hvac

Release 0.6.1

Nov 01, 2018

Contents:

Source code repository hosted at github.com/hvac/hvac.

CHAPTER 1

hvac

HashiCorp Vault API client for Python 2.7/3.x Tested against the latest release, HEAD ref, and 3 previous major versions (counting back from the latest release) of Vault. Currently supports Vault v0.8.3 or later.

1.1 Documentation

Documentation for this module is hosted on readthedocs.io.

1.2 Getting started

1.2.1 Installation

```
pip install hvac
```

or

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

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

1.2.2 Initialize the client

```
import os

import hvac

# Using plaintext
client = hvac.Client()
```

(continues on next page)

(continued from previous page)

```
client = hvac.Client(url='http://localhost:8200')
client = hvac.Client(url='http://localhost:8200', token=os.environ['VAULT_TOKEN'])

# Using TLS
client = hvac.Client(url='https://localhost:8200')

# Using TLS with client-side certificate authentication
client = hvac.Client(url='https://localhost:8200', cert=('path/to/cert.pem', 'path/to/
↪key.pem'))

# Using Namespace
client = hvac.Client(url='http://localhost:8200', token=os.environ['VAULT_TOKEN'],␣
↪namespace=os.environ['VAULT_NAMESPACE'])
```

1.2.3 Read and write to secret backends

```
client.write('secret/foo', baz='bar', lease='1h')

print(client.read('secret/foo'))

client.delete('secret/foo')
```

1.2.4 Authenticate using token auth backend

```
# Token
client.token = 'MY_TOKEN'
assert client.is_authenticated() # => True
```

CHAPTER 2

Usage

2.1 Secrets Engines

2.1.1 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()
```

(continues on next page)

(continued from previous page)

```
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,
)
```

2.1.2 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',
        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()
```

(continues on next page)

(continued from previous page)

```
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_entities_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(
        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))
```

2.1.3 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(
    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'],
```

(continues on next page)

(continued from previous page)

```
))
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(
    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,
```

(continues on next page)

(continued from previous page)

```
)

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',
)

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'),
)
```

(continues on next page)

(continued from previous page)

```
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,
)
```

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',
)
```

2.1.4 Transit

Create Key

hvac.api.secrets_engines.Transit.create_key()

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

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

Read Key

hvac.api.secrets_engines.Transit.read_key()

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

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))
```

List Keys

hvac.api.secrets_engines.Transit.list_keys()

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

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))
```

Delete Key

hvac.api.secrets_engines.Transit.delete_key()

```
import hvac
client = hvac.Client()
client.secrets.transit.delete_key(name='hvac-key')
```

Update Key Configuration

hvac.api.secrets_engines.Transit.update_key_configuration()

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

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

Rotate Key

hvac.api.secrets_engines.Transit.rotate_key()

```
import hvac
client = hvac.Client()
client.secrets.transit.rotate_key(name='hvac-key')
```

Export Key

hvac.api.secrets_engines.Transit.encrypt_key()

```
import hvac
client = hvac.Client()
export_key_response = client.secrets.transit.export_key(name='hvac-key')

first_key = export_key_response['keys']['1']
```

Encrypt Data

hvac.api.secrets_engines.Transit.decrypt_data()

```
import base64
import hvac
client = hvac.Client()

encrypt_data_response = client.secrets.transit.encrypt_data(
        name='hvac-key',
        plaintext=base64.urlsafe_b64encode('hi its me hvac').decode('ascii'),
)
ciphertext = encrypt_data_response['data']['ciphertext']
print('Encrypted plaintext ciphertext is: {cipher}'.format(cipher=ciphertext))
```

Decrypt Data

hvac.api.secrets_engines.Transit.decrypt_data()

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

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

Rewrap Data

hvac.api.secrets_engines.Transit.rewrap_data()

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

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))
```

Generate Data Key

hvac.api.secrets_engines.Transit.generate_data_key()

```
import hvac
client = hvac.Client()
gen_key_response = client.secrets.transit.generate_data_key(name='hvac-key')
ciphertext = gen_data_key_response['data']
print('Generated data key is: {cipher}'.format(cipher=ciphertext))
```

Generate Random Bytes

hvac.api.secrets_engines.Transit.generate_random_bytes()

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

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))
```

Hash Data

hvac.api.secrets_engines.Transit.hash_data()

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

hash_data_response = client.secrets.transit.hash_data(
        name='hvac-key',
        hash_input=base64.urlsafe_b64encode('hi its me hvac').decode('ascii'),
)
sum = hash_data_response['data']['sum']
print('Hashed data is: {sum}'.format(sum=sum))
```

Generate Hmac

hvac.api.secrets_engines.Transit.generate_hmac()

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

generate_hmac_response = client.secrets.transit.hash_data(
        name='hvac-key',
        hash_input=base64.urlsafe_b64encode('hi its me hvac').decode('ascii'),
)
hmac = generate_hmac_response['data']['sum']
print('HMAC'd data is: {hmac}'.format(hmac=hmac))
```

Sign Data

hvac.api.secrets_engines.Transit.sign_data()

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

sign_data_response = client.secrets.transit.sign_data(
        name='hvac-key',
        hash_input=base64.urlsafe_b64encode('hi its me hvac').decode('ascii'),
)
signature = sign_data_response['data']['signature']
print('Signature is: {signature}'.format(signature=signature))
```

Verify Signed Data

hvac.api.secrets_engines.Transit.verify_signed_data()

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

verify_signed_data_response = client.secrets.transit.verify_signed_data(
        name='hvac-key',
        hash_input=base64.urlsafe_b64encode('hi its me hvac').decode('ascii'),
)
valid = verify_signed_data_response['data']['valid']
print('Signature is valid?: {valid}'.format(valid=valid))
```

Backup Key

hvac.api.secrets_engines.Transit.backup_key()

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

backup_key_response = client.secrets.transit.backup_key(
        name='hvac-key',
        mount_point=TEST_MOUNT_POINT,
)
backed_up_key = backup_key_response['data']['backup']
```

Restore Key

hvac.api.secrets_engines.Transit.restore_key()

```
import hvac
client = hvac.Client()
client.secrets.transit.restore_key(backup=backed_up_key)
```

Trim Key

hvac.api.secrets_engines.Transit.trim_key()

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

client.secrets.transit.trim_key(
        name='hvac-key',
        min_version=3,
)
```

2.2 Auth Methods

2.2.1 Approle

Authentication

```
client.auth_approle('MY_ROLE_ID', 'MY_SECRET_ID')
```

2.2.2 AWS

Contents

IAM Authentication

Source reference: *hvac.v1.Client.auth_aws_iam()*

Static Access Key Strings

Various examples of authenticating with static access key strings:

```
import hvac

client = hvac.Client()

client.auth_aws_iam('MY_AWS_ACCESS_KEY_ID', 'MY_AWS_SECRET_ACCESS_KEY')
client.auth_aws_iam('MY_AWS_ACCESS_KEY_ID', 'MY_AWS_SECRET_ACCESS_KEY', 'MY_AWS_
↪SESSION_TOKEN')
client.auth_aws_iam('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(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
```

(continues on next page)

(continued from previous page)

```
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(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']
    else:
        security_credentials = load_aws_ec2_role_iam_credentials(iam_role)
        return security_credentials['AccessKeyId'], security_credentials[
↪'SecretAccessKey']


access_key_id, secret_access_key = infer_credentials_from_iam_role('some-role')

client = hvac.Client()
client.auth_aws_iam(access_key_id, secret_access_key)
```

Caveats For Non-Default AWS Regions

I.e., calling *hvac.v1.Client.auth_aws_iam()* 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.create_vault_ec2_client_configuration(
    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(
    creds.access_key,
    creds.secret_key,
    creds.token,
    region="us-west-1",
    header_value=VAULT_HEADER_VALUE,
    role='some-role,
    use_token=True,
)
```

EC2 Authentication

Source reference: *hvac.v1.Client.auth_ec2()*

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
```

(continues on next page)

(continued from previous page)

```
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)
    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):
```

(continues on next page)

(continued from previous page)

```
    """
    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.')
        nonce = load_aws_ec2_nonce_from_disk()

    auth_ec2_resp = vault_client.auth_ec2(
        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:
```

(continues on next page)

(continued from previous page)

```
            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')

    return vault_client


authenticated_vault_client = get_vault_client()
```

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.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

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

`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

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

hvac.api.auth.Gcp.delete_config()

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

client.auth.gcp.delete_config()
```

create-role

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

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'],
)
```

(continues on next page)

(continued from previous page)

```
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

`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

`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

`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

`hvac.api.auth.Gcp.delete_role()`

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

client.gcp.delete_role(
)
```

Login

`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
```

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(
```

(continues on next page)

(continued from previous page)

```
    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 Kubernetes

Authentication

```
# Kubernetes (from k8s pod)
f = open('/var/run/secrets/kubernetes.io/serviceaccount/token')
jwt = f.read()
client.auth_kubernetes("example", jwt)
```

2.2.7 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.list_auth_backends():
    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',
```

(continues on next page)

(continued from previous page)

```
    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(
```

(continues on next page)

(continued from previous page)

```
    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(
```

(continues on next page)

(continued from previous page)

```
    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
```

2.2.8 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,
)
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
```

2.2.9 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'])
```

2.2.10 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)
```

(continues on next page)

(continued from previous page)

```
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

```
audit_devices = client.list_enabled_audit_devices()

options = {
        'path': '/tmp/vault.log',
        'log_raw': True,
}

client.enable_audit_device('file', options=options, path='somefile')
client.disable_audit_backend('oldfile')
```

List Enabled Audit Devices

hvac.api.system_backend.Audit.list_enabled_audit_devices()

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

enabled_audit_devices = self.client.sys.list_enabled_audit_devices()
print('The following audit devices are enabled: {audit_devices_list}'.format(
        audit_devices_list=enabled_audit_devices['data'].keys(),
)
```

Enable Audit Device

hvac.api.system_backend.Audit.enable_audit_device()

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

options = {
        'path': '/tmp/vault.audit.log'
```

(continues on next page)

(continued from previous page)

```
}

self.client.sys.enable_audit_device(
        device_type='file',
        options=options,
        path='tmp-file-audit',
)
```

Disable Audit Device

hvac.api.system_backend.Audit.disable_audit_device()

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

self.client.sys.disable_audit_device(
        path='tmp-file-audit',
)
```

Calculate Hash

hvac.api.system_backend.Audit.calculate_hash()

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

input_to_hash = input()

audit_hash = self.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'])
```

2.3.2 Auth

```
methods = client.sys.list_auth_methods()

client.sys.enable_auth_method('userpass', path='customuserpass')
client.sys.disable_auth_method('github')
```

List Auth Methods

hvac.api.system_backend.Auth.list_auth_methods()

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

auth_methods = self.client.sys.list_auth_methods()
```

(continues on next page)

(continued from previous page)

```
print('The following auth methods are enabled: {auth_methods_list}'.format(
        auth_methods_list=auth_methods['data'].keys(),
)
```

Enable Auth Method

hvac.api.system_backend.Auth.enable_auth_method()

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

self.client.sys.enable_auth_method(
        method_type='github',
        path='hvac-github',
)
```

Disable Auth Method

hvac.api.system_backend.Auth.disable_auth_method()

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

self.client.sys.disable_auth_method(
        path='hvac-github',
)
```

Read Auth Method Tuning

hvac.api.system_backend.Auth.read_auth_method_tuning()

```
import hvac
client = hvac.Client()
response = self.client.sys.read_auth_method_tuning(
        path='github-hvac',
        description='The Github auth method for hvac users',
)

print('The max lease TTL for the auth method under path "github-hvac" is: {max_ttl}'.
↪format(
        max_ttl=response['data']['max_lease_ttl'],
)
```

Tune Auth Method

hvac.api.system_backend.Auth.tune_auth_method()

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

(continues on next page)

(continued from previous page)

```
self.client.sys.tune_auth_method(
        path=self.TEST_AUTH_METHOD_PATH,
        description='The Github auth method for hvac users',
)
```

2.3.3 Health

Read Status

hvac.api.system_backend.Health.read_health_status()

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

status = self.client.sys.read_health_status()
print('Vault initialization status is: %s' % status['initialized'])
```

2.3.4 Init

```
methods = client.sys.list_auth_methods()

client.sys.enable_auth_method('userpass', path='customuserpass')
client.sys.disable_auth_method('github')
```

Read Status

hvac.api.system_backend.Init.read_init_status()

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

read_response = client.sys.read_init_status()
print('Vault initialize status: %s' % read_response['initialized'])
```

Is Initialized

hvac.api.system_backend.Init.is_initialized()

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

print('Vault initialize status: %s' % client.sys.is_initialized())
```

Initialize

hvac.api.system_backend.Init.initialize()

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

init_result = client.sys.initialize()

root_token = init_result['root_token']
unseal_keys = init_result['keys']
```

2.3.5 Key

Read Root Generation Progress

hvac.api.system_backend.Key.read_root_generation_progress()

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

root_gen_progress = client.sys.read_root_generation_progress()
print('Root generation "started" status: %s' % root_gen_progress['started'])
```

Start Root Token Generation

hvac.api.system_backend.Key.start_root_token_generation()

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

    new_otp = 'RSMGkAqBH5WnVLrDTbZ+UQ=='
    start_generate_root_response = client.sys.start_root_token_generation(
    otp=new_otp,
)
    print('Nonce for root generation is: %s' % start_generate_root_response['nonce'])
```

Cancel Root Generation

hvac.api.system_backend.Key.cancel_root_generation()

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

client.sys.cancel_root_generation()
```

Generate Root

hvac.api.system_backend.Key.generate_root()

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

client.sys.generate_root(
        key=key,
```

(continues on next page)

(continued from previous page)

```
        nonce=nonce,
)
```

Get Encryption Key Status

hvac.api.system_backend.Key.get_encryption_key_status()

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

print('Encryption key term is: %s' % client.sys.key_status['term'])
```

Rotate Encryption Key

hvac.api.system_backend.Key.rotate_encryption_key()

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

client.sys.rotate_encryption_key()
```

Read Rekey Progress

hvac.api.system_backend.Key.read_rekey_progress()

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

print('Rekey "started" status is: %s' % client.sys.read_rekey_progress()['started'])
```

Start Rekey

hvac.api.system_backend.Key.start_rekey()

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

client.sys.start_rekey()
```

Cancel Rekey

hvac.api.system_backend.Key.cancel_rekey()

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

client.sys.cancel_rekey()
```

Rekey

hvac.api.system_backend.Key.rekey()

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

client.sys.rekey(
        key=key,
        nonce=nonce,
        recovery_key=recovery_key,
)
```

Rekey Multi

hvac.api.system_backend.Key.rekey_multi()

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

client.sys.rekey_multi(keys, nonce=nonce)
```

Read Backup Keys

hvac.api.system_backend.Key.read_backup_keys()

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

print('Backup keys are: %s' % client.sys.read_backup_keys()['keys'])
```

2.3.6 Leader

Read Leader Status

hvac.api.system_backend.Leader.read_leader_status()

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

status = self.client.sys.read_leader_status()
print('HA status is: %s' % status['ha_enabled'])
```

2.3.7 Lease

View and Manage Leases

Read a lease:

New in version 0.6.2.

```
>>> client.read_lease(lease_id='pki/issue/my-role/d05138a2-edeb-889d-db98-2057ecd5138f
↪')
{'lease_id': '', 'warnings': None, 'wrap_info': None, 'auth': None, 'lease_duration':␣
↪0, 'request_id': 'a08768dc-b14e-5e2d-f291-4702056f8d4e', 'data': {'last_renewal':␣
↪None, 'ttl': 259145, 'expire_time': '2018-07-19T06:20:02.000046424-05:00', 'id':
↪'pki/issue/my-role/d05138a2-edeb-889d-db98-2057ecd5138f', 'renewable': False,
↪'issue_time': '2018-07-16T06:20:02.918474523-05:00'}, 'renewable': False}
```

Renewing a lease:

```
>>> client.sys.renew_lease(lease_id='pki/issue/my-role/d05138a2-edeb-889d-db98-
↪2057ecd5138f')
{'lease_id': 'pki/issue/my-role/d05138a2-edeb-889d-db98-2057ecd5138f', 'lease_duration
↪': 2764790, 'renewable': True}
```

Revoking a lease:

```
>>> client.sys.revoke_lease(lease_id='pki/issue/my-role/d05138a2-edeb-889d-db98-
↪2057ecd5138f')
```

Read Lease

hvac.api.system_backend.Lease.read_lease()

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

read_lease_resp = client.sys.read_lease(
        lease_id=lease_id,
)

print('Current expire time for lease ID {id} is: {expires}'.format(
        id=lease_id,
        expires=read_lease_resp['data']['expire_time'],
)
```

List Leases

hvac.api.system_backend.Lease.list_leases()

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

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'])
```

Renew Lease

hvac.api.system_backend.Lease.renew_lease()

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

client.sys.renew_lease(
        lease_id=lease_id,
        increment=500,
)
```

Revoke Lease

hvac.api.system_backend.Lease.revoke_lease()

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

client.sys.revoke_lease(
        lease_id=lease_id,
)
```

Revoke Prefix

hvac.api.system_backend.Lease.revoke_prefix()

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

client.sys.revoke_prefix(
        prefix='pki',
)
```

Revoke Force

hvac.api.system_backend.Lease.revoke_force()

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

client.sys.revoke_force(
        lease_id=lease_id,
)
```

2.3.8 Mount

Manipulate secret backends

```
backends = client.sys.list_secret_backends()['data']

client.sys.enable_secrets_engine('aws', path='aws-us-east-1')
client.sys.disable_secrets_engine('mysql')
```

(continues on next page)

(continued from previous page)

```
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

hvac.api.system_backend.Mount.list_mounted_secrets_engines()

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

secrets_engines_list = client.sys.list_mounted_secrets_engines()['data']
print('The following secrets engines are mounted: %s' % secret_engines_list.keys())
```

Enable Secrets Engine

hvac.api.system_backend.Mount.enable_secrets_engine()

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

client.sys.enable_secrets_engine(
        backend_type='github',
        path='hvac-github',
)
```

Disable Secrets Engine

hvac.api.system_backend.Mount.disable_secrets_engine()

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

client.sys.disable_secrets_engine(
        path='hvac-github',
)
```

Read Mount Configuration

hvac.api.system_backend.Mount.read_mount_configuration()

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

secret_backend_tuning = client.sys.read_mount_configuration(path='hvac-github')
print('The max lease TTL for the "hvac-github" backend is: {max_lease_ttl}'.format(
        max_lease_ttl=secret_backend_tuning['data']['max_lease_ttl'],
 ))
```

Tune Mount Configuration

hvac.api.system_backend.Mount.tune_mount_configuration()

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

client.sys.tune_mount_configuration(
        path='hvac-github',
        default_lease_ttl='3600s',
        max_lease_ttl='8600s',
)
```

Move Backend

hvac.api.system_backend.Mount.move_backend()

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

client.sys.move_backend(
        from_path='hvac-github',
        to_path='github-hvac',
)
```

2.3.9 Policy

Manipulate policies

```
policies = client.sys.list_policies()['data']['policies'] # => ['root']

policy = """
path "sys" {
  policy = "deny"
}

path "secret" {
  policy = "write"
}

path "secret/foo" {
  policy = "read"
}
"""

client.sys.create_or_update_policy(
        name='secret-writer',
        policy=policy,
)

client.sys.delete_policy('oldthing')

policy = client.sys.get_policy('mypolicy')
```

(continues on next page)

(continued from previous page)

```
# Requires pyhcl to automatically parse HCL into a Python dictionary
policy = client.sys.get_policy('mypolicy', parse=True)
```

Using Python Variable(s) In Policy Rules

```
import hvac

client = hvac.Client()

key = 'some-key-string'

policy_body = """
path "transit/encrypt/%s" {
        capabilities = "update"
}
""" % key
client.sys.create_or_update_policy(name='my-policy-name', rules=policy_body)
```

List Policies

hvac.api.system_backend.Policy.list_policies()

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

list_policies_resp = client.sys.list_policies()['data']['policies']
print('List of currently configured policies: %s' % list_policies_resp)
```

Read Policy

hvac.api.system_backend.Policy.read_policy()

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

hvac_policy_rules = client.sys.read_policy(name='hvac-policy')['data']['rules']
print('Rules for the hvac policy are: %s' % hvac_policy_rules)
```

Create Or Update Policy

hvac.api.system_backend.Policy.create_or_update_policy()

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

policy = '''
        path "sys" {
                policy = "deny"
        }
        path "secret" {
```

(continues on next page)

(continued from previous page)

```
                policy = "write"
        }
'''
client.sys.create_or_update_policy(
        name='secret-writer',
        policy=policy,
)
```

Delete Policy

hvac.api.system_backend.Policy.delete_policy()

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

client.sys.delete_policy(
        name='secret-writer',
)
```

2.3.10 Seal

Seal Status

hvac.api.system_backend.Seal.seal_status()

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

print('Is Vault sealed: %s' % client.sys.seal_status['sealed'])
```

Is Sealed

hvac.api.system_backend.Seal.is_sealed()

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

print('Is Vault sealed: %s' % client.sys.is_sealed())
```

Read Seal Status

hvac.api.system_backend.Seal.read_status()

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

print('Is Vault sealed: %s' % client.sys.read_seal_status()['sealed'])
```

Seal

hvac.api.system_backend.Seal.read_status()

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

client.sys.seal()
```

Submit Unseal Key

hvac.api.system_backend.Seal.submit_unseal_key()

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

client.sys.submit_unseal_key(key=key)
```

Submit Unseal Keys

hvac.api.system_backend.Seal.read_status()

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

client.sys.submit_unseal_keys(keys=keys)
```

2.3.11 Wrapping

Unwrap

hvac.api.system_backend.Wrapping.unwrap()

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

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
```

(continues on next page)

(continued from previous page)

```
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 3

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_CA_BUNDLE* environment variable.

As documented in the advanced usage section for requests this environment variable should 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
$ export REQUESTS_CA_BUNDLE=/tmp/bundle.pem
```

Alternative, this envrionmental variable can be set via the *os* module in-line with other Python statements. The following example would be one way to manage this configuration on a Ubuntu host:

```
import os

import hvac


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...')
        if certs:
```

(continues on next page)

(continued from previous page)

```
                # 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.
                os.environ['REQUESTS_CA_BUNDLE'] = '/etc/ssl/certs/ca-certificates.crt
↪'

        vault_client = hvac.Client(
                url=vault_url,
                cert=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 *Request* 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 an instance of this custom class to the adapter argument of the *Client constructor*

CHAPTER 4

Source Reference

4.1 hvac.v1

class hvac.v1.**Client**(*url=u'http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *adapter=None*, *namespace=None*)

Bases: object

The hvac Client class for HashiCorp's Vault.

__init__(*url=u'http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *adapter=None*, *namespace=None*)

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.Request
- **namespace** (*str*) – Optional Vault Namespace.

adapter

allow_redirects

audit_hash(***kwargs*)

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

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=u'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=u'approle'*, *use_token=True*)
POST /auth/<mount_point>/login

Parameters

- **role_id** –
- **secret_id** –
- **mount_point** –
- **use_token** –

Returns

Return type

auth_aws_iam(*access_key*, *secret_key*, *session_token=None*, *header_value=None*, *mount_point=u'aws'*, *role=u''*, *use_token=True*, *region=u'us-east-1'*)

POST /auth/<mount point>/login

Parameters

- **access_key** (*str*) – AWS IAM access key ID
- **secret_key** (*str*) – AWS IAM secret access key
- **session_token** (*str*) – Optional AWS IAM session token retrieved via a GetSessionToken AWS API request. see: https://docs.aws.amazon.com/STS/latest/APIReference/API_GetSessionToken.html
- **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.
- **mount_point** (*str*) – The "path" the AWS auth backend was mounted on. Vault currently defaults to "aws". "aws-ec2" is the default argument for backwards comparability within this module.
- **role** (*str*) – Name of the role against which the login is being attempted. If role is not specified, then the login endpoint looks for a role bearing the name of the AMI ID of the EC2 instance that is trying to login if using the ec2 auth method, or the "friendly name" (i.e., role name or username) of the IAM principal authenticated. If a matching role is not found, login fails.
- **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.

Returns The response from the AWS IAM login request attempt.

Return type requests.Response

auth_cubbyhole(*token*)

POST /v1/sys/wrapping/unwrap

Parameters **token** –

Returns

Return type

auth_ec2(*pkcs7*, *nonce=None*, *role=None*, *use_token=True*, *mount_point=u'aws-ec2'*)

POST /auth/<mount point>/login

Parameters

- **pkcs7** (*str.*) – PKCS#7 version of an AWS Instance Identity Document from the EC2 Metadata Service.
- **nonce** (*str.*) – Optional nonce returned as part of the original authentication request. Not required if the backend has "allow_instance_migration" or "disallow_reauthentication" options turned on.
- **role** (*str.*) – Identifier for the AWS auth backend role being requested.
- **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 AWS auth backend was mounted on. Vault currently defaults to "aws". "aws-ec2" is the default argument for backwards comparability within this module.

Returns parsed JSON response from the auth POST request

Return type dict.

auth_gcp(***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(***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=u'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(***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=u'cert'*, *use_token=True*)
POST /auth/<mount point>/login

Parameters

- **mount_point** –
- **use_token** –

Returns

Return type

auth_userpass(*username*, *password*, *mount_point=u'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(***kwargs*)

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(***kwargs*)

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(***kwargs*)

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=u'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=u'aws-ec2'*)

POST /auth/<mount_point>/role/<role>

Parameters

- **role** –
- **bound_ami_id** –
- **bound_account_id** –
- **bound_iam_role_arn** –
- **bound_iam_instance_profile_arn** –
- **bound_ec2_instance_id** –
- **bound_region** –
- **bound_vpc_id** –
- **bound_subnet_id** –
- **role_tag** –
- **ttl** –
- **max_ttl** –
- **period** –
- **policies** –
- **allow_instance_migration** –
- **disallow_reauthentication** –
- **resolve_aws_unique_ids** –
- **mount_point** –

Returns

Return type

create_ec2_role_tag(*role*, *policies=None*, *max_ttl=None*, *instance_id=None*, *disallow_reauthentication=False*, *allow_instance_migration=False*, *mount_point=u'aws-ec2'*)

POST /auth/<mount_point>/role/<role>/tag

Parameters

- **role** –
- **policies** –
- **max_ttl** –
- **instance_id** –
- **disallow_reauthentication** –
- **allow_instance_migration** –

- **mount_point** –

Returns

Return type

create_kubernetes_configuration(*kubernetes_host*, *kubernetes_ca_cert=None*, *token_reviewer_jwt=None*, *pem_keys=None*, *mount_point=u'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=u''*, *max_ttl=u''*, *period=u''*, *policies=None*, *mount_point=u'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=u'approle'*, ***kwargs*)
POST /auth/<mount_point>/role/<role name>

Parameters

- **role_name** –
- **mount_point** –
- **kwargs** –

Returns

Return type

create_role_custom_secret_id(*role_name*, *secret_id*, *meta=None*, *mount_point=u'approle'*)
POST /auth/<mount_point>/role/<role name>/custom-secret-id

Parameters

- **role_name** –
- **secret_id** –
- **meta** –
- **mount_point** –

Returns

Return type

create_role_secret_id(*role_name*, *meta=None*, *cidr_list=None*, *wrap_ttl=None*, *mount_point=u'approle'*)
POST /auth/<mount_point>/role/<role name>/secret-id

Parameters

- **role_name** –
- **meta** –
- **cidr_list** –
- **wrap_ttl** –
- **mount_point** –

Returns

Return type

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*)
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** –

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=u'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=u'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=u'aws-ec2'*)

POST /auth/<mount_point>/config/certificate/<cert_name>

Parameters

- **cert_name** –
- **aws_public_cert** –
- **mount_point** –

Returns

Return type

create_vault_ec2_client_configuration(*access_key*, *secret_key*, *endpoint=None*, *mount_point=u'aws-ec2'*)

POST /auth/<mount_point>/config/client

Configure the credentials required to perform API calls to AWS as well as custom endpoints to talk to AWS APIs. 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

Parameters

- **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.
- **mount_point** (*str|unicode*) – The "path" the AWS auth backend was mounted on. Vault currently defaults to "aws". "aws-ec2" is the default argument for backwards comparability within this module.

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=u'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=u'aws-ec2'*)
DELETE /auth/<mount_point>/role/<role>

Parameters

- **role** –
- **mount_point** –

Returns

Return type

delete_kubernetes_role(*role*, *mount_point=u'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(***kwargs*)

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=u'approle'*)
DELETE /auth/<mount_point>/role/<role name>

Parameters

- **role_name** –
- **mount_point** –

Returns

Return type

delete_role_secret_id(*role_name*, *secret_id*, *mount_point=u'approle'*)
POST /auth/<mount_point>/role/<role name>/secret-id/destroy

Parameters

- **role_name** –
- **secret_id** –
- **mount_point** –

Returns

Return type

delete_role_secret_id_accessor(*role_name*, *secret_id_accessor*, *mount_point=u'approle'*)
DELETE /auth/<mount_point>/role/<role name>/secret-id/<secret_id_accessor>

Parameters

- **role_name** –
- **secret_id_accessor** –
- **mount_point** –

Returns

Return type

delete_token_role(*role*)
Deletes the named token role.

Parameters **role** –

Returns

Return type

delete_user_id(*user_id*, *mount_point=u'app-id'*)
DELETE /auth/<mount_point>/map/user-id/<user_id>

Parameters

- **user_id** –

- **mount_point** –

Returns

Return type

delete_userpass(*username*, *mount_point=u'userpass'*)
DELETE /auth/<mount point>/users/<username>

Parameters

- **username** –
- **mount_point** –

Returns

Return type

delete_vault_ec2_client_configuration(*mount_point=u'aws-ec2'*)
DELETE /auth/<mount_point>/config/client

Parameters **mount_point** –

Returns

Return type

disable_audit_backend(***kwargs*)

Call to deprecated function 'disable_audit_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 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(***kwargs*)

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(***kwargs*)

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(***kwargs*)

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.

Returns The response of the request.

Return type requests.Response

enable_auth_backend(***kwargs*)

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.

Returns The response of the request.

Return type requests.Response

enable_secret_backend(***kwargs*)

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 "method_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.

Returns The response of the request.

Return type requests.Response

generate_root(***kwargs*)

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

generate_root_status

get_app_id(*app_id*, *mount_point=u'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(***kwargs*)

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(***kwargs*)

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=u'aws-ec2'*)
GET /auth/<mount_point>/role/<role>

Parameters

- **role** –
- **mount_point** –

Returns

Return type

get_kubernetes_configuration(*mount_point=u'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=u'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=u'approle'*)
GET /auth/<mount_point>/role/<role name>

Parameters

- **role_name** –
- **mount_point** –

Returns

Return type

get_role_id(*role_name*, *mount_point=u'approle'*)
GET /auth/<mount_point>/role/<role name>/role-id

Parameters

- **role_name** –
- **mount_point** –

Returns

Return type

get_role_secret_id(*role_name*, *secret_id*, *mount_point=u'approle'*)
POST /auth/<mount_point>/role/<role name>/secret-id/lookup

Parameters

- **role_name** –
- **secret_id** –
- **mount_point** –

Returns

Return type

get_role_secret_id_accessor(*role_name*, *secret_id_accessor*, *mount_point=u'approle'*)
POST /auth/<mount_point>/role/<role name>/secret-id-accessor/lookup

Parameters

- **role_name** –
- **secret_id_accessor** –
- **mount_point** –

Returns

Return type

get_secret_backend_tuning(***kwargs*)

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=u'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=u'aws-ec2'*)
GET /auth/<mount_point>/config/certificate/<cert_name>

Parameters

- **cert_name** –
- **mount_point** –

Returns

Return type

get_vault_ec2_client_configuration(*mount_point=u'aws-ec2'*)
GET /auth/<mount_point>/config/client

Parameters **mount_point** –

Returns

Return type

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(***kwargs*)

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(***kwargs*)

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(***kwargs*)

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

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(***kwargs*)

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(***kwargs*)

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=u'aws-ec2'*)
GET /auth/<mount_point>/roles?list=true

Parameters **mount_point** –

Returns

Return type

list_kubernetes_roles(*mount_point=u'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(***kwargs*)

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=u'approle'*)
LIST /auth/<mount_point>/role/<role name>/secret-id

Parameters

- **role_name** (*str|unicode*) – Name of the AppRole.
- **mount_point** (*str|unicode*) – The "path" the AppRole auth backend was mounted on. Vault currently defaults to "approle".

Returns The JSON response of the request.

Return type dict

list_roles(*mount_point=u'approle'*)
GET /auth/<mount_point>/role

Parameters **mount_point** –

Returns

Return type

list_secret_backends(***kwargs*)

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=u'userpass'*)
GET /auth/<mount point>/users?list=true

Parameters **mount_point** –

Returns

Return type

list_vault_ec2_certificate_configurations(*mount_point=u'aws-ec2'*)
GET /auth/<mount_point>/config/certificates?list=true

Parameters **mount_point** –

Returns

Return type

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(***kwargs*)

Call to deprecated function 'read_lease'. This method will be removed in version '0.9.0' Please use the 'read_lease' m 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=u'userpass'*)
GET /auth/<mount point>/users/<username>

Parameters

- **username** –
- **mount_point** –

Returns

Return type

rekey(***kwargs*)

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(***kwargs*)

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

rekey_status

remount_secret_backend(***kwargs*)

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(***kwargs*)

Call to deprecated function 'renew_secret'. This method will be removed in version '0.9.0' Please use the 'renew_leas
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_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

revoke_secret(***kwargs*)

Call to deprecated function 'revoke_secret'. This method will be removed in version '0.9.0' Please use the 'revoke_lea
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(***kwargs*)

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(***kwargs*)

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(***kwargs*)

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

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

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*

session

set_policy(***kwargs*)

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.

Returns The response of the request.

Return type requests.Response

set_role_id(*role_name*, *role_id*, *mount_point=u'approle'*)
POST /auth/<mount_point>/role/<role name>/role-id

Parameters

- **role_name** –

- **role_id** –
- **mount_point** –

Returns

Return type

start_generate_root(***kwargs*)

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(***kwargs*)

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

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*

token

token_role(*role*)

Returns the named token role.

Parameters **role** –

Returns

Return type

transit_create_key(***kwargs*)

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)
- **rsa-2048**: RSA with bit size of 2048 (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(***kwargs*)

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 requests.Response

transit_delete_key(***kwargs*)

Call to deprecated function 'transit_delete_key'. This method will be removed in version '0.9.0' Please use the 'delet
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(***kwargs*)

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 requests.Response

transit_export_key(***kwargs*)

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 requests.Response

transit_generate_data_key(***kwargs*)

Call to deprecated function 'transit_generate_data_key'. This method will be removed in version '0.9.0' Please use th
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 requests.Response

transit_generate_hmac(***kwargs*)

Call to deprecated function 'transit_generate_hmac'. This method will be removed in version '0.9.0' Please use the 'g
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 requests.Response

transit_generate_rand_bytes(***kwargs*)

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 requests.Response

transit_hash_data(***kwargs*)

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 requests.Response

transit_list_keys(***kwargs*)

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.

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

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 requests.Response

transit_read_key(***kwargs*)

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 requests.Response

transit_rewrap_data(***kwargs*)

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 requests.Response

transit_rotate_key(***kwargs*)

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(***kwargs*)

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

Returns The JSON response of the request.

Return type requests.Response

transit_update_key(***kwargs*)

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(***kwargs*)

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

Returns The JSON response of the request.

Return type requests.Response

tune_auth_backend(***kwargs*)

Call to deprecated function 'tune_auth_backend'. This method will be removed in version '0.9.0' Please use the 'tune
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.

Returns The response of the request.

Return type requests.Response

tune_secret_backend(***kwargs*)

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.

Returns The response from the request.

Return type request.Response

unseal(***kwargs*)

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(***kwargs*)

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

- **key** (*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(***kwargs*)

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(***kwargs*)

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=u'userpass'*)
POST /auth/<mount point>/users/<username>/password

Parameters

- **username** –
- **password** –

- **mount_point** –

Returns

Return type

update_userpass_policies(*username*, *policies*, *mount_point=u'userpass'*)
POST /auth/<mount point>/users/<username>/policies

Parameters

- **username** –
- **policies** –
- **mount_point** –

Returns

Return type

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.

class hvac.api.**AuthMethods**(*adapter*)
Bases: hvac.api.vault_api_category.VaultApiCategory

Auth Methods.

implemented_classes = [<class 'hvac.api.auth_methods.azure.Azure'>, <class 'hvac.api.a

unimplemented_classes = ['AppId', 'AppRole', 'AliCloud', 'Aws', 'Jwt', 'Kubernetes', '

class hvac.api.**SecretsEngines**(*adapter*)
Bases: hvac.api.vault_api_category.VaultApiCategory

Secrets Engines.

implemented_classes = [<class 'hvac.api.secrets_engines.azure.Azure'>, <class 'hvac.ap

unimplemented_classes = ['Ad', 'AliCloud', 'AWS', 'Azure', 'Consul', 'Database', 'Gcp'

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.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.policy.Policy, hvac.api.system_backend.seal.Seal, hvac.api.system_backend.wrapping.Wrapping

__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.

__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.

__init__(*adapter*)
API Category class constructor.

Parameters **adapter** (hvac.adapters.Adapter) – Instance of *hvac.adapters.Adapter*; used for performing HTTP requests.

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

implemented_classes
List of implemented classes under this category.

Returns List of implemented classes under this category.

Return type List[*hvac.api.VaultApiBase*]

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.

class hvac.api.auth_methods.**AuthMethods**(*adapter*)
Bases: hvac.api.vault_api_category.VaultApiCategory

Auth Methods.

implemented_classes = [<class 'hvac.api.auth_methods.azure.Azure'>, <class 'hvac.api.a

unimplemented_classes = ['AppId', 'AppRole', 'AliCloud', 'Aws', 'Jwt', 'Kubernetes', '

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

configure(*tenant_id*, *resource*, *environment='AzurePublicCloud'*, *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_location=None*, *bound_subscription_ids=None*, *bound_resource_group_names=None*, *bound_scale_sets=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** (*list*) – Policies to be set on tokens issued using 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_location** (*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_group_names** (*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}/roles. 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.**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

configure(*credentials=''*, *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=''*, *max_ttl=''*, *period=''*, *policies=None*, *bound_service_accounts=None*, *max_jwt_exp='15m'*, *allow_gce_inference=True*, *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

configure(*organization*, *base_url=''*, *ttl=''*, *max_ttl=''*, *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.**Ldap**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

LDAP Auth Method (API).

Reference: https://www.vaultproject.io/api/auth/ldap/index.html

configure(*user_dn*, *group_dn*, *url='ldap://127.0.0.1'*, *case_sensitive_names=False*, *starttls=False*, *tls_min_version='tls12'*, *tls_max_version='tls12'*, *insecure_tls=False*, *certificate=None*, *bind_dn=None*, *bind_pass=None*, *user_attr='cn'*, *discover_dn=False*, *deny_null_bind=True*, *upn_domain=None*, *group_filter='(|(memberUid={{.Username}})(member={{.UserDN}})(uniqueMember={{.UserDN}}))'*, *group_attr='cn'*, *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.
- **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.

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

4.4 hvac.api.secrets_engines

Vault secrets engines endpoints

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

configure(*subscription_id*, *tenant_id*, *client_id=''*, *client_secret=''*, *environment='AzurePublicCloud'*, *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=''*, *max_ttl=''*, *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.**Identity**(*adapter*)
Bases: hvac.api.vault_api_base.VaultApiBase

Identity Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/identity/entity.html

create_or_update_entity(*name*, *entity_id=None*, *metadata=None*, *policies=None*, *disabled=False*, *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=False*, *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

- **group_id** (*str | unicode*) – ID of the group. If set, updates the corresponding existing group.
- **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

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

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

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

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.

Return type dict

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.

Return type dict

merge_entities(*from_entity_ids*, *to_entity_id*, *force=False*, *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_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

update_entity(*entity_id*, *name=None*, *metadata=None*, *policies=None*, *disabled=False*, *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=''*, *canonical_id=''*, *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

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

__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']

default_kv_version

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

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

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

configure(*max_versions=10*, *cas_required=None*, *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.

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_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*, *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.
- **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.**Transit**(*adapter*)

Bases: hvac.api.vault_api_base.VaultApiBase

Transit Secrets Engine (API).

Reference: https://www.vaultproject.io/api/secret/transit/index.html

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 requests.Response

create_key(*name*, *convergent_encryption=False*, *derived=False*, *exportable=False*, *allow_plaintext_backup=False*, *key_type='aes256-gcm96'*, *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)
  - **rsa-2048**: RSA with bit size of 2048 (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=''*, *nonce=''*, *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 requests.Response

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=''*, *key_version=0*, *nonce=None*, *batch_input=None*, *type='aes256-gcm96'*, *convergent_encryption=''*, *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 requests.Response

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 requests.Response

generate_data_key(*name*, *key_type*, *context=''*, *nonce=''*, *bits=256*, *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 requests.Response

generate_hmac(*name*, *hash_input*, *key_version=None*, *algorithm='sha2-256'*, *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 requests.Response

generate_random_bytes(*n_bytes=32*, *output_format='base64'*, *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 requests.Response

hash_data(*hash_input*, *algorithm='sha2-256'*, *output_format='hex'*, *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 requests.Response

list_keys(*mount_point='transit'*)
List keys.

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

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 requests.Response

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 requests.Response

restore_key(*backup*, *name=None*, *force=False*, *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=''*, *key_version=None*, *nonce=''*, *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 requests.Response

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='sha2-256'*, *context=''*, *prehashed=False*, *signature_algorithm='pss'*, *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
- **mount_point** (*str | unicode*) – The "path" the method/backend was mounted on.

Returns The JSON response of the request.

Return type requests.Response

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 requests.Response

update_key_configuration(*name*, *min_decryption_version=0*, *min_encryption_version=0*, *deletion_allowed=False*, *exportable=False*, *allow_plaintext_backup=False*, *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='sha2-256'*, *context=''*, *prehashed=False*, *signature_algorithm='pss'*, *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
- **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.**SecretsEngines**(*adapter*)

Bases: hvac.api.vault_api_category.VaultApiCategory

Secrets Engines.

implemented_classes = [<class 'hvac.api.secrets_engines.azure.Azure'>, <class 'hvac.ap

unimplemented_classes = ['Ad', 'AliCloud', 'AWS', 'Azure', 'Consul', 'Database', 'Gcp'

4.5 hvac.api.system_backend

Collection of Vault system backend API endpoint classes.

class hvac.api.system_backend.**Audit**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin

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*)
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.

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

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*)

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.

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=''*, *passthrough_request_headers=None*)
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.

Returns The response of the request.

Return type requests.Response

class hvac.api.system_backend.**Health**(*adapter*)
Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin

.

Reference: https://www.vaultproject.io/api/system/index.html

read_health_status(*standby_ok=False*, *active_code=200*, *standby_code=429*, *dr_secondary_code=472*, *performance_standby_code=473*, *sealed_code=503*, *uninit_code=501*, *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

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

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_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_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

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

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

class hvac.api.system_backend.**Lease**(*adapter*)

Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin

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

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*)
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 "method_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.

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

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*)

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.

Returns The response from the request.

Return type request.Response

class hvac.api.system_backend.**Policy**(*adapter*)
Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin

create_or_update_policy(*name*, *policy*)
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.

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.**Seal**(*adapter*)
Bases: hvac.api.system_backend.system_backend_mixin.SystemBackendMixin

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

- **key** (*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.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.policy.Policy, hvac.api.system_backend.seal.Seal, hvac.api.system_backend.wrapping.Wrapping

__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

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

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.**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.**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**(*status_code*, *message=None*, *errors=None*)
Helper method to raise exceptions based on the status code of a response received back from Vault.

Parameters

- **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.VaultDown | hvac.exceptions.UnexpectedError

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

4.7 hvac.aws_utils

class hvac.aws_utils.**SigV4Auth**(*access_key*, *secret_key*, *session_token=None*, *region='us-east-1'*)
Bases: object

__init__(*access_key*, *secret_key*, *session_token=None*, *region='us-east-1'*)
x.__init__(...) initializes x; see help(type(x)) for 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

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*)

Bases: object

Abstract base class used when constructing adapters for use with the Client class.

__init__(*base_uri='http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *namespace=None*)

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.

auth(***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

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

request(*method*, *url*, *headers=None*, ***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.

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.**Request**(*base_uri='http://localhost:8200'*, *token=None*, *cert=None*, *verify=True*, *timeout=30*, *proxies=None*, *allow_redirects=True*, *session=None*, *namespace=None*)

Bases: *hvac.adapters.Adapter*

The Request adapter class

request(*method*, *url*, *headers=None*, ***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.
- **kwargs** (*dict*) – Additional keyword arguments to include in the requests call.

Returns The response of the request.

Return type requests.Response

4.9 hvac.exceptions

exception hvac.exceptions.**Forbidden**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**InternalServerError**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**InvalidPath**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**InvalidRequest**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**ParamValidationError**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**RateLimitExceeded**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**Unauthorized**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**UnexpectedError**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**VaultDown**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

exception hvac.exceptions.**VaultError**(*message=None*, *errors=None*)

Bases: exceptions.Exception

__init__(*message=None*, *errors=None*)

x.__init__(...) initializes x; see help(type(x)) for signature

exception hvac.exceptions.**VaultNotInitialized**(*message=None*, *errors=None*)

Bases: *hvac.exceptions.VaultError*

CHAPTER 5

Contributing

Feel free to open pull requests with additional features or improvements!

5.1 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 `VAULT_BRANCH=release scripts/install-vault-release.sh`
2. Install Tox
3. Run tests: `make test`

5.2 Documentation

5.2.1 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.

5.3 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.

5.4 Package Publishing Checklist

The follow list uses version number `0.6.2`, this string should be updated to match the intended release version. It is based on this document: https://gist.github.com/audreyr/5990987

- [ ] Ensure your working directory is clear by running: .. code-block:: guess

  make distclean

- [ ] Checkout a working branch: .. code-block:: guess

  git checkout -b master_v0-6-2

- [ ] Update CHANGELOG.md with a list of the included changes. Those changes can be reviewed, and their associated GitHub PR number confirmed, via GitHub's pull request diff using the previous version's tag. E.g.: https://github.com/hvac/hvac/compare/v0.6.1...master

- [ ] Commit the changelog changes: .. code-block:: guess

  git add CHANGELOG.md git commit -S -m "Updates for upcoming release 0.6.2"

- [ ] Update version number using bumpversion. This example is for the "patch" version but can also be "minor" or "major" as needed. .. code-block:: guess

  bumpversion patch version

- [ ] Commit the version changes: .. code-block:: guess

  git add version setup.cfg git commit -S -m "Bump patch version to $(cat version)"

- [ ] Install the package again for local development, but with the new version number: .. code-block:: guess

  python setup.py develop

- [ ] Run the tests and verify that they all pass: .. code-block:: guess

  make test

- [ ] Invoke setup.py / setuptools via the "package" Makefile job to create the release version's sdist and wheel artifacts:

```
make package
```

- [ ] Publish the sdist and wheel artifacts to TestPyPI using twine:

```
twine upload --repository-url https://test.pypi.org/legacy/ dist/*.tar.gz dist/*.
↪whl
```

- [ ] Check the TestPyPI project page to make sure that the README, and release notes display properly: https://test.pypi.org/project/hvac/

- [ ] Test that the version is correctly listed and it pip installs (`mktmpenv` is available via the virtualenvwrapper module) using the TestPyPI repository (Note: installation will currently fail due to missing recent releases of `requests` on TestPyPI): .. code-block:: guess

  mktmpenv pip install –no-cache-dir –index-url https://test.pypi.org/simple hvac== <verify releaes version shows up with the correct formatting in the resulting list> pip install –no-cache-dir –index-url https://test.pypi.org/simple hvac==0.6.2 <verify hvac functionality> deactivate

- [ ] Create a **draft** GitHub release using the contents of the new release version's CHANGELOG.md content: https://github.com/hvac/hvac/releases/new
- [ ] Upload the sdist and whl files to the draft GitHub release as attached "binaries".
- [ ] Push up the working branch (`git push`) and open a PR to merge the working branch into master: https://github.com/hvac/hvac/compare/master...master_v0-6-2
- [ ] After merging the working branch into master, tag master with the release version and push that up as well:

```
git checkout master
git pull
git tag "v$(cat version)"
git push "v$(cat version)"
```

- [ ] Publish the sdist and wheel artifacts to PyPI using twine:

```
twine upload dist/*.tar.gz dist/*.whl
```

- [ ] Check the PyPI project page to make sure that the README, and release notes display properly: https://pypi.org/project/hvac/
- [ ] Test that the version is correctly listed and it pip installs (`mktmpenv` is available via the virtualenvwrapper module) using the TestPyPI repository:

```
mktmpenv
pip install --no-cache-dir hvac==
<verify releaes version shows up with the correct formatting in the resulting␣
↪list>
pip install --no-cache-dir hvac==0.6.2
<verify hvac functionality>
deactivate
```

- [ ] Publish the draft release on GitHub: https://github.com/hvac/hvac/releases

CHAPTER 6

Changelog

6.1 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.

6.2 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.

6.3 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.

6.4 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.

6.5 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.

6.6 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.

6.7 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.

6.8 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.

6.9 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!

6.10 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]

6.11 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]

6.12 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

6.13 0.2.14 (June 2nd, 2016)

BUG FIXES:

- Fix improper URL being used when leader redirection occurs [GH-56]

6.14 0.2.13 (May 31st, 2016)

IMPROVEMENTS:

- Add support for Requests sessions [GH-53]

BUG FIXES:

- Properly handle redirects from Vault server [GH-51]

6.15 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]

6.16 0.2.10 (April 8th, 2016)

IMPROVEMENTS:

- Add support for list operation [GH-47]

6.17 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

6.18 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

6.19 0.2.7 (December 16th, 2015)

IMPROVEMENTS:

- Add support for PGP keys when rekeying [GH-28]

BUG FIXES:

- Fixed token metadata parameter [GH-27]

6.20 0.2.6 (October 30th, 2015)

IMPROVEMENTS:

- Add support for `revoke-self`
- Restrict `requests` dependency to modern version

6.21 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

6.22 0.2.4 (July 23rd, 2015)

BUG FIXES:

- Fix write response handling [GH-19]

6.23 0.2.3 (July 18th, 2015)

BUG FIXES

- Fix error handling for next Vault release

IMPROVEMENTS:

- Add support for rekey/rotate APIs

6.24 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`

6.25 0.2.1 (June 3rd, 2015)

BUG FIXES:

- Use arguments passed to `initialize` method

6.26 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

6.27 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

6.28 0.1.0 (May 17th, 2015)

- Initial release

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

P

R

S

T

U

V

W