Secrets Management

This article contains everything you need to know about the new Secrets management feature from Hogwarts.

A new way to map your secrets is now available. Take a look at the “Using secrets in Airflow” section!

Introduction

Purpose

Before this feature was released, users had no standardized secret storage options when developing their analytic on JupyterHub. Furthermore, Airflow users were forced to use *connections & variables* which have to be shared with other users on the platform.

The Secrets management feature offers users the ability to securely store and fetch their secrets when developing and running their analytic within Hogwarts. This feature not only solves the secret storage use-case, but bolsters the overall security of the Hogwarts platform.

How will the secrets be stored?

We secure your secrets in a tightly controlled HashiCorp Vault which runs inside of each Hogwarts cluster. Any actions performed against these Vaults are fully audited and traceable.

Furthermore, the Vault is **only** accessible from within the cluster - meaning you cannot manage your secrets from outside Spellbook, Airflow, or Jupyterhub. Each Vault is unique and does not share secrets across environments (i.e. you may have to copy some secrets from one environment to the other).

In addition, due to how the HashiCorp Vault works, secrets are stored as a map of keys and values. This approach is different to other Secret storage solutions like Azure KeyVault where each secret only has one value.

Comparison:

Saving multiple secrets related to a certain database with Azure KeyVault

- Say we have a *database* which we need to store a *username* and *password :*
  - Secret with name `database_username` and value `<username>`
  - Secret with name `database_password` and value `<password>`

Now let’s replicate the above example with a secret named `database` in Hogwarts' HashiCorp Vault

```
{
  "username": "<username>",
  "password": "<password>"
}
```

As you can see, you can now easily group up related secrets into one!

We highly recommend you group up your Hogwarts secrets whenever they are related. This minimizes startup times for your jobs!

Types of secrets

There exists two types of secrets within Hogwarts: **personal** and **group** secrets.

Personal secrets

These secrets are only accessible to you. No setup is required for this type of secret.

We recommend that you use this type of secret for most of your use-cases as they’re inherently the most secure.

Group secrets

These secrets are only accessible to users of a specific Active Directory group which must be linked through Spellbook. For more information on how to create your Secrets groups, follow the Spellbook guide below.

We recommend that you only use groups secrets if you are comfortable sharing these secrets with any member of the AD group.

Managing secrets

There are two ways you can manage your secrets within Hogwarts:

- Using Spellbook
- Using Python code through Notebooks or scripts

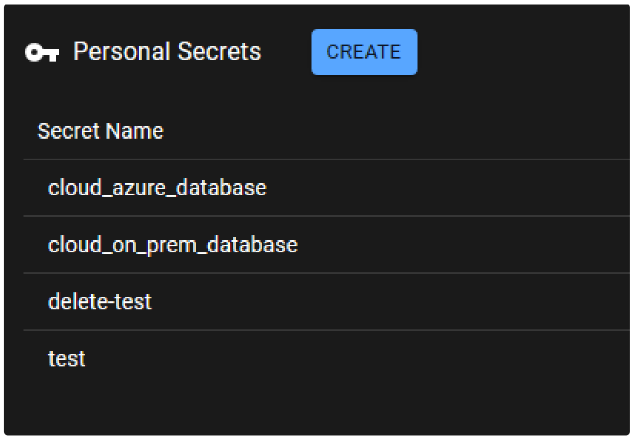
From Spellbook

While in the Spellbook app, click the following left menu icon (key)

Then select either Personal or Group secrets from the sub-menu, whichever you wish to manage.

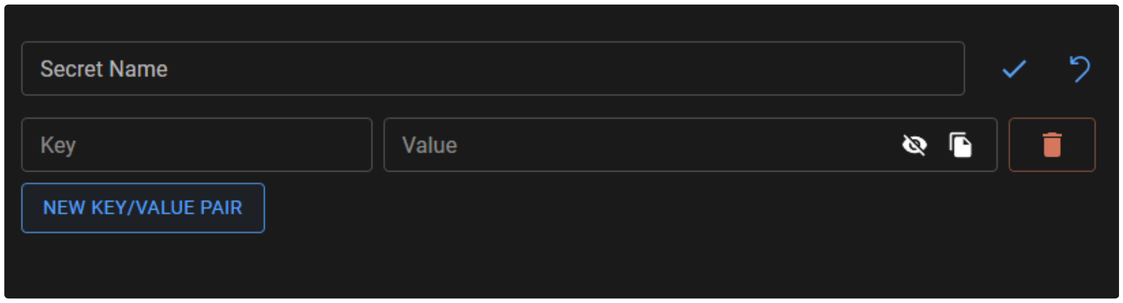
Personal secrets

Once you select "Personal Secrets", you will land on a page which lists your personal secrets.



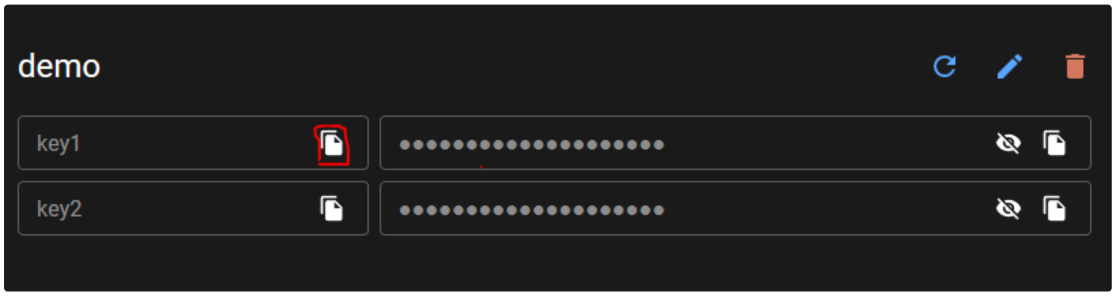
Secret Name
cloud_azure_database
cloud_on_prem_database
delete-test
test

You can create a new Personal secret by clicking the "CREATE" button at the top of the page. You will then be given the form below:



Enter your Secret's name, keys and values. When you are ready to save your Secret, press the "Save changes" button (Checkmark) and confirm your submission.

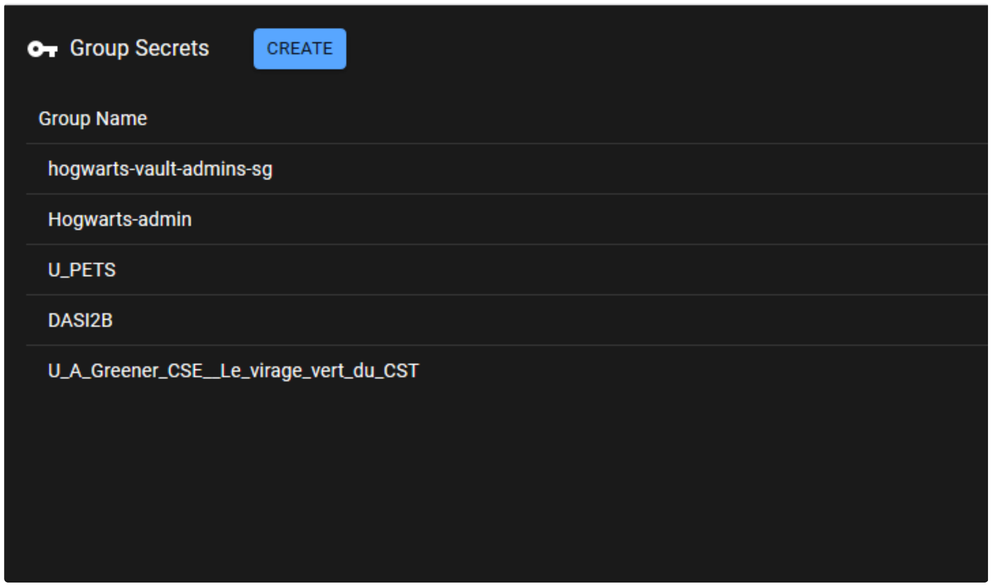
You will then end up on your new Secret's view page:



Pro-tip: Hover the copy icon highlighted above to see how your secret can be accessed through Airflow. You can also press the icon to copy this value.

Group secrets

Once you select “Group Secrets”, you will land on a page which lists all of your Groups.

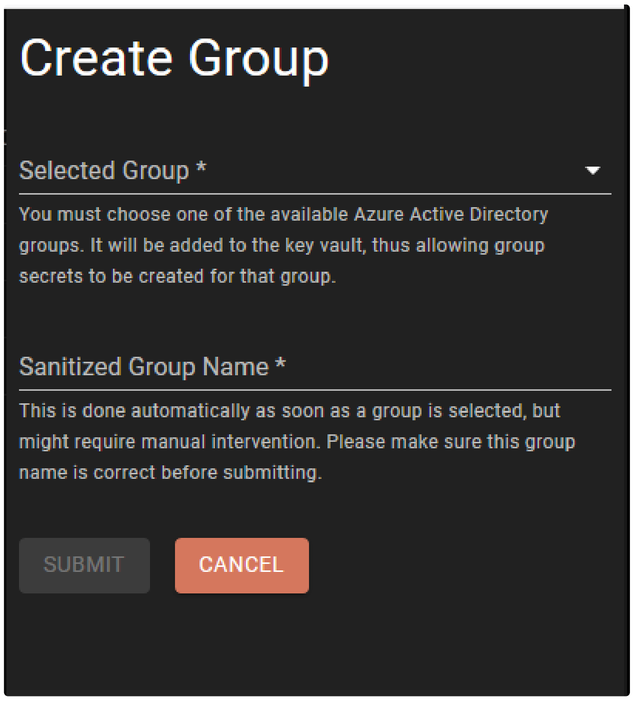


Group Name
hogwarts-vault-admins-sg
Hogwarts-admin
U_PETS
DASI2B
U_A_Greener_CSE__Le_virage_vert_du_CST

If the group you wish to use already exists, you can skip the “Creating your group” section below.

Creating your group

On the page above, click the “CREATE” button. This will open up the following form:


Create Group

Selected Group *

You must choose one of the available Azure Active Directory groups. It will be added to the key vault, thus allowing group secrets to be created for that group.

Sanitized Group Name *

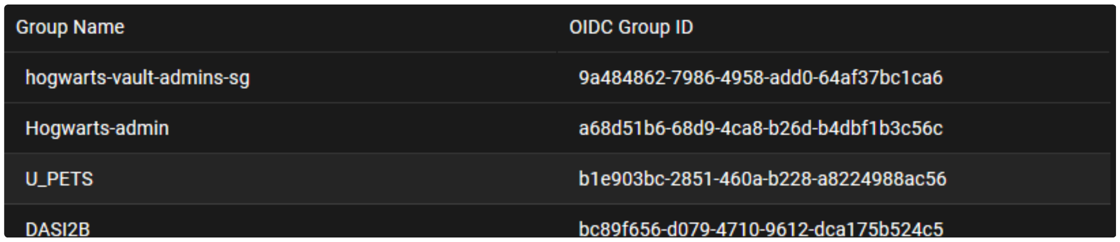
This is done automatically as soon as a group is selected, but might require manual intervention. Please make sure this group name is correct before submitting.

First, select your Azure Active Directory group. Then, modify the name of the group to your liking. Submit the form when completed. Your new group should now show up in the group selection page.

⚠ The name of this group will be used anytime you try to access group secrets. Make sure it is readable and valid.

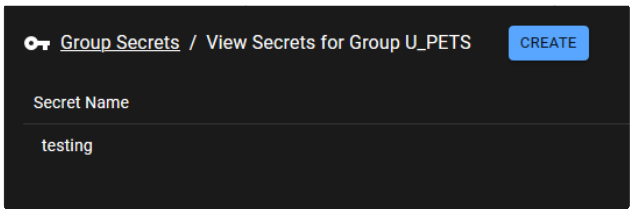
Manage secrets

From the Group list page, select the Group you wish to use.



Group Name	OIDC Group ID
hogwarts-vault-admins-sg	9a484862-7986-4958-add0-64af37bc1ca6
Hogwarts-admin	a68d51b6-68d9-4ca8-b26d-b4dbf1b3c56c
U_PETS	b1e903bc-2851-460a-b228-a8224988ac56
DASI2B	bc89f656-d079-4710-9612-dca175b524c5

The following page will now list all of your Group’s secrets.



Secret Name
testing

Note: This page offers the same functionality as the Personal secrets page except in the context of the selected Group (e.g. U_PETS in this example). Refer to the Personal secrets section for instructions on how to use this page.

From Python code

For users with a preference for code, the Hogwarts Vault library offers an easy-to-use Python library available on Airflow and JupyterHub.

Initialization

Install the library

Note: The library should be installed by default on all Jupyterhub server and Airflow workers.

```
pip install hogwarts-auth
```

Before interacting with the Vault, you must initialize the client and login :

```
from hogwarts.auth.vault.vault_client import VaultClient
vault_client = VaultClient()

# Login to the Vault (session is kept in the vault_client)
vault_client.login()
```

Personal secrets

Creating or updating secrets

```
vault_client.set_personal_secret("database", { 'username': "<your_username>", 'password': '<your_password>'})
```

Fetch a secret

```
vault_client.get_personal_secret("database")
```

List secrets

```
vault_client.list_personal_secrets()
```

Remove a secret

```
vault_client.remove_personal_secret("database")
```

Group secrets

Get groups

```
vault_client.get_groups()
```

Creating or updating group secrets

```
vault_client.set_group_secret('dasi-2b', 'database', { 'username': '<your_username>', 'password': '<your_passwor
```

Fetch a group secret

```
vault_client.get_group_secret('dasi-2b', 'database')
```

List secrets for a group

```
vault_client.list_group_secrets('dasi-2b')
```

Remove a secret

```
vault_client.remove_group_secret('dasi-2b', 'database')
```

For the full code, visit the Hogwarts-example repository.

Using secrets in Airflow

⚠ A job/DAG that is not present in Spellbook **cannot** use secrets. This is because Spellbook links your identity to your job/DAG during the creation or adopting process.

Secrets cannot be pulled at the DAG level at this time. Using the same approach as connections/variables will fail at the DAG validation step.

There are two ways you can retrieve secrets during an Airflow task:

- From Hogwarts supported operators (Daggers)
- From Python code

From Hogwarts supported operators (Daggers)

Daggers operators offer two unique parameters to pull your secrets, as well as two different ways to map your secrets.

Pulling your secrets

In the following examples, we will use a list of strings to fetch our secrets. This is the most basic mapping approach for secrets as it will auto-generate all of the secret's keys as environment variables. For more detailed mapping, please refer to the **Mapping your secrets** section.

Personal secrets

Parameter : `personal_secrets: Union[List[str], List[Secret]]`

Input : List of secret names (strings) or Secret objects (both are *case-sensitive* ⚠)

Description : Each personal secret in the given list will be pulled then mapped as environment variables - depending on the mapping approach used.

Example :

Suppose you set the following parameter `personal_secrets=['database']` in your operator.

If your secret named **database** had the following keys & values:

```
{
  "username": "<username>",
  "password": "<password>"
}
```

The following environment variables would be generated at runtime :

- `DATABASE_USERNAME=<username>`
- `DATABASE_PASSWORD=<password>`

Group secrets

Parameter : `group_secrets: dict`

Input : Dictionary of type `{ str : Union[List[str], List[Secret]] }` (*case-sensitive* ⚠)

Description : For each group and secret specified, the secret will be pulled then mapped as environment variables - depending on the mapping approach used.

Example :

Suppose you are part of a Secrets group called **dasi-2b** and you set the following parameter `group_secrets={'dasi-2b': ['database']}` in your operator.

If your group secret **database** has the following keys & values:

```
{
  "username": "<username>",
  "password": "<password>"
}
```

The following environment variables would be generated at runtime :

- `DASI_2B_DATABASE_USERNAME=<username>`
- `DASI_2B_DATABASE_PASSWORD=<password>`

ℹ Dashes are converted to underscores to support environment variable names.

Full code example(s)

Here is a code snippet of DAG task pulling then printing **all** secrets listed from the above examples:

```python
from daggers.operators.native import HogwartsPythonOperator

(...)
def python_callable():
    import os

    print('Personal environment variables')
    print(f'DATABASE_USERNAME: {os.getenv("DATABASE_USERNAME")}')
    print(f'DATABASE_PASSWORD: {os.getenv("DATABASE_PASSWORD")}')

    print('Group environment variables')
    print(f'DASI_2B_DATABASE_USERNAME: {os.getenv("DASI_2B_DATABASE_USERNAME")}')
    print(f'DASI_2B_DATABASE_PASSWORD: {os.getenv("DASI_2B_DATABASE_PASSWORD")}')

t1 = HogwartsPythonOperator(
    dag=dag,
    task_id='echo_secret_env_vars',
    python_callable=python_callable,
    personal_secrets=['database'],
    group_secrets={
        'dasi-2b': ['database']
    }
)
```

Mapping your secrets

Although auto-generated environment variables are great, they don't offer much flexibility when mapping to specific environment variable names.

With the introduction of `Secret` and `SecretMapper`, you can now specify the exact environment variable names to use when mapping your secrets.

Secret

Represents a secret to pull from the Hogwarts Vault

Parameters:

- name : `str` - The name of the Secret.
- map : `Optional[List[SecretMapper]] = []` - List of SecretMapper detailing how to map your values.
- ignore_missing : `Optional[bool] = False` - Ignore the keys which were not provided in the `map` property. If not provided, any keys missing from the `map` parameter will be auto-generated like usual.

SecretMapper

Maps a Secret key and value to a specific environment variable.

Parameters:

- key : `str` - The secret's key.
- env_var : `str` - The environment variable to map the secret key's value to.

Example(s)

We have taken the example from the previous section and converted it to use Secret+SecretMapper.

```python
from daggers.operators.native import HogwartsPythonOperator
from daggers.secrets import Secret, SecretMapper

(...)
def python_callable():
    import os

    print('Personal environment variables')
    print(f'USERNAME: {os.getenv("USERNAME")}')
    print(f'PASSWORD: {os.getenv("PASSWORD")}')

    print('Group environment variables')
    print(f'USER_GENERATED_ENV_VAR: {os.getenv("USER_GENERATED_ENV_VAR")}')

t1 = HogwartsPythonOperator(
    dag=dag,
    task_id='echo_secret_env_vars',
    python_callable=python_callable,
    personal_secrets=[
      Secret('database', [
        SecretMapper('username', 'USERNAME'),
        SecretMapper('password', 'PASSWORD')
        ]
      )
    ],
    group_secrets={
        'dasi-2b': [
          Secret('database', [
            SecretMapper('password', 'USER_GENERATED_ENV_VAR')], ignore_missing=True)
          ])
        ]
    }
)
```

⚠ Don't forget to import the Secret and SecretMapper from `daggers.secrets`

From Python code

Use the same code from the *Managing secrets* section

Code example:

```python
from hogwarts.auth.vault.vault_client import VaultClient
vault_client = VaultClient()
vault_client.login()

database_secret = vault_client.get_personal_secret('database')
# Code using database_secret['username'] and database_secret['password']

# If you wish to update the secret
from datetime import datetime
database_secret['last_connection'] = str(datetime.now())
vault_client.set_personal_secret('database', database_secret)
```

Alternatives

We are currently looking into other solutions for secret injection (e.g. Airflow macro). Stay tuned!