GH-174]
- Add auth_iam_aws() method [GH-170]
- lookup_token function POST token not GET [GH-164]
- Create_role_secret_id with wrap_ttl & fix get_role_secret_id_accessor [GH-159]
- Fixed json() from dict bug and added additional arguments on auth_ec2() method [GH-157]
- Support specifying period when creating EC2 roles [GH-140]
- Added support for /sys/generate-root endpoint [GH-131] / [GH-199]
- Added “auth_cubbyhole” method [GH-119]
- Send token/accessor as a payload to avoid being logged [GH-117]
- Add AppRole delete_role method [GH-112]

BUG FIXES:

- Always Specify auth_type In create_ec2_role [GH-197]
- Fix "double parasing" of JSON response in auth_ec2 method [GH-181]

Thanks to @freimer, @ramiamar, @marcoslopes, @ianwestcott, @marc-sensenich, @sunghyun-lee, @jnaulty, @sijis, @Myles-Steinhauser-Bose, @oxmane, @ltm, @bchannak, @tkinz27, @crmulliner, for their lovely contributions.

## 6.33 0.5.0 (February 20, 2018)

IMPROVEMENTS:

- Added `disallowed_policies` parameter to `create_token_role` method [GH-169]

Thanks to @morganda for their lovely contribution.

## 6.34 0.4.0 (February 1, 2018)

IMPROVEMENTS:

- Add support for the `period` parameter on token creation [GH-167]
- Add support for the `cidr_list` parameter for approle secrets [GH-114]

BUG FIXES:

- Documentation is now more accurate [GH-165] / [GH-154]

Thanks to @ti-mo, @dhoeric, @RAbraham, @lhdumittan, @ahsanali for their lovely contributions.

## 6.35 0.3.0 (November 9, 2017)

This is just the highlights, there have been a bunch of changes!

IMPROVEVEMENTS:

- Some AppRole support [GH-77]
- Response Wrapping [GH-85]
- AWS EC2 stuff [GH-107], [GH-109]

BUG FIXES

- Better handling of various error states [GH-79], [GH-125]

Thanks to @ianwestcott, @s3u, @mracter, @intgr, @jkdihenkar, @gaelL, @henriquegemignani, @bfeeser, @nicr9, @mwielgoszewski, @mtougeron for their contributions!

## 6.36 0.2.17 (December 15, 2016)

IMPROVEMENTS:

- Add token role support [GH-94]
- Add support for Python 2.6 [GH-92]
- Allow setting the explicit_max_ttl when creating a token [GH-81]
- Add support for write response wrapping [GH-85]

BUG FIXES:

- Fix app role endpoints for newer versions of Vault [GH-93]

## 6.37 0.2.16 (September 12, 2016)

Thanks to @otakup0pe, @nicr9, @marcoslopes, @caiotomazelli, and @blarghmatey for their contributions!

IMPROVEMENTS:

- Add EC2 auth support [GH-61]
- Add support for token accessors [GH-69]
- Add support for response wrapping [GH-70]
- Add AppRole auth support [GH-77]

BUG FIXES:

- Fix `no_default_policy` parameter in `create_token` [GH-65]
- Fix EC2 auth double JSON parsing [GH-76]

## 6.38 0.2.15 (June 22nd, 2016)

Thanks to @blarghmatey, @stevenmanton, and @ahlinc for their contributions!

IMPROVEMENTS:

- Add methods for manipulating app/user IDs [GH-62]
- Add ability to automatically parse policies with pyhcl [GH-58]
- Add TTL option to `create_userpass` [GH-60]
- Add support for backing up keys on rekey [GH-57]
- Handle non-JSON error responses correctly [GH-46]

BUG FIXES:

- `is_authenticated` now handles new error type for Vault 0.6.0

## 6.39 0.2.14 (June 2nd, 2016)

BUG FIXES:

- Fix improper URL being used when leader redirection occurs [GH-56]

## 6.40 0.2.13 (May 31st, 2016)

IMPROVEMENTS:

- Add support for Requests sessions [GH-53]

BUG FIXES:

- Properly handle redirects from Vault server [GH-51]

## 6.41 0.2.12 (May 12th, 2016)

IMPROVEMENTS:

- Add support for `increment` in renewel of secret [GH-48]

BUG FIXES:

- Use unicode literals when constructing URLs [GH-50]

## 6.42 0.2.10 (April 8th, 2016)

IMPROVEMENTS:

- Add support for list operation [GH-47]

## 6.43 0.2.9 (March 18th, 2016)

IMPROVEMENTS:

- Add support for nonce during rekey operation [GH-42]
- Add get method for policies [GH-43]
- Add delete method for userpass auth backend [GH-45]
- Add support for response to rekey init

## 6.44 0.2.8 (February 2nd, 2016)

IMPROVEMENTS:

- Convenience methods for managing userpass and app-id entries
- Support for new API changes in Vault v0.4.0

## 6.45 0.2.7 (December 16th, 2015)

IMPROVEMENTS:

- Add support for PGP keys when rekeying [GH-28]

BUG FIXES:

- Fixed token metadata parameter [GH-27]

## 6.46 0.2.6 (October 30th, 2015)

IMPROVEMENTS:

- Add support for `revoke-self`
- Restrict `requests` dependency to modern version

## 6.47 0.2.5 (September 29th, 2015)

IMPROVEMENTS:

- Add support for API changes/additions in Vault v0.3.0
  - Tunable config on secret backends
  - MFA on username/password and LDAP auth backends
  - PGP encryption for unseal keys

## 6.48 0.2.4 (July 23rd, 2015)

BUG FIXES:

- Fix write response handling [GH-19]

## 6.49 0.2.3 (July 18th, 2015)

BUG FIXES

- Fix error handling for next Vault release

IMPROVEMENTS:

- Add support for rekey/rotate APIs

## 6.50 0.2.2 (June 12th, 2015)

BUG FIXES:

- Restrict `requests` dependency to 2.5.0 or later

IMPROVEMENTS:

- Return latest seal status from `unseal_multi`

## 6.51 0.2.1 (June 3rd, 2015)

BUG FIXES:

- Use arguments passed to `initialize` method

## 6.52 0.2.0 (May 25th, 2015)

BACKWARDS COMPATIBILITY NOTICE:

- Requires Vault 0.1.2 or later for `X-Vault-Token` header
- `auth_token` method removed in favor of `token` property
- `read` method no longer raises `hvac.exceptions.InvalidPath` on nonexistent paths

IMPROVEMENTS:

- Tolerate falsey URL in client constructor
- Add ability to auth without changing to new token
- Add `is_authenticated` convenience method
- Return `None` when reading nonexistent path

## 6.53 0.1.1 (May 20th, 2015)

IMPROVEMENTS:

- Add `is_sealed` convenience method
- Add `unseal_multi` convenience method

BUG FIXES:

- Remove secret_shares argument from `unseal` method

## 6.54 0.1.0 (May 17th, 2015)

- Initial release

CHAPTER

SEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

Source code repository hosted at github.com/hvac/hvac.

# PYTHON MODULE INDEX

## h

# INDEX

## Symbols



## A



## B



## C

## D

## E



## F



## G

## H



## I

## J



## K



## L

## M

## N



## O



## P



## R

## S



## T

## U



## V

## W