Delinea

DevOps Secrets Vault

Administrator Guide

Version: 2023.x

Publication Date: 12/11/2024

DevOps Secrets Vault Administrator Guide

Version: 2023.x, Publication Date: 12/11/2024

Table of Contents

Administrator Guide i

DevOps Secrets Vault Overview 1

Key Features 1
Free Version and Quick Start 1
API 2

Quick Links 2

Delinea Links 2
Third-Party Downloads 2

Quick Start 2

Step 1 - Create a DSV Account 3
Video Guide 3
Procedure 3
Step 2 - Download the Command Line Interface (CLI) 7
Video Guide 7
Windows Guide 7
Linux Video Guide 8
Procedure 8
Step 3 - Initialize the CLI 10
Video Guide 11
Procedure 11
Step 4 - Create a Secret 14
Video Guide 14
Procedure 14
Creating Secrets from a File 14
Creating Secrets from Direct Command 16
Retrieve a Secret 17
Filter JSON Command Output for Specific Fields 18
Separately Update Attributes, Data, and Description 18
Step 5 - Create Users 19
Creating Local Users 19
Authenticating the Local User 19
Step 6 - Provide Users Access to Secrets 20
Creating a User Group 20
Creating a Policy to Allow Access 20
Creating a Policy to Deny Access 21

Developer Resources 22

DSV API 22
SDKs 22
Downloads 22
Integrations 22

Delinea In-Product Integrations 23
Delinea In-Product Customization 23
Delinea Created Unpaid Integrations 23
Third Party Integrations to Delinea 23
Third Party Supporting Tools 24
Professional Services Integrations 24
APIs and SDKs 24
Downloads 24

DSV Concepts 24

Architecture and Security 25
Availability 25
Business Continuity and Disaster Recovery 26
Allow List 26
Confidentiality 26
Data at Rest 26
Data in Transit 26
Client Authentication 26
Integrity Checks 27
CLI Code Signing 27
Token Signing 27
Personally Identifiable Information (PII) and GDPR 27
Third Party SOC 2 Conformance Assessment 27
Audit 28
Permissions 28
API Endpoint 28
CLI Command 29
UI View 29
SIEM 29
Available Audit Logs 29
Break Glass 33
Bring Your Own Key (BYOK) Encryption 34
DSV's BYOK Approach 34
Dynamic Secrets 34
Linking 35
Search for linked Secrets 36
Encryption as a Service 36
DSV Engine 36
Organization Firewall 37

Usage 37

CLI Reference 37
CLI Command Syntax 37
Objects 38
Workflows 39

Parameters ... 39
Output Modifiers ... 41
Encoding ... 41
Filter ... 41
Out ... 42
Output Piping ... 42
Secret ... 43
Commands that Act on Secrets ... 43
Examples ... 44
User ... 50
Commands that Act on Users ... 50
Examples ... 51
Group ... 53
Commands that Act on Groups ... 54
Examples ... 54
Role ... 56
Commands that Act on Roles ... 57
Examples ... 57
Client ... 59
Commands that Act on Clients ... 59
Examples ... 59
Policy ... 62
Commands that Act on Policy ... 62
Policy Evaluation ... 63
Policy Examples ... 64
Admin Policy and Auth Providers ... 71
Commands that Act on Configuration ... 71
Audit Command ... 75
Flags ... 75
Usage Examples ... 75
Report Command ... 77
Secret Reporting ... 77
Group Reporting ... 79
Home Vault ... 81
Examples ... 81
DSV UI Reference ... 86
Navigating the UI ... 86
Customizing the UI ... 87
Audit ... 88
Viewing Vaults ... 89
Dashboard ... 90
Secrets ... 90
Viewing Secrets Metadata ... 91
Accessing Audit Details ... 92

Creating and Deleting Secrets 92
Auth Providers 93
Downloading AuthProvider Information 94
Create a New Authentication Provider 94
Users 95
Viewing Users 95
Creating Users 97
Assigning Group Membership 98
Groups 98
Viewing Groups 99
Creating Groups 101
Deleting a Group 101
Roles 102
Viewing Roles 102
Viewing Role Details 103
Creating Roles 104
Deleting a Role 105
Policies 105
Viewing Policies 105
Viewing Policy Details 106
Creating a New Policy 107
Engines and Pools 108
Viewing and Pools 108
Creating a Pool 108
Viewing Pool Details 108
Viewing Attached Engines 109
Creating a New Engine 109
Viewing Engine Details 110
SIEM 111
Viewing SIEM Integrations 111
Creating a SIEM Integration for Auditing 112
Deleting a SIEM Integraion 113
Authentication 113
Password 113
Client Credentials 113
Thycotic One Authentication 114
Third Party Authentication 114
Profiles 115
Add a Profile to a Config 115
See the Config Contents 115
Using an Alternate Profile for a Specific CLI Action 115
Authentication: AWS 115
AWS Authentication Provider 116
AWS User Example 116

AWS Role Example 118
Authentication: Azure 119
Azure Authentication Provider 119
Azure User Assigned MSI Example 120
Azure Resource Group 121
Authentication Google Cloud Platform (GCP) 122
Google Service Account Authentication 122
Google Compute Engine (GCE) Metadata Authentication 128
Google Kubernetes Engine (GKE) Authentication 132
Authentication: OIDC 138
OIDC Providers 139
Common Steps 139
Google Identity Provider Example 141
Azure AD OIDC Example 143
Okta Identity Provider Example 144
Authentication: Certificate 150
Prerequisites 150
CLI Configuration 151
Dynamic Secrets 151
Linking 152
Search for linked Secrets 153
IaaS Dynamic Secrets 153
AWS Dynamic Secrets 153
AAD Graph Dynamic Secrets 159
Microsoft Graph Dynamic Secrets 168
GCP Dynamic Secrets 175
Database Dynamic Secrets 179
DSV Engine Required 180
Microsoft SQL Dynamic Secrets 180
MySQL Dynamic Secrets 182
Oracle Dynamic Secrets 184
Dynamic Secret Examples 186
PostgreSQL Dynamic Secrets 188
MongoDB Dynamic Secrets 191
DSV Engine 193
Starting an Engine 193
Engine Wizard 194
Engine Flags 194
CLI & Engine Program 194
Starting an Engine in a Container 195
Installing the Engine as a Service/Daemon 196
Supported Service Frameworks/Process Managers 196
Installation Commands 196
Installation Steps 197

Encryption as a Service 197
Management Subcommands 197
Operation Subcommands 197
Key Management Subcommands 197
Flags 198
Encrypting Data 199
Automatic Key Creation 199
Manual Key Creation 199
String Encryption 200
File Encryption 200
Key Rotation and Versioning 201
Creating a New Key Version 202
Rotating to an Existing Key Version 202
Manual Key Updating 203
Certificate Issuance 203
Generate a Signing Certificate 204
Register (Import) a Signing Certificate 206
Generate and Sign a Leaf Certificate 207
Sign a Certificate Given a Certificate Signing Request (CSR) 209
SSH Key Issuance 211
Adding an SSH public key to a server 211
Trusting a group of keys signed by a root key 212
Break Glass 213
Commands and Flags 213
Break Glass Setup 214
Trigger Break Glass 214
Bring Your Own Key (BYOK) Configuration 215
Verify Key Changes in Your AWS Account: Assuming CloudTrail is Enabled 215
SIEM Audits 216
Syslog 216
Common Event Format (CEF) 218
JSON 220
Splunk 221

Tutorials 222

Administration and Configuration Tutorials 222
Policy Tutorial 223
Policy Structure 223
Least Privilege Approach 224
Create Users, Groups and Permissions 225
Initialize the New Admin Account 228
Delegate Secret Management Rights to DevOps Team1 229
Test the DevOps Team Permissions to Read Secrets 232
Grant Groups the Ability to Search Entities via List Privileges 234
Test the DevOps Team Permissions to Search Resources 236

Delegate Rights to Manage Policies to a DevOps Team Member ... 236
Test DevOpsUsr1's Permission to Create Policies ... 238
Delegate Rights to "Create Roles" to a DevOps Team Member ... 239
Create DevOpsTeam1's Client Credentials for an Application ... 241
Test the "Read Secret" Permissions of the DevOpsTeam1's Client Credential ... 242
Use DSV With Direnv ... 244
Challenges ... 244
Quick Start on Creating a Secret Like This ... 244
DSV Tweaks ... 245
Limit Scope Of Secret When Possible ... 245
Azure Dynamic Secrets ... 245
Challenge/Scenarios ... 245
Solution ... 245
Benefits ... 246

DSV Integrations ... 246

Kubernetes ... 247
Selecting a Kubernetes plugin ... 247
Kubernetes Sidecar Architecture ... 247
Description of Operations ... 248
Introduction to the Client ... 249
Introduction to the Broker ... 251
Kubernetes Mutating Webhook ... 254
Architecture ... 254
Implementing the Kubernetes Mutating Webhook ... 255
Terraform ... 255
Jenkins ... 255
Usage ... 255
Jenkins Declarative Pipeline ... 257
Pipeline Script ... 257
GitHub ... 259
GitLab ... 259
Azure DevOps ... 260
DSV lookup plugin for Ansible ... 260
Requirements ... 260
Authentication ... 260
Usage ... 260
Permissions ... 261
Example ... 261
Puppet ... 264
Chef ... 264

Release Notes ... 264

DSV Cloud Service: Change Log ... 264

Support 282
Free Licenses 282
Paid Licenses 282
Obtaining a Support PIN 283
Support by Phone 283
Support by Email 283
Support Ticketing 284

DevOps Secrets Vault Overview

Delinea's DevOps Secrets Vault is a high velocity vault that centralizes secrets management, enforces access, and provides automated logging trails. This cloud-based solution is platform agnostic and designed to replace hard-coded credentials in applications, micro-services, DevOps tools, and robotic process automation. This vault ensures IT, DevOps and Security teams the speed and agility needed to stay competitive without sacrificing security.

DevOps Secrets Vault is deployed as an API-as-a Service. Organizations can sign-up and create their first secrets in minutes with no infrastructure to manage or maintain.

Key Features

- Command line interface (CLI) for Windows, Mac, and Linux/Unix
- RESTful Application programming interface (API)
- APIaaS offering infinite scalability, high-speed access, and agility with no infrastructure maintenance
- Automated and searchable logging
- Five-nines availability
- Disaster recovery via multi-region deployment and hot-standby
- Local caching (with the CLI)
- Sandbox tenant available for testing before deployment to production
- Cloud authentication
  - Amazon Web Services (AWS)
  - Microsoft Azure
  - Google Cloud Platform (GCP)
- Developer Resources (SDK, CLI, integrations, plug-ins)
- SOC II Compliance - report available upon request

Free Version and Quick Start

Delinea offers a feature-complete, non-time-limited free version of DevOps Secrets Vault that supports up to 250 Secrets and 20000 API calls a month.

Signing up for the free version is the first step in getting a DevOps Secrets Vault tenant even if you plan to upgrade to a paid plan immediately.

To get started with guided help, go to the Quick Start section.

When you are ready to begin your trial, head here to sign-up for a free tenant: DevOps Secrets Vault Free and download the CLI here: DevOps Secrets Vault CLI

API

This documentation is for general DevOps Secrets Vault Operation and the CLI. If you prefer the API, here is the API documentation

Quick Links

Delinea Links

- DSV Product Home Page
- Delinea Support Portal Login Page (gets PIN for email or phone support)
- DSV CLI Executables Download Page
- DSV API Documentation
- Delinea GitHub Page (SDKs and Plug-ins)

Third-Party Downloads

- jq Library for filtering JSON results
- Linux pass
- Windows Credential Manager
- AWS CLI
- Azure User Assigned MSI

Quick Start

The Quick Start gets you up and running with the DSV application with easy step-by-step instructions that allow you to create a DSV tenant and populate it with secrets.

The following steps are presented, along supporting video clips. All you need is a valid email and client where the application will be installed.

- Step 1 - Sign Up with Delinea for a Tenant
- Step 2 - Download the Command Line Interface (CLI) tool
- Step 3 - Initialize the CLI for the first time
- Step 4 - Create and retrieve your first secret
- Step 5 - Create Users
- Step 6 - Provide access with policies

Step 1 - Create a DSV Account

Video Guide

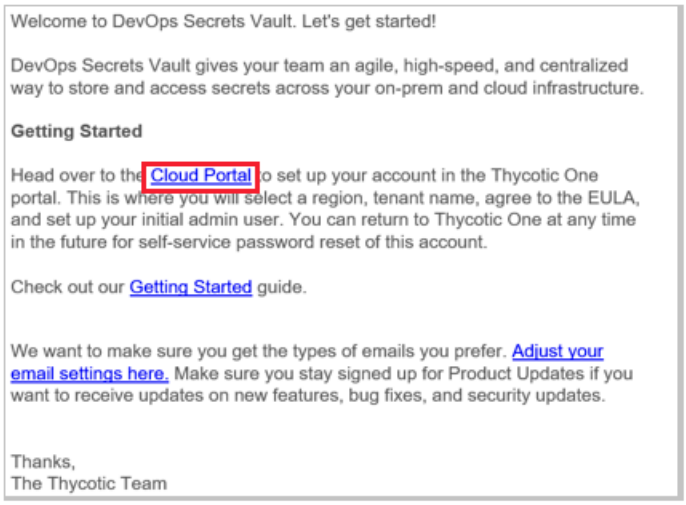
Procedure

Your tenant is your DevOps Secrets Vault cloud account and the rights to access it. Signing up qualifies you for a free feature-complete trial version of DevOps Secrets Vault. The trial version is limited to **250 Secrets** and **2500 API calls** per month. Start by configuring the free version and upgrade when you need more capacity.

To get your tenant:

1. Visit Delinea's DevOps Secrets Vault Home Page. Click **Try It Free** and fill out the DevOps Secrets Vault free web form and submit.
2. After submitting the form, you will receive an email from Delinea Sales, with the subject "DevOps Secrets Vault". Click **Cloud Portal**.

   Note: You can wait until you need support to sign up. If you already have a support account because of a previous Delinea cloud subscription, use your existing account for support. Refer to the Support Services Guide for complete details about our support policy.


Welcome to DevOps Secrets Vault. Let's get started!

DevOps Secrets Vault gives your team an agile, high-speed, and centralized way to store and access secrets across your on-prem and cloud infrastructure.

Getting Started

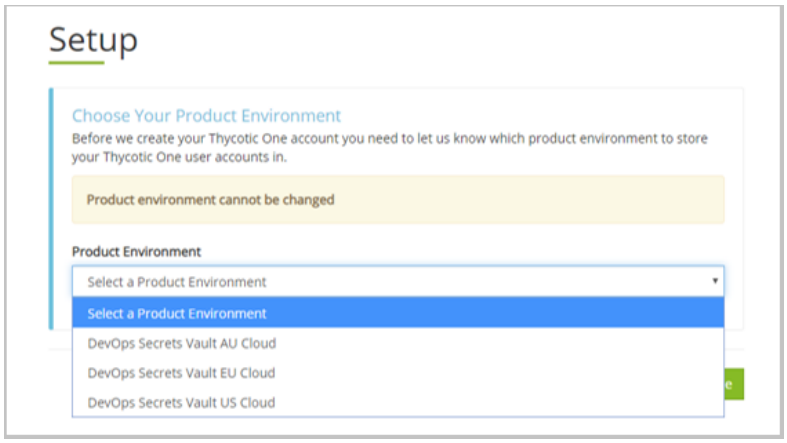
Head over to the Cloud Portal to set up your account in the Thycotic One portal. This is where you will select a region, tenant name, agree to the EULA, and set up your initial admin user. You can return to Thycotic One at any time in the future for self-service password reset of this account.

Check out our Getting Started guide.

We want to make sure you get the types of emails you prefer. Adjust your email settings here. Make sure you stay signed up for Product Updates if you want to receive updates on new features, bug fixes, and security updates.

Thanks,
The Thycotic Team

3. At the Setup dialog, select your Product Environment. The three regions are independent for data sovereignty reasons (like GDPR). All three provide geographical redundancy as follows:
   - secretsvaultcloud.eu (Frankfurt), Active Standby: Ireland
   - secretsvaultcloud.com (US-East), Active Standby: US-West
   - secretsvaultcloud.com.au (Sydney), Active Standby: Singapore

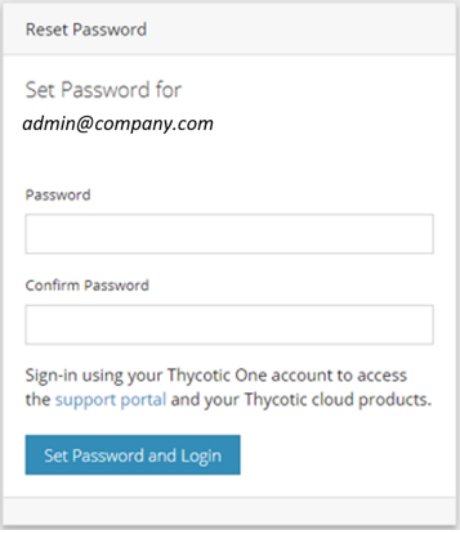


4. Next you are taken to **Thycotic One** to set a password.

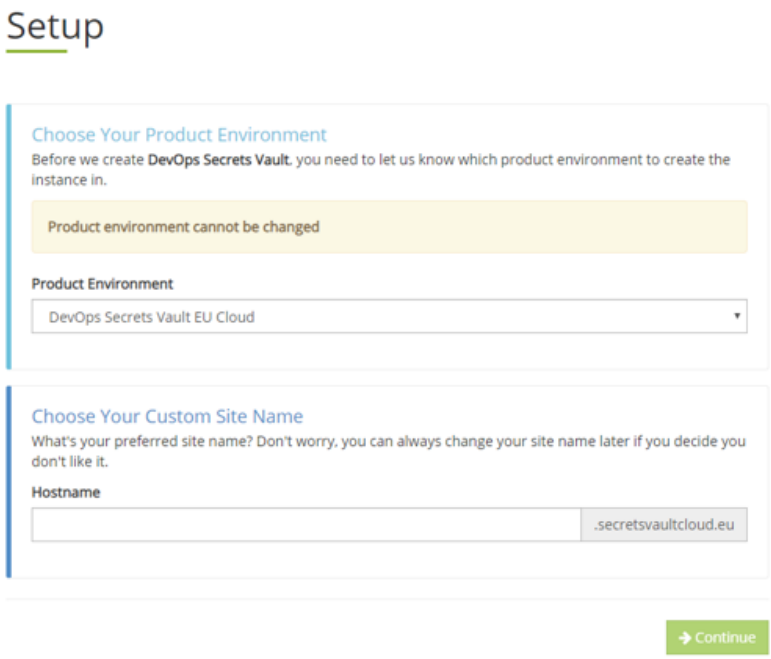
   Note: The person setting up the DevOps Secrets Vault tenant will be considered the *initial administrator* andThycotic One will be established as that person's authentication provider. This is to enable Delinea to help in case the password is lost.

   You can set future users as local or use Thycotic One, AWS, Azure or GCP.

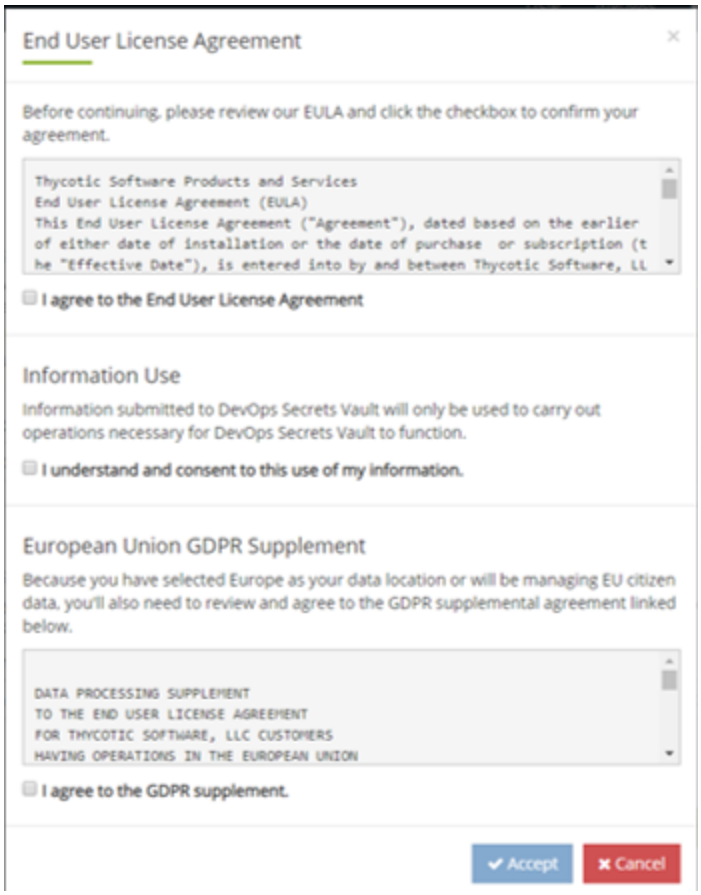
   Later *Thycotic One can be setup later to enable SSO to an identity provider of your choice using OIDC, or enable 2FA when used as the identity provider. The options are TOTP (such as Google Authenticator) and SMS.



5. Next, select your **Product Environment** and supply a **Hostname**. The hostname is your tenant name.



6. Read and agree to the EULA and GDPR (if applicable).



7. The tenant will be created. DSV takes between 5-20 minutes to complete tenant creation. When complete, click **Go To Application**.

   Proceed to Step2 - Download the Command Line Interface (CLI).

Step 2 - Download the Command Line Interface (CLI)

Video Guide

Windows Guide

Linux Video Guide

Procedure

1. Go to the DevOps Secrets Vault Downloads page. Locate the most recent CLI executable for your platform.

   Note: Once installed, periodically check the download site for updates and inform the user if an update is available.

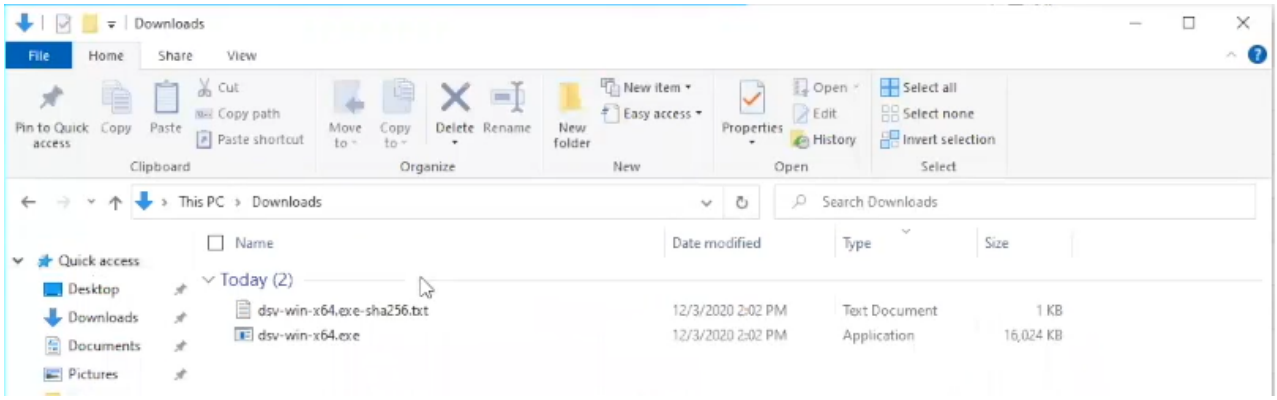
DevOps Secrets Vault Downloads

Name	Date Modified	Size	Type
downloads	Jan 11th 2022, 09:40:13 am	16.61 MB	Folder
cli	Jan 11th 2022, 09:40:13 am	16.61 MB	Folder
1.30.0	Jan 11th 2022, 09:40:13 am	16.61 MB	Folder
dsv-darwin-x64	Jan 11th 2022, 09:40:13 am	16.61 MB	Binary
dsv-darwin-x64-sha256.txt	Jan 11th 2022, 09:40:13 am	81 B	TXT file
dsv-linux-x64	Jan 11th 2022, 09:40:14 am	15.45 MB	Binary
dsv-linux-x64-sha256.txt	Jan 11th 2022, 09:40:14 am	80 B	TXT file
dsv-linux-x86	Jan 11th 2022, 09:40:14 am	13.48 MB	Binary
dsv-linux-x86-sha256.txt	Jan 11th 2022, 09:40:15 am	80 B	TXT file
dsv-win-x64.exe	Jan 11th 2022, 09:40:15 am	15.93 MB	EXE file
dsv-win-x64.exe-sha256.txt	Jan 11th 2022, 09:40:15 am	134 B	TXT file
dsv-win-x86.exe	Jan 11th 2022, 09:40:15 am	14.15 MB	EXE file
dsv-win-x86.exe-sha256.txt	Jan 11th 2022, 09:40:16 am	134 B	TXT file

2. Install the dsv executable onto the workstation.

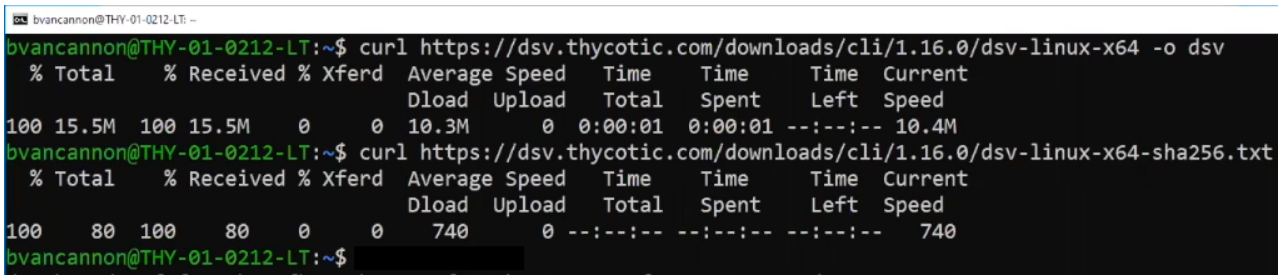
   Windows and macOS - Download the executable file onto each of the workstations that will operate DevOps

Secrets Vault. Locate the executable in your downloads folder. The file name will reflect the OS and 32-bit or 64-bit architecture.



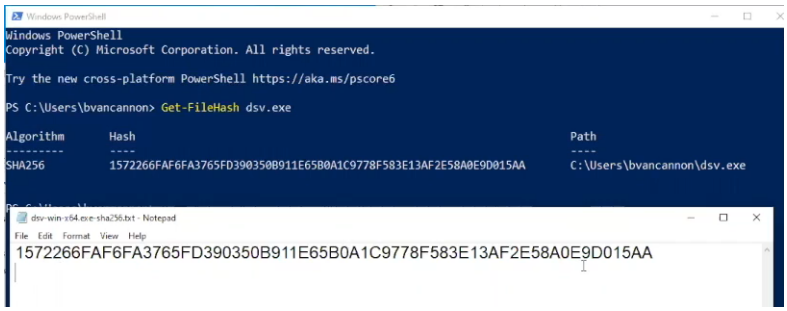
To simplify command entry, Rename the executable to "dsv" (macOS); "dsv" or "dsv.exe" (Windows). Place the executable in the file directory location of your choice and note the path.

Linux - Copy the dsv executable file, open a shell window and use the `curl` command to download the file. Do the same for the hash file.



3. Check the hash for the dsv executable you downloaded, in order to ensure that there are no issues with the file and that the file is not corrupt.

   Windows - In a Command window, type `Get-FileHash dsv.exe` in the directory where your executable resides and observe the Hash code in the response. The Hash code should match the code obtained when you open the executable file (see example).



macOS and Linux - Then, run the `sha256sum` command to check the hash for the dsv executable you downloaded. Next run the `cat` command to catalog the hash file. Ensure the hash codes match to ensure that there are no issues with the file and that the file is not corrupt.

```
bvancannon@THY-01-0212-LT: ~
bvancannon@THY-01-0212-LT:~$ curl https://dsv.thycotic.com/downloads/cli/1.16.0/dsv-linux-x64 -o dsv
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 15.5M  100 15.5M    0     0  10.3M      0  0:00:01  0:00:01 --:--:-- 10.4M
bvancannon@THY-01-0212-LT:~$ curl https://dsv.thycotic.com/downloads/cli/1.16.0/dsv-linux-x64-sha256.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    80  100    80    0     0    740      0 --:--:-- --:--:-- --:--:--   740
bvancannon@THY-01-0212-LT:~$ sha256sum dsv
d047b774d4c4faf493cb2c2fb4719bac771cf252eb696ea31882fc7ae1283c46  dsv
bvancannon@THY-01-0212-LT:~$ cat dsv-hash
d047b774d4c4faf493cb2c2fb4719bac771cf252eb696ea31882fc7ae1283c46  dsv-linux-x64
bvancannon@THY-01-0212-LT:~$
```

4. **Windows and Linux Only** - Change permissions on the dsv executable using the `chmod` command.

```
bvancannon@THY-01-0212-LT:~$ chmod +x dsv
bvancannon@THY-01-0212-LT:~$ ls
dsv  dsv-hash
bvancannon@THY-01-0212-LT:~$
```

5. Add the executable path to the Environment Variable. Adding the location of the executable to your environment variable enables you to invoke `dsv` without specifying its path or having to pre-pend `.\`

   Note: Setting a new path requires a system restart.

   - For Windows, press the Windows key and type *edit environment variables*. At the Environment Variables dialog, locate the System Variables (or under User Variables, if you want make it available only in the context of that user), select **Path**, click **edit**, and add the path to the dsv executable (example: *C:\Users\<name>*). Click **OK**.
   - For macOS, open Terminal. Run `touch ~/.bash_profile; open ~/.bash_profile`. In **TextEdit**, add the path to the dsv executable to `export PATH="` . Save the . bash_profile file.
   - For Linux or macOS, use *export* to modify the shell profile file, *~.profile* or *~.bash_profile* typically, so that it adds dsv to the PATH on system startup: *`export PATH=~thycotic/cli:$PATH`*.

6. Enable Autocomplete. Autocomplete is supported for bash, zsh, and fish shells only. To turn on Autocomplete for the CLI, run `dsv -install` and restart your shell. Now when you type out the beginning of a command such as `dsv s` and hit tab. The command automatically updates to `dsv secret`.

   Autocomplete also helps with expanding the secret path on `dsv secret read`. Put in the beginning of the path, such as `dsv secret read resources` and hit tab to get the next part of the path. If there are multiple matching sub-paths, hit tab twice to print out the available options.

   For example: typing `dsv secret read resources/us-east-` and hitting tab twice will show the output of any secrets below that path, such as *resources/us-east-1/server resources/us-east-2/server*.

   Proceed to Step 3 - Initialize the Command Line Interface (CLI).

Step 3 - Initialize the CLI

DSV CLI initialization presents you with a series of prompts and options. If you are the **initial administrator** who setup the tenant, then you will have the required information from signing-up. If you are not the initial administrator,

you will need the collect this information from that person:

- tenant name
- domain
- local or federated user, and if federated, which authentication provider
- credentials - username or access key, password, or secret key

Video Guide

Procedure

1. Begin setup with the `dsv init` command. This will start a workflow.
2. Enter your tenant name.

   ```
   ? Please enter tenant name:
   ```

   The tenant name was provided to the initial administrator by ## when you set up your account.

   > **Note:** You need only enter your tenant name, i.e., *example* not *example.secretsvaultcloud.com*, because the domain is set by region and that is covered in the next question.

3. Select the domain.

```
? Please choose domain:  [Use arrows to move, type to filter]
                                                > secretsvaultcloud.com
                                                secretsvaultcloud.eu
                                                secretsvaultcloud.com.au
                                                secretsvaultcloud.ca
```

Your domain is based on the server location that was chosen during provisioning: US, EU, AU or CA.

4. Choose a type of credentials and cache storage.

```
? Please select store type:  [Use arrows to move, type to filter]
                                                > File store
                                                None (no caching)
                                                Pass (Linux only)
                                                Windows Credential Manager (Windows only)
```

- Select `File store` to keep the credentials in files. If you select this, DSV prompts for the directory location.
- Select `None (no caching)` to omit storing the credentials. With this option active,DSV requires authentication with every command.
- Select `Pass (Linux only)` to use Linux pass for encrypted storage.
- Select `Windows Credential Manager (Windows only)` to use Windows Credential Manager. to store credentials.

5. Choose a cache strategy for secrets.

```
? Please enter cache strategy for secrets:  [Use arrows to move, type to filter]
                                                > Never
                                                Server then cache
                                                Cache then server
                                                Cache then server, but allow expired cache if server
```

The choice here depends on your organization's security, network connectivity, performance, and systems availability.

> **Note:** `Server` refers to your DSV tenant and `cache` refers to storage on the local machine with the CLI installed.

- Select `Never` to never cache secrets. Every secret read request requires an API call.
- Select `Server then cache` to make an API call every time. If not accessible, then the cached secret is used.
- Select `Cache then server` to use the cached secret unless it has expired, in which case an API call is made.
- Select `Cache then server, but...` to make an API call if the cached secret has expired, but if the API call fails, then the expired cached Secret is used.

6. Select an authentication type.

```
? Please enter auth type:  [Use arrows to move, type to filter]
                                                > Password (local user)
                                                Client Credential
```

```
Thycotic One (federated)
AWS IAM (federated)
Azure (federated)
GCP (federated)
OIDC (federated)
x509 Certificate
```

- Select `Password (local user)` to authenticate by username and password.
- Select `Client Credential` to authenticate by Client ID and Client Secret.
- Select `## (federated)` to authenticate using ##'s access manager.

  Note: The person who signed up for DevOps Secrets Vault is the *initial administrator* and is automatically setup using ##. If this is you, then select this option. This enables you to reset the password if it is ever lost and/or setup up 2FA if desired. It is up to the customer to then decide if all other users are local or federated through one the available providers.

- Select `AWS IAM (federated)` to authenticate as a trusted Identity Access Management Role or User. Refer to AWS Authentication.
- Select `Azure (federated)` to authenticate as a trusted Azure Managed Service Identity (MSI). Refer to Azure Authentication.
- Select `GCP (federated)` to authenticate as a trusted Google Service Account. Refer to GCP Authentication.
- Select `OIDC (federated)` to authenticate through ## to an external IDP using the OIDC protocol. Refer to OIDC Authentication.
- Select `x509 Certificate` to authenticate using certificates. Refer to Certificate Authentication.

7. Complete the authentication.

After initialization was completed, type `$ dsv auth` to obtain and display your access token.

You can now use the CLI to create your first secret in the DevOps Secrets Vault. Refer to Step 4 - Create a Secret.

Step 4 - Create a Secret

Video Guide

Procedure

Two methods for entering secrets are supported: File and Direct Command.

- File - The File method uses a file that contains the attributes for secrets that are uploaded in bulk to a path in your vault, using the CLI.
- Direct Command - The Direct Command method uses the CLI to individually specify the creation of secrets directly into a path in your vault.

> **Note:** After secrets are created, they can be viewed in the "DSV UI Reference" on page 86 in your Home Vault.

Creating Secrets from a File

1. To create a secret, open a text editor and create and save a file (.json) similar to this example. The JSON is arbitrary, so you can set any number of fields (key-value pairs) for a secret. Files may also be used to enter attributes `--attributes` or a description `--desc`

```
{
```

```
  "host": "server01",
  "username": "administrator",
  "password": "secretp@ssword"
}
```

2. Add as many secrets as needed. Save the file and note its location.
3. Issue a `dsv secret create` command and specify the path to its storage location.

Note: Every secret correlates uniquely with a specific path that describes the location of the secret in your Home Vault. The idea here is no different than the concept of a path to a file on a hard drive. Paths are also the basis for creating policies to determine who (or what) has which rights to those secrets in your Home Vault.

Linux:

```
dsv secret create --path servers:us-east:server01 --data @secret.json
```

Powershell:

```
dsv secret create --path servers:us-east:server01 --data '@secret.json'
```

CMD:

```
dsv secret create --path servers:us-east:server01 --data @secret.json
```

Outputs:

```
{
  "attributes": null,
  "created": "2019-01-03T23:11:48Z",
  "createdBy": "users:thy-one:admin@company.com",
  "data": {
    "host": "server01",
    "password": "secretp@sssword",
    "username": "administrator"
  },
  "description": "",
  "id": "c5239a6c-422e-4f57-b3a6-5167656af852",
  "lastModified": "2019-01-03T23:11:48Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "path": "servers:us-east:server01",
  "version": "0"
}
```

Creating Secrets from Direct Command

Instead of using a file, the data can be entered as part of the command. The following options are available:

```
--data -d      JSON object containing the secret data
--attributes   JSON object containing attributes about the secret
--desc         String with description of the secret
--body         JSON object with 1 or more of the above options
--path -r      Target path to a secret (required)
--help         Help with this command
```

Note: If the `--body` option is passed in and any of the other options are also passed in (data, attributes or desc), the body option will be politely ignored.

Linux:

```
dsv secret create --path servers:us-east:server01 --data '
{"host":"server01","username":"administrator","password":"secretp@sssword"}'

dsv secret create --path servers:us-east:server01 --attributes '
{"secretType":"webServer","serverName":"server01","adminLevel":"readOnly"}'

dsv secret create --path servers:us-east:server01 --desc "webserver secret values"

dsv secret create --path servers:us-east:server01 --body '{"data":
{"host":"server01","username":"administrator","password":"secretp@sssword"},
"attributes":
{"secretType":"webServer","serverName":"server01","adminLevel":"readOnly"},"desc":"webserv
er secret values"}'
```

Powershell:

```
dsv secret create --path servers:us-east:server01 --data '
{\"host\":\"server01\",\"username\":\"administrator\",\"password\":\"secretp@sssword\"}'

dsv secret create --path servers:us-east:server01 --attributes '
{\"secretType\":\"webServer\",\"serverName\":\"server01\",\"adminLevel\":\"readOnly\"}'

dsv secret create --path servers:us-east:server01 --desc \"webserver secret values\"

dsv secret create --path servers:us-east:server01 --body '{\"data\":
{\"host\":\"server01\",\"username\":\"administrator\",\"password\":\"secretp@sssword\"},
\"attributes\":
{\"secretType\":\"webServer\",\"serverName\":\"server01\",\"adminLevel\":\"readOnly\"},\"d
esc\":\"webserver secret values\"}'
```

Note: `dsv secret create` can be replaced with `dsv home create`.

Outputs:

```
{
  "attributes": null,
  "created": "2019-01-03T23:11:48Z",
  "createdBy": "users:thy-one:admin@company.com",
  "data": {
    "host": "server01",
    "password": "secretp@sssword",
    "username": "administrator"
  },
  "description": "",
  "id": "c5239a6c-422e-4f57-b3a6-5167656af852",
  "lastModified": "2019-01-03T23:11:48Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "path": "servers:us-east:server01",
  "version": "0"
}
```

Retrieve a Secret

To retrieve a secret use the Secret read command and specify the path to the Secret's storage location.

```
dsv secret read --path /servers/us-east/server01
```

Output defaults to JSON:

```
{
  "attributes": null,
  "created": "2019-11-08T15:46:14Z",
  "createdBy": "users:thy-one:admin@company.com",
  "data": {
    "host": "server01",
    "password": "secretp@ssword",
    "username": "administrator"
  },
  "description": "",
  "id": "c5239a6c-422e-4f57-b3a6-5167656af852",
  "lastModified": "2020-01-17T15:38:49Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "path": "servers:us-east:server01",
  "version": "0"
}
```

If you would like the output to be in YAML:

```
dsv secret read --path /servers/us-east/server01 -e yaml
```

Outputs:

```
attributes: null
created: "2019-11-08T15:46:14Z"
createdBy: users:thy-one:admin@company.com
data:
  host: server01
  password: secretp@ssword
  username: administrator
description: ""
id: c5239a6c-422e-4f57-b3a6-5167656af852
lastModified: "2020-01-17T15:38:49Z"
lastModifiedBy: users:thy-one:admin@company.com
path: servers:us-east:server01
version: "0"
```

Filter JSON Command Output for Specific Fields

When you need to locate a specific field in a JSON output, use a JSON filter. An example use case is writing scripts that need to obtain a password but lack the capacity to efficiently parse JSON.

```
dsv secret read --path /servers/us-east/server01 -f data.password
```

Would return just the password.

Separately Update Attributes, Data, and Description

Using the `--data`, `--attributes`, and `--desc` flags, respectively, you can update a Secret's data, attributes, and description separately. For example:

```
dsv secret update servers/us-east/server01 --data '{"host": "server01", "password":
"badpassword","username": "admin"}' --desc 'update description'  --attributes '{"attr":
"add one"}'
```

```
{
  "attributes": {
    "attr": "add one"
  },
  "created": "2019-11-08T15:46:14Z",
  "createdBy": "users:thy-one:admin@company.com",
  "data": {
    "host": "server01",
    "password": "badpassword",
    "username": "admin"
  },
  "description": "update description",
  "id": "4348e941-f945-460d-98e8-2ab659362f51",
  "lastModified": "2020-02-22T20:48:05Z",
```

```
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "path": "servers:us-east:server01",
  "version": "1"
}
```

Refer to Steps 5 Create Users and 6 Provide User Access to create users, user groups and policies that provide the framework for managing the secrets.

Step 5 - Create Users

With the first Secrets created, the next step is to create Users or Roles that will access those secrets.

For this quick-start guide, as the intial admin, we will create a local User. To use other authentication methods, see authentication.

Note: This procedure steps through creating users with the CLI. Users can also be created, viewed, and managed in the DSV User Interface.

Creating Local Users

Create a user and assign credentials using the following format:

```
dsv user create --username local@company.com --password userpassword
```

Note: For local users, the email address serves only as the username.

Authenticating the Local User

The local user can then, on their own machine, download the CLI and start the `dsv init` process. The admin will have to provide the user with their password, DSV tenant name, and domain (region).

The process is here: Initializing the CLI for the first time

When they get to the **Please enter auth type:**

```
Please enter auth type:
        (1) Password (local user)(default)
        (2) Client Credential
        (3) #{ThycoticOne}# (federated)
        (4) AWS IAM (federated)
        (5) Azure (federated)
        (6) GCP (federated)
        (7) OIDC (federated)
```

The user will select `(1)` and enter their username and password. The user should change their password immediately as a best practice. The command to change the password is:

```
dsv auth change-password
```

At this point, the users are created and able to authenticate to DSV (they can confirm with the command `dsv auth` and get a token), however, they will not have permission to access anything yet because DSV defaults to `deny all`. In the next step, the admin will create policies granting permission to these users.

Step 6 - Provide Users Access to Secrets

Note: This procedure steps through creating user groups and policies with the CLI. User Groups and policies can also be created, viewed, and managed in the DSV User Interface.

- With two secrets, each located at:

  servers:us-east:server01 *and* servers:us-east:production:server01

- And two users:

  developer1@company.com *and* developer2@company.com

You can create a policy to allow:

- both users access to servers:us-east:server01
- developer1@company.com to **have access** to servers:us-east:production:server01
- developer2@company.com to be **denied access** to servers:us-east:production:server01

Creating a User Group

Optionally, we can put these Users in a Group with two commands.

- The first command creates the group:

```
dsv group create --group-name firstgroup
```

- The second command puts the Users in the Group

```
dsv group add-members --group-name firstgroup --data '{"memberNames":
["developer1@company.com","developer2@company.com"]}'
```

Creating a Policy to Allow Access

The admin has to create a policy for the Group to get access to the Secrets. Here is a sample CLI command:

```
dsv policy create --path secrets:servers:us-east --actions '<.*>' --desc 'Allow Policy' --
subjects groups:firstgroup --effect allow
```

- **`path`** starts with **`secrets:`** followed by the secret path.

  Note: `resources` is not specified separately, but will default to the path and everything below it, so in this case `secrets:servers:us-east:<.*>`

- **`actions`** is a wildcard, so full `create, read, update, delete` are allowed.
- **`subjects`** are the Users that are getting access to the secrets.
- **`effect`** will either allow or deny access.
- Use the command **`dsv policy read secrets:servers:us-east -e yaml`** to read the resulting policy:

```
  path: secrets:servers:us-east
  permissionDocument:  - actions:    - <.*>    conditions: {}    description: Allow
Policy    effect: allow    id: xxxxxxxxxxxxxxxxxxxx    meta: null    resources:    -
secrets:servers:us-east:<.*>    subjects:    - groups:firstgroup  version: "0"
```

- This policy will now enable both Users (*developer1@company.com* and *developer2@company.com*) to gain full access to all secrets located at the path `servers:us-east` and below.

Creating a Policy to Deny Access

If we decide that the *developer2@company.com* should no longer have access to the secrets at `servers:us-east:production`, we can write another policy to deny access. The command would look like this:

```
dsv policy create --path secrets:servers:us-east:production --actions '<.*>' --desc 'Deny
Policy' --subjects 'users:<developer2@company.com>' --effect deny
```

Use the command **`dsv policy read secrets:servers:us-east:production -e yaml`** to view the resulting policy:

```
path: secrets:servers:us-east:production
permissionDocument:
- actions:
  - <.*>
  conditions: {}
  description: Deny Policy
  effect: deny
  id: xxxxxxxxxxxxxxxxxxxx
  meta: null
  resources:
  - secrets:servers:us-east:production:<.*>
  subjects:
  - users:<developer2@company.com>
version: "0"
```

Now developer1@company.com has access to everything at `servers:us-east` and below, including `servers:us-east:production`. However, developer2@company.com only has access to the secrets at `servers:us-east` and not at `servers:us-east:production`

This is the end of the quick-start guide, but for more on policies see "CLI Reference" on page 37 in this documentation.

Developer Resources

The following resources and integrations are available for the DevOps Secrets Vault.

Contacting the Integrations Team

Any questions or issues, please reach out to integrations@delinea.com.

DSV API

API Documentation

SDKs

- Python SDK for DSV
- Go SDK for DSV
- Java SDK for DSV

Downloads

DSV CLI Executables Download Page

Integrations

Integrations are supported to the extent of the third-party product procedures documented for those integrations. Please contact the third-party for any customized setup of the integrated product.

> **Note:** As a prerequisite of support for any of Delinea's integrations, the customer must have sufficient access to the Delinea product and all parts of the third-party integration, and must be able to provide Delinea with requested information and meetings to examine in order to progress reported issues.

Access each integrated product folder to learn more about the integration details. These include:

- Kubernetes Sidecar
- Kubernetes Mutating Webhook
- Terraform
- Azure DevOps
- Jenkins
- Puppet
- Ansible
- Chef

- GitHub
- GitLab

Delinea In-Product Integrations

Integrations that are directly built into paid Delinea products will be supported by the Delinea support team and defects will be handled by the Delinea product developers who maintain the Delinea product where the issue occurs.

Delinea In-Product Customization

Many Delinea products can be customized in order to achieve an integration between the Delinea product and third-party systems. If Delinea documents an integration as a supported integration, the integration will be configured as specified in our documentation and is verified at the time of their creation by Delinea to ensure that they work as designed.

Note: Assistance with design, configuration, or troubleshooting of customization designed to interact with third-party systems is not within the scope of what the Delinea Support organization can provide at this time. Delinea does not guarantee that every configuration of third-party systems will work with in product customizations. Assistance with design, configuration or troubleshooting for customization of Delinea products can be worked on as part of a paid engagement with the Professional Services team.

Delinea Created Unpaid Integrations

Unpaid integrations created by Delinea are code or applications that are not sold by Delinea for monetary compensation. They are provided for the use of Delinea customers and in some cases are available to the public.

An example of this type of integration would be the RabbitMQ Helper, migration tools created by Delinea, and code provided on the Delinea GitHub.These integrations were verified at the time of their creation by Delinea to ensure that they work as designed.

Note: Assistance with configuration or troubleshooting of these tools with third-party systems is not within the scope of what the Delinea Support organization can provide at this time. Delinea does not guarantee that every configuration of third-party systems will work with 3rd-Party Integrations. Assistance with the use of these tools, configuration, or troubleshooting for customization of Delinea products can be worked on as part of a paid engagement with the Professional Services team.

Third Party Integrations to Delinea

This category of integration encompasses any code or script which integrates with Delinea usually by API that is written by a third-party vendor. Delinea does not guarantee that third-party code is written correctly or that it respects Delinea product limitations.

For instance, the third-party code may fail to respect Token expiry or issue calls too quickly without waiting for responses and time-outs. Third-party integrations are supported by verifying that the Delinea application is functioning correctly. Delinea does not support, code or maintain third-party code or scripts.

For commercially sold third-party products which have vendor support, Delinea may elect to attend calls. The third-party product must be able to provide a knowledgeable resource and share specifics about how they integrate with

the Delinea application. The goal of such calls would be to advise the third-party vendor about what they need to change to better integrate with Delinea products.

Third Party Supporting Tools

jq Library*for filtering JSON results*

Linux pass

Windows Credential Manager

AWS CLI

Azure User Assigned MSI

Professional Services Integrations

Code or scripts written for or provided to customers as part of Professional Services are not included in the definitions above. Please refer to the terms of the warranty on your Professional Services engagement.

APIs and SDKs

DSV API Documentation

Python SDK for DSV

Go SDK for DSV

Java SDK for DSV

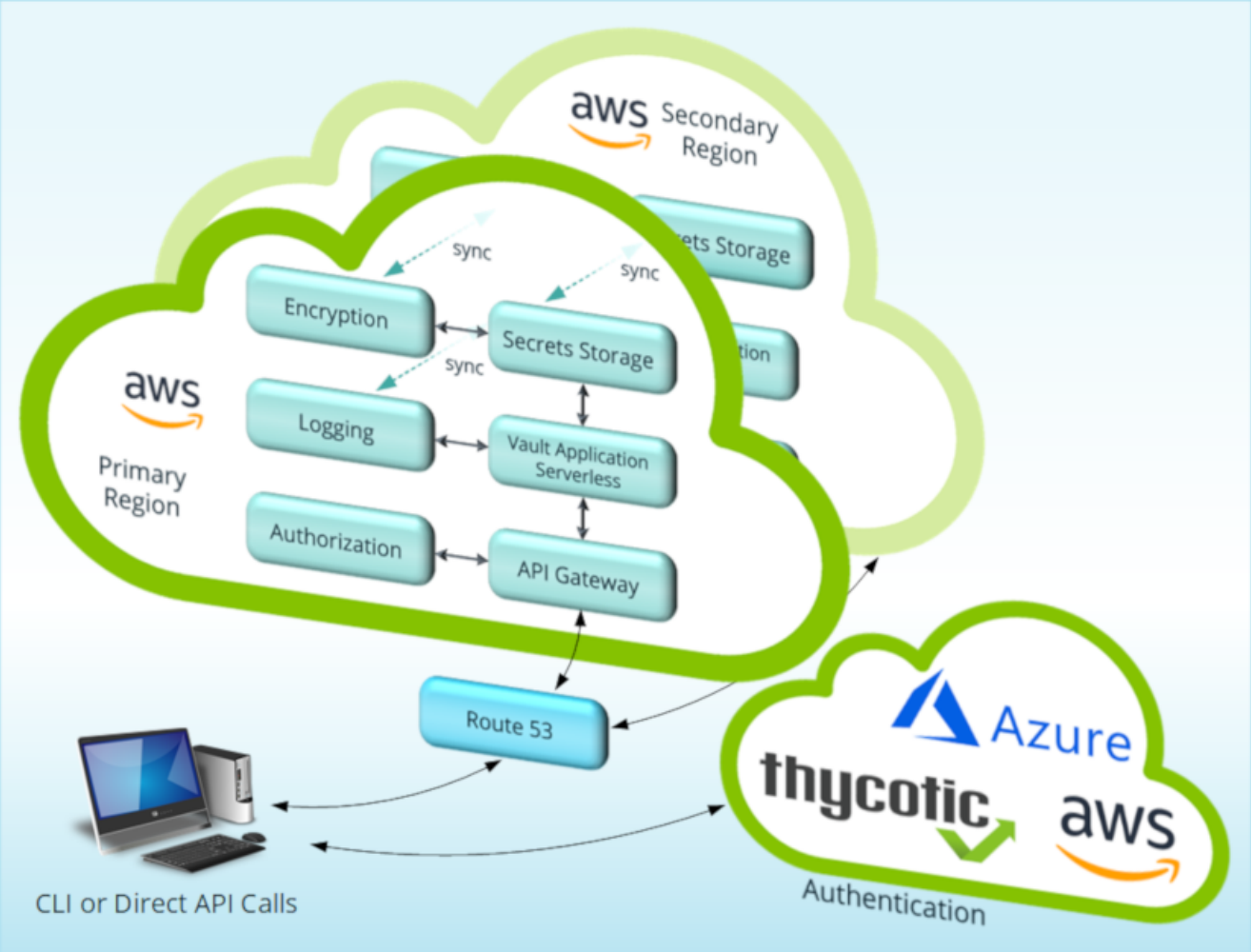
Downloads

DSV CLI Executables Download Page

DSV Concepts

This section covers some key concepts that are important to understand DSV operations. It is recommended to read each subsection for understanding before using the application.

Architecture and Security



Users authenticate locally or by a Thycotic One, Amazon AWS, Microsoft Azure, or Google Cloud Platform authentication provider.

Within the DSV application platform, the API Gateway receives API calls, obtains the responses, and relays them to the caller using HTTPS GET, PUT, POST and other methods common to the REST architecture. The Authorizer uses OAuth to handle API Gateway authorization.

The Vault Application hosts the core DSV functionality and auto-scales to demand.

Extensive logging enables strong audit trails and protections, while encryption protects Secrets at-rest an in-transit.

Availability

Delinea architected DSV to support 5-nines (99.999%) uptime.

Business Continuity and Disaster Recovery

DevOps Secrets Vault leverages AWS DynamoDB global tables for data storage, with a configuration using automatic dual-region replication as a continuous backup mechanism.

- Of the two AWS Regions used in this architecture, one serves as the primary application platform and the other as a hot stand-by.
- Delinea monitors both regions via AWS Route 53 so that if the primary platform fails, client traffic automatically routes to the hot stand-by in under one minute.

Allow List

Since DSV's outbound IP for syslog comes from AWS Lambda, a static address cannot be provided for allow list purposes. Instead, use the cmds from the AWS documentation below and filter by the AWS public IP ranges for the service and region: https://aws.amazon.com/blogs/developer/querying-the-public-ip-address-ranges-for-aws/. Alternatively, syslog messages can be routed through the engine to remove the need for an IP allow list.

Note: As AWS ranges change, the allow list should be periodically updated.

Confidentiality

Data at Rest

Information about customers in DynamoDB, application activity and related logs stored in S3 and sometimes in Elasticsearch during analysis, will always be encrypted transparently.

Customer Secret data is further encrypted by the application with a customer specific key in AWS KMS.

Data in Transit

DSV establishes the HTTPS connection using the TLS 1.2 protocols. For server-side authentication, DSV relies on Amazon-issued digital certificates.

Client Authentication

DSV provides five methods for client authentication:

- Username/password (local)
- Username/password (Thycotic One)
- Client ID
- AWS IAM
- Microsoft MSI

Authentication grants an access token with a one-hour time-to-live (TTL). When the token times out, DSV requires re-authentication.

The username/password authentication method uses a refresh token good for 48 hours. The refresh token renews along with each new access token, so the 48 hours counts relative to the last access token’s time of issuance. If the refresh token expires, DSV requires re-authentication.

The initial administrator (the one who signs up for a tenant) is always setup with Thycotic One to enable Delinea support.

Integrity Checks

Both code signing and token signing are used to ensure integrity.

CLI Code Signing

The download website provides a 256-bit hash of the executable files in a text file, so that customers may run a hash check on the downloaded material. The Windows CLI executable is also signed.

Token Signing

Access tokens granted to Users or applications must transit from the client to the API, potentially allowing an unauthorized party to tamper with the tokens. To prevent this, DSV signs access tokens.

Personally Identifiable Information (PII) and GDPR

DSV requires certain personally identifiable information (PII) to identify each User's account. This includes the User's name, email address, and password, these being the minimum necessary for authentication, and the User's IP address, used during auditing as an indicator of the User's location.

DSV functions to store and protect User's "Secrets," and to make the Secrets accessible to the User and potentially their designees. The term Secrets here commonly means passwords, which are not PII, but DSV Users can store anything they choose as a Secret—for example, images, documents, or other files.

- Accordingly, only Users know whether DSV Secrets have PII status.
- Because the nature of DSV is to encrypt and protect Secrets for Users, Secrets that are PII will de facto benefit from DSV's stringent controls for privacy and user control, in accordance with both the letter and spirit of the GDPR.

Only select, trusted employees of Delinea can access Secrets data and decrypt it, and only via a controlled process that generates an audit trail inaccessible to those employees. This serves the interests of users without compromising their privacy and control.

In GDPR terms, Delinea customers are the data controllers, and Delinea is the data processor.

- The customer determines all information (the Secrets) stored in the vault and decides how long to store it.
- Each DSV customer entirely controls their Users, their User Roles, and the access to Secrets by their Users, according to the policies of the customer organization. DSV logs activity so the customer can monitor access and changes to the Secrets, Users, and Roles within the vault—again, all according to the customer's policies.
- For traceability, DSV logs include source IP addresses and time stamps.

Delinea conducts a Privacy Impact Assessment (PIA) annually to verify continued conformance to GDPR principles.

Third Party SOC 2 Conformance Assessment

The Delinea SOC 2 Type II report contains an independent third-party assessment of our control environment. The report is available upon request with an NDA.

The report ties to the AICPA's Trust Services Criteria (specifically the Security, Availability, and Confidentiality criteria) and issues annually in accordance with the AICPA's AT Section 101 (Attest Engagements).

Audit

DSV captures and stores audit logs of actions taken. The following fields are captured in audit data.

Attribute	Description	Example
id	Audit ID	"00000000-1111-2222-8b1f-b94bb1fab746"
tenant	Tenant ID	"abcd1234567890jbo090"
tenantName	Tenant Name	"test"
principal	Security principal that performed action	"users:user"
principalItemId	Principal item ID	"12345678-0000-41b8-8b02-0123456789ab"
action	Action performed	"POST"
status	Response status code	200
path	Resource path action performed on	"token"
ipaddress	IP Address logged from client	"10.10.10.10"
created	Audit created date	"2020-05-01T01:09:07.225694779Z"
message	Additional details	"login succeeded"

Permissions

To allow reading audit logs create a policy that allows `list` action on `audit` resource. Example of creating such a policy via CLI:

```
dsv policy create --path audit --actions list --resources audit --subjects groups:audit-
readers
```

API Endpoint

You can make direct REST API requests to access audit logs. Example using `curl` as follows:

```
curl -s -H "Authorization: ${DSV_TOKEN}"
'https://example.secretsvaultcloud.com/v1/audit?startDate=2023-04-20'
```

Read more at Audit API documentation page.

CLI Command

DSV CLI supports reading and filtering audit logs via the `dsv audit` command. Read more at Audit Command page.

UI View

DSV Web UI (or simply UI) can display audit logs. Learn more at Audit page.

SIEM

The audit logs can be sent to registered Security Information and Event Management (SIEM) servers in near real time. DSV supports following types of SIEM listeners:

Type	Transport
Syslog	UDP, TCP, TLS
CEF	UDP, TCP, TLS
JSON	UDP, TCP, HTTP, HTTPS
Splunk	HTTPS

Read more at SIEM Audits page.

Available Audit Logs

Path	Method	Status	Description
clients	POST	201	Log when client is created successfully
clients:{clientId}	GET	200	Log when client is read
clients:bootstrap:{clientId}	GET	200	Log when client credential is read
clients	GET	200	Log when client search is performed
clients:{clientId}	DELETE	200	Log when client is deleted
clients:{clientId}:restore	GET	200	Log when client is restored
config:auth	POST	201	Log when new auth provider is saved
config:auth:{name}	GET	200	
config:auth:{name}	PUT	200	Log when auth provider is updated

Path	Method	Status	Description
config:auth:{name}:version:{version}	GET	404,200	Log when auth provider is read by version
config:auth	GET	200	Log when auth provider is searched
config:auth:{name}:rollback:{version}	PUT	404,200	Log when auth provider config is rolled back
config:auth:{name}	DELETE	200	Log when auth provider config is deleted
config:auth:{name}:restore	GET	200	Log when auth provider config is restored
config:policies:{path}	GET	200	Log when policy is read
config:policies:{path}:version:{version}	GET	404,200	Log when policy is ready by version
config:policies	POST	201	Log when policy is created
config:policies:{path}	PUT	200	Log when policy is updated
config:policies:{path}:rollback:{version}	PUT	404,200	Log when policy is rolled back
config:policies	GET	200	Log when policy is searched
config:policies:{path}	DELETE	200	Log when policy is deleted
config:siem	POST	201	Log when siem endpoint is registered
config:siem:{name}	PUT	200	Log when siem endpoint is updated
config:siem:{name}	GET	200	Log when siem endpoint is read
config:siem:{name}	DELETE	200	Log when siem endpoint is deleted
crypto:key:{path}	POST	201	Log when new encryption key is created
crypto:rotate	POST	201	Log when data and key are rotated
crypto:key:{path}	GET	200	Log when key metadata is read
crypto:key:{path}	DELETE	204	Log when key is deleted

Path	Method	Status	Description
crypto:key:{path}:restore	PUT	204	Log when key is restored
crypto:encrypt	POST	200	Log when data is encrypted
crypto:decrypt	POST	200	Log when data is decrypted
engines	POST	201	Log when dsv engine is created
engines:{name}:ping	POST	200	Log when an engine is pinged
engines:{name}	GET	200	Log when an engine is read
engines:{name}	DELETE	200	Log when an engine is deleted
pools	POST	201	Log when a pool is created
pools:{name}	GET	200	Log when a pool is read
pools:{name}	DELETE	204	Log when a pool is deleted
groups	POST	201	Log when a group is created
groups:{name}:members	POST	200	Log when a group member is added
groups:{name}	GET	200	Log when a group is read
users:{name}:group	GET	200	Log when group members are read
groups:{name}:members	DELETE	204	Log when group members are deleted
groups:{name}	DELETE	200	Log when group is deleted
groups:{name}:restore	GET	200	Log when group is restored
groups	GET	200	Log when groups are searched
pki:register	POST	201	Log when CA root is saved
pki:root	POST	200	Log when CA root is generated
pki:sign	POST	200	Log when certificate is signed
pki:leaf	POST	200	Log when leaf certificate & key are created
pki:ssh-cert	POST	200	Log when SSH cert is saved/generated

Path	Method	Status	Description
roles	POST	201	Log when role is created
roles:{name}	PUT	200	Log when role is updated
roles:{name}	GET	200	Log when role is read
roles:{name}:version:{version}	GET	200	Log when role is read by version
roles	GET	200	Log when roles are searched
roles:{name}	DELETE	200	Log when role is deleted
roles:{name}:restore	GET	200	Log when role is restored
task:status:{id}	GET	200	Log when task status is read
token	POST	200	Log when user authenticates successfully
revoke:{refreshtoken}	POST	204	Log when a refresh token is revoked
token	POST	0	Log when user authentication attempt fails
users:{name}	PUT	200	Log when a user is updated
users	POST	201	Log when a user is created
users:{name}:password	POST	200	Log when user password is updated
users:{name}	GET	200	Log when user is read
users:{name}:version:{version}	GET	200	Log when user is read by version
users	GET	200	Log when users are searched
users:{name}	DELETE	200	Log when user is deleted
users:{name}:restore	GET	200	Log when user is restored
config	GET	500,404,200	Log when config is read
config:version:{version}	GET	404,500,200	Log when config is read by version
config	POST	400,500,201	Log when config is created or updated

Path	Method	Status	Description
secrets:{path,id}	GET	404,200	Log when secret is read
secrets:{path,id}:version:{version}	GET	404,200	Log when secret is read by version
secrets:{path,id}:rollback:{version}	PUT	404,200	Log when secret is rolled back
secrets:{path,id}::description	GET	404,200	Log when secret is described
secrets:{path}::listpaths	GET	0	Log when secret paths are listed [disabled]
secrets:{path}	POST	201	Log when secret is created
secrets:{path,id}	PUT	200	Log when secret is updated
secrets:{path,id}	DELETE	200	Log when secret is deleted
secrets:{path,id}:restore	GET	200	Not logged
secrets	GET	200	Log when secrets are searched
home:{principal}:{path}	GET	404,200	Log when home secret is read
home:{principal}:{path}	POST	201	Log when home secret is created
home:{principal}:{path}	PUT	200	Log when home secret is updated
home:{principal}:{path}	DELETE	200	Log when home secret is deleted
home:{principal}:{path}::description	GET	404,200	Log when home secret is described
home:{principal}	GET	200	Log when home is searched
home:{principal}:{path}:version:{version}	GET	404,200	Log when home secret is read by version

Break Glass

The **Break Glass** feature is intended for emergency use if the **Super Administrator** account credentials are lost or compromised. Break Glass allows a selected group of DSV users to recover Super Administrator access.

When **Break Glass** is first setup, DSV distributes **shares** of the Super Administrator credentials to users who will have Super Administrator access after Break Glass is triggered. If enough shares are combined, the users can change ownership of the Super Administrator account to a new group of admins.

To trigger a Break Glass event, a user will run the `breakglass` command along with the minimum number of shares needed to recover the account.

Steps for using break glass.

Bring Your Own Key (BYOK) Encryption

All customer data in DSV is encrypted at rest and in transit, using Delinea-managed keys in AWS Key Management Service (KMS). BYOK encryption allows you to encrypt your cloud product data with keys hosted in your own AWS account. With BYOK encryption, you have more control over the management of your keys. You can also revoke access at any time.

There are many benefits of BYOK encryption.

- Reduced risk: BYOK adds another layer of protection for sensitive data.
- Improved data governance: Access to encryption keys hosted in your AWS account can be logged and monitored via AWS CloudTrail.
- Increased control: You can revoke access to your encryption keys.

DSV's BYOK Approach

We support encryption using encryption keys generated and hosted in your AWS account via the AWS Key Management Service (KMS). This solution enables encryption of your data at different layers throughout the applications.

Usage and Examples of BYOK

Dynamic Secrets

Dynamic Secrets are automatically generated at the time of request. This differs from the standard Secret store read request where the credentials remain the same until changed by a user. They can be used when you need to provide credentials to a user or resource, like a configuration tool, but the access should expire after a set period of time.

Supported Types:

IaaS Dynamic Secrets

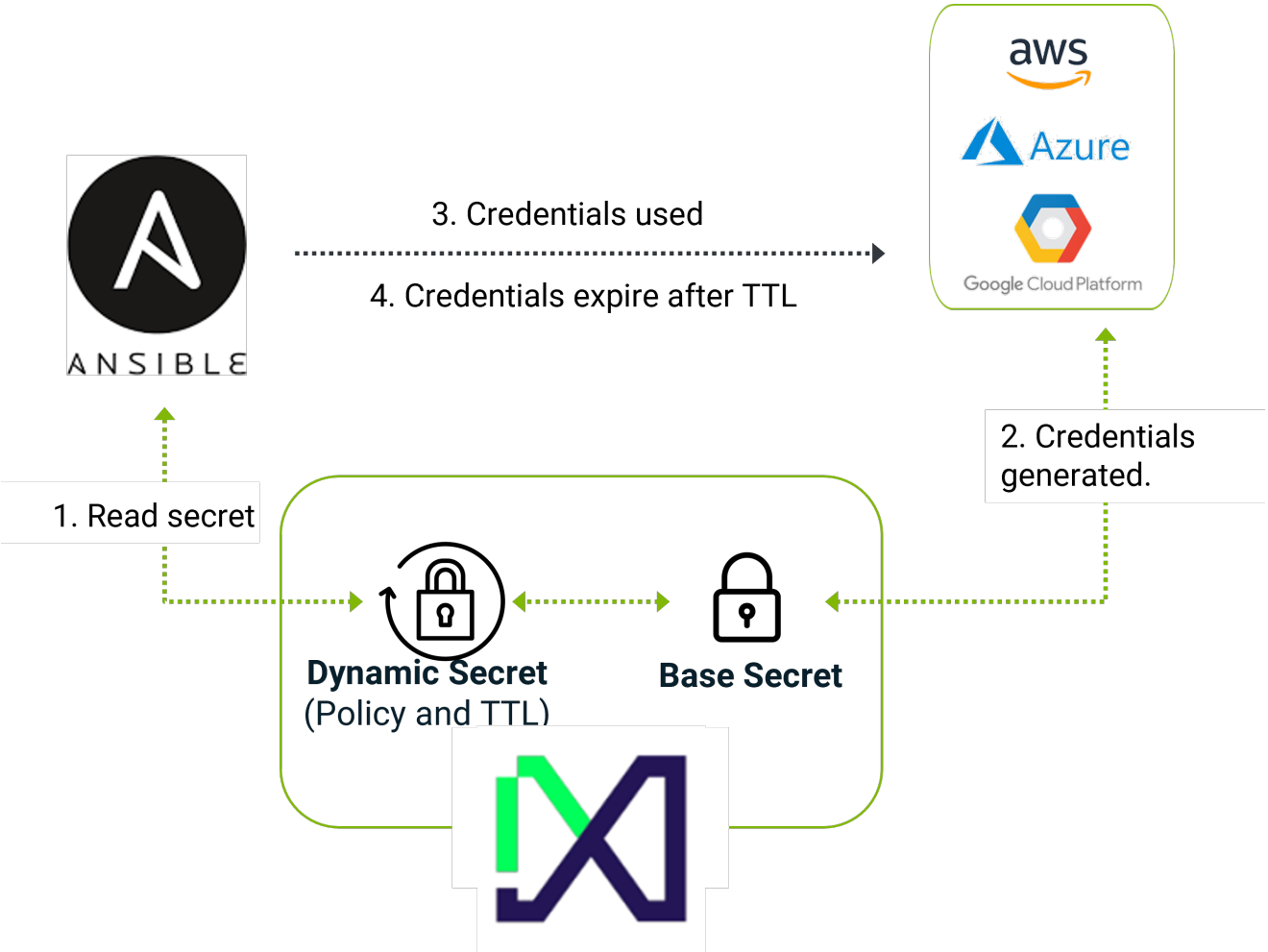
- AWS
- Azure AD Graph
- Azure MS Graph
- GCP

Database Dynamic Secrets

- MSSQL
- MySQL
- Oracle
- PostgreSQL
- MongoDB

Linking

In order for Dynamic Secrets to be generated, they rely on a Base Secret stored in DSV that contains the provider's credentials that are used to automatically generate the ephemeral access keys.



The linking is done through the `attributes` section in the Secret JSON. For example the following Secret `temp-api` has no data, but is linked to a different AWS IAM Secret that contains the access and secret key information. The `linkConfig` defines the type of linking and the linked Secret path.

Attribute	Description
linkConfig	link type and path to the linked Secret
linkConfig.linkType	the only valid value is "dynamic"
linkConfig.linkedSecret	secret path to the base credential

```
{
    "id": "cc619722-6538-4891-b0a6-2c7fa1776a67",
    "path": "dynamic:aws:creds:temp-api",
    "attributes": {
        "linkConfig": {
            "linkType": "dynamic",
            "linkedSecret": "base:aws:creds:api-account"
        }
    },
    "description": "",
    "data": {
    }
}
```

Search for linked Secrets

To get a list of all dynamic secrets linked to a base secret, issue the command `dsv secret search --query <base secret path> --search-links`

Refer also to dynamic secrets and steps.

Refer also to engine and steps.

Encryption as a Service

DSV offers both a **fully managed** and a **user managed** Encryption as a Service (EaaS). DSV is able to encrypt/decrypt **strings** and **files** under 2MB via the fully-managed encryption API, the manual encryption API or in the CLI using the `crypto` command. The key used for the encryption and decryption is stored as a secret-like object within DSV's architecture. The operations of encrypting and decrypting data are done on-the-fly. Those results are returned to the caller immediately and are not saved within DSV.

steps

DSV Engine

An engine is an agent performing tasks on any remote machine. After deployment, the agent opens a real-time two-way communication channel with the main DSV API. Users of the API can send the agent tasks to complete, and the agent, having completed a task or failed, reports back to the caller.

An engine is designed to be a long-running process that completes tasks on demand and automatically in the background.

The initial use of the DSV Engine will be to support database dynamic secrets. In this use-case, a user or application will request access to a database. DSV will have a "base" secret that gives DSV access to the database and permission to create users along with permissions and credentials. DSV will provide those new credentials to the user or application for use. Then when the TTL expires, DSV will go back to the database and delete that user. This provides just-in-time access and eliminates the need for credential rotation.

Future uses of the DSV Engine will include additional authentication methods and password rotation.

Organization Firewall

The DSV Engine uses secure websockets (wss) on port 443 TCP outbound. Since most organizations will already have this port open for web access, you will likely not need to make firewall changes.

steps

Usage

This section covers detail day-to-day vault usage and operations. Every page in this section is recommended reading for anyone using or operating DSV.

CLI Reference

Organized by the type of command object, these articles use task-oriented examples to show you how to use DevOps Secrets Vault.

CLI commands commonly act on these object types:

- Secret
- User
- Policy
- Group
- Role
- Client
- Config

This Reference complements the separately maintained DevOps Secrets Vault API Reference.

CLI Command Syntax

With few exceptions, CLI commands follow a simple syntax:

```
dsv <object> <action> [<args>]
```

For example, the CLI command that creates a new role looks like this:

```
dsv role create --name "my-role" --desc "role for example"
```

Objects

Object	Definition
audit	Show audit records.
auth	Get auth token, manage auth cache or change password.
breakglass	Manage Break-Glass setup.
cli-config	Manage the CLI configuration.
client	Manage client credentials.
config	Manage main config or auth providers.
crypto	Encryption-as-a-Service.
engine	Manage engines.
eval	Inspect environment and configuration values.
group	Manage groups.
home	Manage Home Vault secrets.
init	Initialize or add a new profile to the CLI configuration.
pki	Manage certificates.
policy	Manage policies.
pool	Manage engine pools.
report	Shows report records for secrets and groups.
role	Manage roles.
secret	Manage secrets.
siem	Manage SIEM endpoints.
usage	Fetch API usage info.
user	Manage users.
whoami	Show current identity.

Workflows

A workflow is a series of questions that guides the user through the create or update process. For many objects, if the action is `create` or `update`, then adding no flags will start a workflow.

Workflow supported commands include:

- dsv init
- dsv config auth-provider
- dsv policy
- dsv siem
- dsv pki
- dsv user
- dsv group
- dsv role

If the object doesn't support a workflow, then the flag `--help` is assumed.

Parameters

Parameters can be:

- strings or numerics
- Boolean
- JSON data
- file path

Strings

Most commands take strings as parameters, quoted or unquoted. For example, the username uses quotes but the password does not. Both are valid string parameter values.

```
dsv user create --username "admin1" --password BadP@ssword
```

If a string value has spaces, it must be wrapped in quotes. For example, when creating a Role, the description should be quoted.

```
dsv role create --name test-role --desc "a test role"
```

Boolean

Some parameters are simple Boolean flags controlling whether or not something applies. For example, use `--plain` to not beautify the JSON output.

```
dsv secret read --path example:bash-json --plain
```

JSON Data and OS-Specific Syntax

In some cases the parameter expects JSON. For example, the `--data` parameter on a `dsv secret create` command expects JSON data.

JSON parameter formatting depends on the OS and shell program.

- Linux: wrap the JSON in a single quote (')
- PowerShell: wrap the JSON in a single quote (') and inside the JSON escape each double quote (") with a backslash (\)
- cmd.exe: wrap the JSON in a double quote (") and inside the JSON escape each double quote (") with a backslash (\)

```
dsv secret create --path example:bash-json --data '{"password":"bash-secret"}'
```

```
PS C:> dsv secret create --path example:ps-json --data '{\"password\":\"powershell-
secret\"}'
```

```
C:> dsv secret create --path example:cmd-json --data "{\"password\":\"cmd-secret\"}"
```

File Path and OS-Specific Syntax

Passing JSON as a parameter remains practical only as long as the JSON remains short. Instead of passing JSON as a parameter, you can pass it as a file, using the @ prefix to specify the path to the file.

For instance, here the command is to create a Secret using a local file named secret.json. The examples show the minor variations among operating systems and shells.

```
dsv secret create --path example:bash-json --data @secret.json
```

```
PS C:> dsv secret create --path example:ps-json --data '@secret.json'
```

```
C:> dsv secret create --path example:cmd-json --data @secret.json
```

For passing a file as data, only Powershell requires the file path and name to be wrapped in quote marks, in this case single-quote marks.

Output Modifiers

DSV offers global flags that combine with most commands to format or redirect output.

- `--encoding, -e` specify the output format as either JSON or YAML
- `--filter, -f` filter to output only a specific JSON attribute; this feature uses the jq library
- `--out, -o` control the output destination; valid values: *stdout*, *clip*, and *file:[file-name]*, with *stdout* the default
- `--plain` do not beautify JSON or YAML output

Encoding

```
dsv secret read --path servers:us-east:server01 -e yaml
```

Outputs:

```
attributes: null
data:
  host: server01
  password: Secretp@ssword
  username: administrator
id: c5239a6c-422e-4f57-b3a6-5167656af852
path: servers:us-east:server01
```

Filter

The filter modifier relies on a lightweight, flexible command line JSON processor, the jq library. Visit the JQ GitHub repo to learn more about how to use JQ.

The following code block illustrates:

```
dsv secret read --path resources:server01:mysql
```

Outputs:

```
{
  "attributes": {
    "tag1": "this is a tag"
  },
  "created": "2019-07-17T21:33:35Z",
  "createdBy": "users:ben",
  "data": {
    "foo": ["bar2", "blah"],
    "password": "root-password",
    "username": "blah"
```

```
  },
  "id": "59f2ab72-7f51-4f0e-8ffd-35cb94b818fb",
  "lastModified": "2019-07-17T21:36:01Z",
  "lastModifiedBy": "users:ben",
  "path": "resources:server01:mysql",
  "version": "1"
}
```

```
dsv secret read --path resources:server01:mysql --filter data.password
```

Outputs:

```
root-password
```

The command without the filter produced the entire Secret, while the command with the filter read out only the password value.

Out

The *-o* modifier allows output to be redirected to a file.

```
dsv secret read --path servers:us-east:server01 -o file:Secret.json
\$ nano Secret.json
```

Contents of Secret.json:

```
{
  "attributes": null,
  "data": {
    "host": "server01",
    "password": "Secretp@ssword",
    "username": "administrator"
  },
  "id": "c5239a6c-422e-4f57-b3a6-5167656af852",
  "path": "servers:us-east:server01"
}
```

Using *-o clip* puts the command output on the OS clipboard.

Output Piping

Output piping takes advantage of a common coding practice in which the value of a parameter passed to a command is itself a command or set of commands. When the outer command receiving the parameter executes, it

evaluates the parameter, which requires it to run the command that was passed as a parameter. The output of that command becomes the parameter value for the outer command, which then continues to execute.

As an example, you can save any DSV CLI output into an environment variable by piping the output from the standard output into an environment variable.

```
export MYSecret=$(dsv secret read --path Secret1)
```

```
$MYSecret=dsv secret read --path Secret1
```

Both of the preceding would create an environment variable named *MYSecret* that would store the Secret data. To view the data you would use:

```
echo $MYSecret
```

Secret

Secrets are sensitive data protected in your vault. Many Secrets relate to authentication–such as passwords, SSH keys, and SSL certificates–but Secrets can be anything represented as a file on computer storage media.

When DSV has possession of Secrets outside the vault (that is, the CLI or API has reproduced a Secret anywhere outside the vault), it keeps the Secrets encrypted and locked down in conformance to the specific permissions and policies in the config.

Commands that Act on Secrets

Command	Action
bustcache	clear the Secret cache
create	create a Secret in the vault
search	search for Secrets
describe	view Secret metadata only
read	view a Secret's data
edit	modify a Secret using the OS's default command-line editor, such as **VI**, **nano**, or **Notepad**
update	modify a Secret, with `--data`, `--attributes` and `--desc` flags to modify selected portions only, and a Boolean `--overwrite` flag to control whether the `--data` flag's content overwrites or merges with extant data object fields

Usage

Command	Action
delete	delete a Secret
restore	restore a Secret (if within 72 hours of deletion)
rollback	for a Secret that has had more than one version, roll back to an earlier version

Examples

Bustcache

The *bustcache* command clears the local cache, if present.

```
dsv secret bustcache
```

Create

The `create` command uses the `--data` flag to pass data into the secret. This flag accepts JSON entered directly into the command line or by a path (absolute or relative) to a JSON file.

Bash examples

```
dsv secret create --path us-east:server02 --data '
{"username":"administrator","password":"bash-secret"}'
```

```
dsv secret create --path us-east:server02 --data @/home/user/secret.json
```

```
dsv secret create --path us-east:server02 --data @../secret.json
```

Powershell examples

```
PS C:> dsv secret create --path us-east:server02 --data '
{\"username\":\"administrator\",\"password\":\"powershell-secret\"}'
```

```
dsv secret create --path us-east:server02 --data '@/home/user/secret.json'
```

```
dsv secret create --path us-east:server02 --data '@../secret.json'
```

Usage

CMD Examples

```
PS C:> dsv secret create --path us-east:server02 --data "
{\"username\":\"administrator\",\"password\":\"cmd-secret\"}"
```

```
dsv home secret --path us-east:server02 --data @/home/user/secret.json
```

```
dsv home secret --path us-east:server02 --data @../secret.json
```

The `--attributes` flag can be used to add user-defined metadata in the same way that data is added.

The `--desc` flag can be used to add a simple string. If the string has any spaces, then it should be enclosed in double quotes.

As a Bash example:

```
dsv secret create --path us-east:server02 --attributes '{"priority":"high"}'  --desc
"Covert Secret" --data '{"username":"administrator","password":"bash-secret"}'
```

Update

Use *update* to change a Secret's data. The command has several flags pertinent to Secrets:

- the `--data` flag allows you to only update the data portion of the Secret
  - the Boolean `--overwrite` flag controls whether the `--data` flag's content overwrites or merges with extant data object fields
  - the data object accepts as many fields as you choose
- the `--attributes` flag allows you to only update the attributes of the Secret
- the `--desc` flag allows you to only update the description of the Secret

The `--overwrite` flag applies only at the field level; it does not allow you to merge new attributes of a data field into existing attributes of that field, only to merge new data fields into the extant set of data fields.

As with *create*, for the value of the `--data` parameter `update` accepts JSON entered directly at the command line, or the path to a JSON file.

```
dsv secret update --path us-east:server02 --data {\\"password\\":\\"Secret2\\"}
```

or

Usage

```
dsv secret update --path us-east:server02 --data @secret.json
```

update is similar to *create* but operates on an existing secret. When using *update* for other commands like policy or auth-providers, it is an all or nothing change. ie, for those if you want to change only one field, you have to update all of them. However, for Secrets, it is possible to update only one field and not change the others.

If you have this secret:

```
{
  "attributes": {
    "attr": "add one"
  },
  "created": "2019-09-20T16:12:57Z",
  "createdBy": "users:thy-one:admin@company.com",
  "data": {
    "host": "server01",
    "password": "badpassword"
  },
  "description": "update description",
  "id": "c893b4f8-9425-4fa4-acbf-2806d6f1fa82",
  "lastModified": "2020-01-17T15:43:27Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "path": "servers:us-east:server01",
  "version": "12"
}
```

This Bash command will only change the value for *host* in the data section.

```
dsv secret update servers:us-east:server01 --data '{\"host\":\"unknown\"}'
```

```
{
  "attributes": {
    "attr": "add one"
  },
  "created": "2019-09-20T16:12:57Z",
  "createdBy": "users:thy-one:admin@company.com",
  "data": {
    "host": "unknown",
    "password": "badpassword"
  },
  "description": "update description",
  "id": "c893b4f8-9425-4fa4-acbf-2806d6f1fa82",
  "lastModified": "2020-08-03T17:58:29Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "path": "servers:us-east:server01",
  "version": "13"
}
```

The flag `--overwrite`, if added to the above command would wipe-out the description and any other data KV pairs. So this flag requires caution.

```
dsv secret update servers:us-east:server01 --data '{\"host\":\"unknown\"}' --overwrite
```

Search

You can search for Secrets by path, attribute, or id.

Some examples

```
dsv secret search server

dsv secret search --query server

dsv secret search -q aws:base:secret --search-links

dsv secret search --query aws --search-field attributes.type

dsv secret search --query 900 --search-field attributes.ttl --search-type number

dsv secret search --query production --search-field attributes.stage --search-comparison
equal
```

flags

`--query`, `-q` Query of secrets to fetch (required)

`--limit` Set the maximum number of search results that will display per page (cursor)

`--cursor` Accepts the element used to get the next page of results

`--search-comparison` Specify the operator for advanced field searching, can be 'contains', 'equal', or 'begins_with' Defaults to 'contains' (optional)

`--search-field` Advanced search on a secret field such as 'attribute.type' or 'description'. Defaults to 'path'. (optional)

`--search-links` Find secrets that link to the secret path in the query (optional)

`--search-type` Specify the value type for advanced field searching, can be 'number' or 'string'. Defaults to 'string' (optional)

`--sort` Change the sort order using `asc` or `desc` as values. Sort defaults to descending. (optional)

For a search where there are more results than returned in the first set, the API returns a cursor—a large piece of text. You pass that back to get the next set of results.

For example, if the command `dsv secret search -q admin --limit 10` matched 12 Secrets with admin in the name, the CLI would return the first 10 plus a cursor. To obtain the next two results, you would use this command:

Usage

```
dsv secret search -q admin --limit 10 --cursor AFSDFSD...DKFJLSDJ=
```

Cursors may be lengthy:

```
dsv secret search -q resources --limit 10 --cursor
eyJpZCI6ImEwOTFjOWIzLWE4MmQtNGRiYy1hYThiLTYxMDY0NDZhZjA3MSIsInBhdGgiOiIiLCJ2ZXJzaW9uIjoidi
1jdXJyZW5OIiwidHlwZSI6IiIsImxhdGVzdCI6MH0=
```

Describe

Use *describe* to show only metadata; you will not see the actual Secret value.

```
dsv secret describe --path us-east:server02
```

Read

The `read` command shows both the Secret data and metadata.

```
dsv secret read --path us-east:server02
```

Flags

`--encoding` or `-e` converts the output to JSON (default) or YAML.

`--out` or `-o` can send the read response to `stdout` (default), the clipboard (`clip`), or a file (`file:<filename>`)

`--filter` or `-f` filters to a specific KV pair. So data.password would only output the password value.

This example would send the password value only to the clipboard.

```
dsv secret read secret2 -o clip -f data.password
```

TIP: Although the `-o` flag allows redirection of output to files, it does not support directly assigning the output to an environmental variable. However, you can use piping to achieve that outcome.

Piping refers to passing to a command a parameter value that is itself a command, or assigning to a variable a value that is a command. In effect, piping means assigning as a value the means to obtain the value, rather than the value itself.

```
export TEST=\$(dsv secret read --path us-east:server02)
```

or

Usage

```
\$TEST=dsv secret read --path us-east:server02
```

Both examples use piping to assign to the variable *TEST* the value contained in the Secret, by making the `secret read` command a parameter within a larger command or statement.

Once stored as the value of *TEST*, the data remain easily accessible:

```
echo \$TEST
```

As a well established computing technique of long standing, piping is not limited to Secrets. You can use piping to store any output–search results, configuration states, and more.

Edit

Use *edit* to open the Secret data in the default text editor for bash, such as **vi**, **nano**, or **Notepad**.

- Saving in the editor updates the Secret in the vault, except in the case of Notepad, in which case the update happens when you exit Notepad. Your interim saves are to the working copy.

```
dsv secret edit --path us-east:server02
```

Delete

To *delete* a Secret simply specify the path.

```
dsv secret delete --path us-east:server02
```

When you delete a Secret, it will no longer be usable. However, with the soft delete capacity of DSV, you have 72 hours to use the *restore* command to undelete the Secret. After 72 hours, the Secret will no longer be retrievable.

Should you want to perform a hard delete, precluding any restore operation, you can use the *delete* command's `--force` flag.

Restore

Up to 72 hours after you delete a Secret (but not if you hard deleted it using the `--force` flag), you can restore it:

```
dsv secret restore --path us-east:server02
```

Do not confuse `restore` with `rollback` because the two have no relation. While `restore` un-deletes a deleted Secret, restoring it to the condition it was in at the time of its deletion, `rollback` does not operate on deleted Secrets. It simply sets a Secret back to an earlier version of itself.

Rollback

A Secret that has had more than one version can be rolled back to an earlier version of itself:

```
dsv secret rollback --path us-east:server02 --version 2
```

If you do not include the `--version` flag, the Secret will roll back to the last version before the present version. By serially issuing the rollback command without a version number, you could step back through the versions one at a time.

Note that the rollback is non-destructive; technically, the command does not roll back so much as retrieve the indicated version and duplicate it as a new version, which becomes the current version.

- If you used the `--version` flag to jump back three versions, you would not lose those three versions; they would remain in place, with the version from three back now being replicated into a new version.

It is important to distinguish between the `rollback` feature, which relates to versions, and the `restore` feature, which relates to the `delete` feature and has nothing to do with versions.

A deleted Secret can be restored up to 72 hours after it has been deleted (if it was not hard deleted using the `--force` flag), after which it cannot be restored. Rollback does not change that in any way, because it cannot operate on a deleted Secret.

If a deleted Secret is restored, Rollback can operate on it just as it would any other Secret.

User

For DSV, the term "user" refers to a security principal in the vault that can authenticate locally by a username and password or can authenticate through a federated provider such as Amazon Web Services or Amazon Resource Names.

Understanding Qualified Usernames

When a User or Role ties to a third-party provider, the name will be the fully qualified name to help distinguish potentially duplicate User or Role names across different systems.

The name qualifier format *provider name:local name* means for example that the *test-admin* User will have the username *aws-dev:test-admin* while the local User with username *test-admin* will not have a qualifier, so its username will just be *test-admin*.

Commands that Act on Users

Command	Action
change-password	change a local User's password
create	create a User in the vault
search	find Users by username

Usage

Command	Action
read	read a User's details
delete	delete a User from the vault
restore	restore a deleted User (if within 72 hours of deletion and not hard deleted)
update	change a User's parameters

Examples

Change password

The *change-password* command, effective for local Users only, initiates an elemental password change sequence:

```
dsv auth change-password

Please enter your current password:
*************

Please enter the new password:
*************

Please enter the new password (confirm):
*************
```

With a local User, correct entry for the current password prompt, and valid, matching responses to the first and second prompts for the new password, the response will be a message that the password has been changed.

A Thycotic One Federated User must instead visit Thycotic One to change their password. Attempting to use the *change-password* command within the CLI will fail.

Create

The *create* command takes several `--parameters` that specify foundational aspects of the User record.

Note: Only the `username` and `password` parameters are required. The command is used to updated 'password' and 'displayname'. Other parameters are ignored.

Parameter	Content
`--username`	local username; required; supports local authentication by username and password; need not match that used by a federated authentication provider (if present)
`--password`	password for local authentication by username and password
`--provider`	matches the *name* attribute of the authentication provider in the *settings* section of the config

Usage

Parameter	Content
`--external-id`	identifier recognized by third-party federated authentication providers, such as AWS or ARN
`--displayname`	locally used display name for identifying users in DSV

Create a local User with username *test-admin* and password *secret-password*:

```
dsv user create --username test-admin --password secret-password
```

Create a User account for login by the AWS *IAM test-admin* User, with the account tied to an *aws-dev* account in the configuration:

```
dsv user create --username test-admin --external-id arn:aws:iam::00000000000:user/test-
admin --provider aws-dev
```

Search

The *search* command locates Users by searching on their usernames. It accepts as a `--query` parameter the username you provide, and searches for records with a matching username.

Note: Entering `dsv user search`, without parameters, produces a list of all users.

```
dsv user search --query test-admin
```

Output:

```
[
  {
    "externalId": "arn:aws:iam::00000000000:user/test-admin",
    "provider": "aws-dev",
    "qualifier": "bgno6etchfrc72getij0",
    "userId": "dd632a7f-419f-400b-9e36-f67603bf934b",
    "userName": "test-admin"
  },
  {
    "externalId": "",
    "provider": "",
    "userId": "8be917b3-9577-4dba-b39f-b531f27c1caa",
    "userName": "test-admin"
  }
]
```

Read

The *read* command retrieves and displays information without changing anything.

Provide a fully qualified username and read the User's details:

```
dsv user read --username aws-dev:test-admin
```

Provide a full local username and read the User's details:

```
dsv user read --username test-admin
```

Delete

The *delete* command will remove records of both local Users and Users associated with third-party authentication providers. In both cases, you must provide the fully qualified username.

Delete a third-party User identified by a fully qualified name:

```
dsv user delete --username aws-dev:test-admin
```

Delete a local User identified by the full local username:

```
dsv user delete --username test-admin
```

When you delete a User, it will no longer be usable. However, with the soft delete capacity of DSV, you have 72 hours to use the *restore* command to undelete the User. After 72 hours, the User will no longer be retrievable.

Should you want to perform a hard delete, precluding any restore operation, you can use the *delete* command's `--force` flag.

Restore

Up to 72 hours after you delete a User (but not if you hard deleted it using the `--force` flag), you can restore it:

```
dsv user restore --username test-admin
```

Group

A Group facilitate the application of the same policies to all members of a given set of Users.

Commands that Act on Groups

Command	Action
create	create a Group in the vault
add-members	add members to a Group
read	read a Group's details
delete-members	remove members from a Group
delete	delete a Group
restore	restore a Group (if within 72 hours of deletion and not hard deleted)

Examples

Create

File Example

This example command would create a Group named **admins** from a file **data.json** containing *{"groupName": "admins"}* (or same with single-quote marks, for Powershell) and located in the **tmp** folder:

```
dsv group create --data @/tmp/data.json

{
  "groupName": "admins",
  "id": "2ce6754d-afbc-43a9-bfd4-3b7ec61170a0",
  "members": null,
  "metaData": null
}
```

Direct Data Example

This example would create a Group without referencing a file:

```
dsv group create --data {"groupName": "admins"}
{
  "groupName": "admins",
  "id": "2ce6754d-afbc-43a9-bfd4-3b7ec61170a0",
  "members": null,
  "metaData": null
}
```

Note that in Powershell, single quotes are required and double quotes escaped, like this:

Usage

```
dsv group  create --data  '{\"groupName\": \"admins\"}'
```

Wizard Example

A group can also be created using the wizard:

```
dsv group create
```

Find Group Membership

To see what Groups the user Billy belongs to, you would use:

```
dsv user groups --username billy
{
  "groups": [
    {
      "groupName": "admins"
    }
  ],
  "name": "billy"
}
```

Add-Members

Add members to a Group similarly to this example, wherein the file *newmember.json* contains: *{"memberNames": ["billy","larry']}*

```
dsv group add-members --group-name admins --data '@/tmp/newmember.json

{
  "memberNames": ["billy", "larry"]
}
```

Read

This example demonstrates how to read a Group:

```
dsv group read --group-name admins
{
  "groupName": "admins",
  "id": "2dc756d6-ba71-44e9-94e9-f822e0f7ca3f",
  "members": ["larry"],
  "metaData": null
}
```

Update | Assign Group to Policy

This example would assign the **admins** Group to an existing policy at the path *secrets:servers:us-west*:

```
dsv policy update --actions "<.*>" --subjects groups:admins --path secrets/servers/us-west
```

Note that you can designate paths with either of the colon : or forward slash / characters.

Delete-Members

To remove members from a Group, follow this example, wherein *deletemembers.json* contains: *{"memberNames": ["billy"]}*

```
dsv group delete-members --group-name admins --data @/tmp/deletemembers.json
<no response>
```

Note that this does not delete the user objects that were members. It simply makes those user objects no longer members of the Group.

Delete

To delete a Group, you would follow this example:

```
dsv group delete --group-name admins
<no response>
```

When you delete a Group, it will no longer be usable. However, with the soft delete capacity of DSV, you have 72 hours to use the *restore* command to undelete the Group. After 72 hours, the Group will no longer be retrievable.

Should you want to perform a hard delete, precluding any restore operation, you can use the *delete* command's `--force` flag.

Restore

Up to 72 hours after you delete a Group (but not if you hard deleted it using the `--force` flag), you can restore it:

```
dsv group restore --group-name admins
```

Role

With DSV, the term "role" describes a security principal in the vault that ties to third-party providers or client credentials for granting permissions.

Commands that Act on Roles

Command	Action
create	create a Role in the vault
search	find Roles by Role name
read	read a Role's details
update	upload a superseding Role
delete	delete a Role from the vault
restore	restore a deleted Role to the Vault (if within 72 hours of deletion and not hard deleted)

Examples

Create

The *create* command takes several `--parameters` that spec key aspects of the Role record.

Parameter	Content
`--desc`	description of the Role
`--name`	name of the Role
`--provider`	matches the *name* attribute of the authentication provider in the *settings* section of the config
`--external-id`	identifier recognized by third-party federated authentication providers, such as AWS or ARN

Create a local Role with the name `test-role`:

```
dsv role create --name test-role
```

Search

The *search* command locates Roles by searching on their Role names. It accepts as a *--query* parameter the Role name you provide, and searches for records with a matching Role name.

Search for a Role named `dev-admin`:

```
dsv role search --query dev-admin
```

Or simply:

Usage

```
dsv role search dev-admin
```

You can also specify the maximum number of search results per page (limit) and a cursor to get the next batch of results.

```
dsv role search --query dev-admin --limit 2 --cursor
eyJpZCI6ImZmZjZjODUxTJ2ZXJzaW9uIjo5OIiwidHiJ9
```

Read

The *read* command retrieves and displays information without changing anything.

Provide a Role name and read the Role's details in beautified form:

```
dsv role read --name test-role -b
```

Update

Use *update* to change a Role's data.

Note that *update* rewrites the **entire** set of Role data, even if only a single field has changed.

Provide a Role name and update the Role to replace the description field's value:

```
dsv role update --name test-role --desc "a new description"
```

Delete

The *delete* command will remove Roles.

Provide a Role name and delete the Role:

```
dsv role delete --name test-role
```

When you delete a Role, it will no longer be usable. However, with the soft delete capacity of DSV, you have 72 hours to use the *restore* command to undelete the Role. After 72 hours, the Role will no longer be retrievable.

Should you want to perform a hard delete, precluding any restore operation, you can use the *delete* command's `--force` flag.

Restore

Up to 72 hours after you delete a Role (but not if you hard deleted it using the `--force` flag), you can restore it:

```
dsv role restore --name test-role
```

Client

Client credentials enable applications to authenticate as the Role assigned to the client record.

Commands that Act on Clients

Command	Action
create	create a client in the vault
search	find clients by Role name
read	read a client's details
delete	delete a client from the vault

Examples

Create

The *create* command accepts as its `--role` parameter a fully qualified Role name, and creates a client credential assigned to that Role.

```
dsv client create --role app-role
```

The output will include a *clientId* and *clientSecret* suitable for use during CLI installation, or within REST calls to authenticate as the Role assigned to the *clientId*.

```
{
  "clientId": "01234567-abcd-4eb9-9df4-6f1fea7d9298",
  "clientSecret": "aaabbb777DwTLkdzWkL18UF9blycz3r9yfRhQTYICFc",
  "created": "2022-09-16T09:53:50Z",
  "createdBy": "users:bright",
  "id": "00000000-0123-0123-0123-0123456789ab",
  "role": "app-role",
  "url": false
}
```

NOTE: The client Secret is available only when you create the client. If the Secret is lost, delete the client and create a new one.

Ephemeral Credentials

Client credentials can be made temporary by using the `--uses` and `--ttl` flags.

`--uses` determines the number of times the client credential can be read. If set to 0, it can be used infinitely. Uses defaults to 0.

`--ttl` determines long until the client credential expires. If set to 0, it can be used indefinitely. Ttl defaults to 0.

Search

The *search* command accepts as its `--query` parameter the name of a Role, and searches for clients having that Role.

```
dsv client search --query dev-role
```

or

```
dsv client search dev-role
```

Read

The *read* command accepts a client ID as a parameter and returns the details for the given client. As with most commands, remember that you can apply flags to beautify, redirect, or reformat the returned material.

```
dsv client read --client-id 01234567-abcd-4eb9-9df4-6f1fea7d9298
```

Delete

The *delete* command accepts a client ID as a parameter and deletes from the vault the indicated client.

```
dsv client delete --client-id 01234567-abcd-4eb9-9df4-6f1fea7d9298
```

When you delete a Client, it will no longer be usable. However, with the soft delete capacity of DSV, you have 72 hours to use the *restore* command to undelete the Client. After 72 hours, the Client will no longer be retrievable.

Should you want to perform a hard delete, precluding any restore operation, you can use the *delete* command's `--force` flag.

Bootstrapping

There will be times when machines or applications will require access to DSV to get started, but you can't (or don't want) to hardcode the client secret. In this case, we can create the client ID and get a one-time use URL. When the URL is accessed, then the corresponding client secret will be created and returned. The URL will no longer be valid after the initial use, so if the intended machine or application gets an error "url already used" then there should be an alarm to investigate.

First create the Client ID and URL:

Usage

```
dsv client create --role <role> --url --url-ttl <ttl in seconds>
```

Where "role" is a Role created earlier and is attached to a Policy to provide the proper permissions.

`--url` if present tells DSV to create a one-time use URL instead of a Client Secret right now.

`--url-ttl` is the time to live of the URL in seconds. If it is not accessed in that time frame, then the URL will become invalid.

> **Note:** If a TTL is set for both the URL and the underlying client credentials, then the timer for the client credentials will not start until the URL is accessed.

The result will look something like this:

```
{
  "clientId": "01234567-abcd-4eb9-9df4-6f1fea7d9298",
  "created": "2022-09-16T11:04:45Z",
  "createdBy": "users:admin@company.com",
  "id": "c6bae4ae-469f-4ea7-a72a-8f338fee4867",
  "role": "app-role",
  "url": true,
  "urlExpires": "2022-09-16T12:04:45Z",
  "urlPath": "https://company.secrestvaultcloud.com/v1/clients/bootstrap/01234567-abcd-
4eb9-9df4-6f1fea7d9298",
  "urlTTL": 3600
}
```

Then the machine or application can access that urlpath for the Client Secret. For Example, using CURL (or Invoke-RestMethod for Powershell):

```
curl https://company.secrestvaultcloud.com/v1/clients/bootstrap/01234567-abcd-4eb9-9df4-
6f1fea7d9298
```

With a result containing the Client Secret:

```
{
  "id": "c6bae4ae-469f-4ea7-a72a-8f338fee4867",
  "clientId": "01234567-abcd-4eb9-9df4-6f1fea7d9298",
  "clientSecret": "abcdef0123456789ALGewTlxf4Fdo-cTkS_l_oOki8w",
  "role": "app-role",
  "url": true,
  "urlExpires": "2022-09-16T12:04:45Z",
  "accessed": "2022-09-16T11:08:11Z",
  "created": "2022-09-16T11:04:45Z",
  "createdBy": "users:admin@company.com"
}
```

If the URL is accessed a second time, then the response will contain: `{"code":400,"message":"url has already been used"}`

Policy

Policies control access to resources and authorization to act on resources, such as to change them, via **permissions**. DevOps Secrets Vault permissions are foundational for proper operation and security.

Commands that Act on Policy

Command	Action
create	create a policy in the vault
edit	modify a policy using the OS's default command-line editor, such as VI, nano, or Notepad
read	view a policy details
update	policy updates are all or nothing, so required fields must be included in the update and if optional fields are not included, they are deleted or go to default
rollback	for a policy that has had more than one version, roll back to an earlier version
delete	delete a policy
search	search for a policy
restore	restore a policy (if within 72 hours of deletion and not hard deleted)

To get a json encoded list of all Policies, use:

```
dsv policy search
```

You can add a query item to search Policies by path:

`dsv policy search secrets/database` **or** `dsv policy search --query secrets/databases`

A typical Policy looks like this:

```
created: '2019-09-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2019-09-24T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: secrets:servers:us-west
permissionDocument:
- actions:
  - read
```

```
    conditions: {}
    description: ''
    effect: allow
    id: xxxxxxxxxxxxxxxxxxxx
    meta:
    resources:
    - secrets:servers:us-west:<.*>
    subjects:
    - groups:west admins
  version: '5'
```

A policy contains a list of permissions which define access to resource paths. The policy itself has a top level path which is the identifier of the policy as well. The policy path is used to validate the resource paths in the permission documents. This allows administrators to delegate user ownership of policies without allowing self elevation through modifying the policy to a higher level path.

For example, the policy above has a path of `secrets:servers:us-west`. Permissions can be created for resources paths like `secrets:servers:us-west`, `secrets:servers:us-west:<.*>`, or `secrets:servers:us-west:prod:<.*>`. A permission document cannot be created on the policy to allow users to manage users, i.e. with a resource path of `users:<*>`. Because the policy path must be the root of any resource paths in its permission documents.

The one exception is policy delegation. An admin can create a policy and add a resource path for `config:policies:secrets:servers:us-west` to allow users to manage the policy. An example of this is below

The permission document has the following elements:

Element	Definition
actions	a list of possible actions on the resource including create, read, update, delete, list, and assign (regular expressions and list supported)
conditions	an optional CIDR range to lock down access to a specific IP range
description	human friendly description of the Policy intent
effect	whether the Policy is allowing or preventing access; valid values are allow and deny
id	system-generated unique identifier to track changes to a particular Policy
resources	the resource path defining the targets to which the permissions apply; a resource path prefixes the entity type (secrets, clients, roles, users, config, config:auth, config:policies, audit) to a colon delimited path to the resource.
subjects	the Policy provides authorization to these entries. Includes Users, Roles, and Groups

Policy Evaluation

To correctly evaluate permission Policies, you must know the rules that apply to permissions.

- Values for permission properties may optionally be specified using a regular expression enclosed in angle brackets <>. For example,

  a subject entry could be written as `["users:<bob|alice>"]`. Here, users `bob` and `alice` are specified. A longer alternative would be

  `["users:bob", "users:alice"]`.
- Permissions are cumulative.
  - If there is a top level permission for the path secrets:servers:<.*> that grants a User **write** access, then even if they are only granted **read** access at the resource path secrets:servers:webservers:<.*>, they will still have write access due to the top level implicit match.
- `effect` can either be `allow` or `deny`. If not specified, it defaults to `allow`
- An explicit deny trumps an explicit or implicit allow.
- At least one action must be listed in an array. Actions are explicit. A User assigned **update** and **read** will not automatically have **create** for the resource path.
- For actions, the wildcard form `<.*>` replaces any other values, since it is an all-inclusive form. A wildcard could be written as a standard `<.*>` form, but also as `.*` or `*` for convenience. The backend automatically converts it to `<.*>`.
- Invalid actions are not allowed, unless there is a wildcard element. Valid actions are `create, read, update, delete, assign, list`.
- The **list** action has a special behavior.
  - First, **list** (search) is global–it runs across all items of an entity (any of the resources like Users, Roles, Groups, etc), not limited to paths and sub-paths.
  - Second, to grant a User an ability to search entities via *list*, use the root of the entity if you want *list* to include other entities and actions within the same Policy. The root entity, for example, is secrets, with no other characters following.
  - See the example on Search
- At least one subject must be listed in an array. A prefix is required. For example, a valid subject is `"users:bob"`. Valid prefixes are `groups, roles, users`.
- Subjects and actions are automatically converted to lower case upon save.

Policy Examples

When creating or updating a Policy, a workflow can be started using `dsv policy create` or `dsv policy update` without flags. This will start step-by-step questions to guide you though the process. However, in the following examples, the direct command will be shown.

Deny Access at a Lower Level

Case: Subjects need access to Secrets for an environment, but that logical environment contains a more restricted area.

Solution: Two Policies. The first provides the Subjects (*developer1@thycotic.com*|*developer2@thycotic.com*) general access to the Secrets resources at the path `secrets:servers:us-east-1:<.*>`.

Usage

The direct command to create this policy is

```
dsv policy create --path secrets:servers:us-east-1 --actions '<.*>' --desc 'Developer
Policy' --subjects 'users:<developer1@thycotic.com|developer2@thycotic.com>' --effect
allow
```

With the trickiest part being to remember the "secrets" prefix on the path.

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: secrets:servers:us-east-1
permissionDocument:
- id: xxxxxxxxxxxx
description: Developer Policy.
subjects:
- users:<developer1@thycotic.com|developer2@thycotic.com>
actions:
- "<read|delete|create|update>"
effect: allow
resources:
- secrets:servers:us-east-1:<.*>
```

The second Policy adds a specific path at a level lower (*secrets:servers:us-east-1:production*) to explicitly *deny* access to *developer1@thycotic.com*, as in the following example.

The command to create this policy is

```
dsv policy create --path secrets:servers:us-east-1:production --actions '<.*>' --desc
'Developer Deny Policy' --subjects 'users:<developer1@thycotic.com>' --effect deny
```

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: secrets:servers:us-east-1:production
permissionDocument:
- id: xxxxxxxxxxxx
description: Developer Deny Policy.
subjects:
- users:<developer1@thycotic.com>
actions:
- "<.*>"
effect: deny
```

```
resources:
- secrets:servers:us-east-1:production:<.*>
```

Allow Users to Assign Specific Roles to New Clients

Case: A User needs to assign Roles when they create client credentials, but must not be able to self-elevate by assigning an admin level Role.

Solution: Use a naming convention when creating Roles and specify a prefix with a wildcard to only allow Users to assign Roles that match the naming convention, as modeled in the following example.

The command to run this is

```
dsv policy create roles:dev-role --subjects users:developer@thycotic.com,roles:onboarding-
role --desc 'Role Assignment' --resources 'roles:dev-role-<.*>' --actions assign
```

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: roles:dev-role
permissionDocument:
- id: xxxxxxxxxxxx
description: Limited Role Assignment Policy.
subjects:
- users:developer@thycotic.com
- roles:onboarding-role
actions:
- assign
effect: allow
resources:
- roles:dev-role-<.*>
```

Allow User2 Access to User1's Home Vault

Case User2 need access to a secrets space (folder) in User1's Home Vault

Solution: Have an Admim create a policy that enables access. In this example, we assume User1 has a secret in their home vault at: databases/mongo/primary and wants to give User2 read rights to anything under *databases*, but not their entire Home vault

The command the Admin will run to create the policy would be:

```
dsv policy create --path home:users:user1:databases --actions '<read>' --desc 'User2 to
access User1 Home/databases' --subjects 'users:User2' --effect allow
```

Notice the path starts with `home:users:<username>`

When User1 is authenticated and needs to access the secret the command would be `dsv home read databases/mongo/primary`

When User2 is authenticated and needs to access the secret the command would be `dsv home read users:User1/databases/mongo/primary`

Enable a Group to search Secrets

Case: Allow a Group to search secrets

Solution: Under the Resource entity, Secrets, enable the Group named "admins".

The command to create this policy is

```
dsv policy create secrets --subjects groups:admins --desc 'secret search' --resources
secrets --actions list
```

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: secrets
permissionDocument:
- actions:
  - list
  conditions: {}
  description: secret search
  effect: allow
  id: xxxxxxxxxxxx
  meta: null
  resources:
  - secrets
  subjects:
  - groups:admins
version: "0"
```

Note: Searching secrets only enables the users to see the path, but not the actual data in the secret. That would require Read access at the proper path.

Allow Users to List Specific Entities

Case: A User needs to search across all items but only needs full read access on specific ones

Solution: Add a list action and the root of the entity used for searching.

In the example below, *roles* is the entity for reading and searching (list action). In the **resources** section, *roles:dev-role-<.>** is used for reading, while *roles* is used for searching.

The command to create this policy is

```
dsv policy create roles --subjects users:developer@thycotic.com,roles:onboarding-role --
desc 'Role Searching' --resources 'roles:dev-role-<.*>,roles' --actions read,list
```

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: roles
permissionDocument:
- actions:
  - read
  - list
  conditions: {}
  description: Role Searching
  effect: allow
  id: xxxxxxxxxxxx
  meta: null
  resources:
  - roles:dev-role-<.*>
  - roles
  subjects:
  - users:developer@thycotic.com
  - roles:onboarding-role
version: "0"
```

The syntax of the latter is important. In general, the root form of an entity has no * after the entity name, or anything besides the name.

Delegate Policy Authority

Case: An admin wants to delegate control to various team leads at a sub-path.

Solution: Under Resources, add config:policies followed by the resource path.

The command to create this policy is

```
dsv policy create secrets:servers --actions create,read,update,delete --resources
'secrets:servers:<.*>,config:policies:secrets:servers:<.*>' --subjects
'users:<developer1@thycotic.com|developer2@thycotic.com>'
```

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: secrets:servers
```

```
permissionDocument:
- actions:
  - create
  - read
  - update
  - delete
  conditions: {}
  description: ""
  effect: allow
  id: xxxxxxxxxxxx
  meta: nullb
  resources:
  - secrets:servers:<.*>
  - config:policies:secrets:servers:<.*>
  subjects:
  - users:<developer1@thycotic.com|developer2@thycotic.com>
version: "0"
```

Now the developers can create Policies below the *secrets:servers:* path; for example, developer1 can create Policies for *secrets:servers:webservers* and developer2 can do the same at *secrets:servers:databases*.

Read Audits

Case: A user needs to be able to read audit records

Solution: Add a policy for the audit resource path

The command to create this policy is

```
dsv policy create audit --actions list --resources audit --subjects
users:developer1@thycotic.com
```

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: audit
permissionDocument:
- actions:
  - list
  conditions: {}
  description: ""
  effect: allow
  id: xxxxxxxxxxxx
  meta: null
  resources:
  - audit
  subjects:
```

```
  - users:developer1@thycotic.com
version: "0"
```

Manage An Auth Provider

Case: A user needs to update a single auth provider

Solution: Add a policy for the config:auth provider path

The command to create this policy is

```
dsv policy create config:auth:gcp-dev --actions read,update --resources config:auth:gcp-
dev --subjects users:developer1@thycotic.com
```

```
created: '2020-06-24T18:12:26Z'
createdBy: users:thy-one:admin@company.com
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
lastModified: '2020-07-16T20:13:53Z'
lastModifiedBy: users:thy-one:admin@company.com
path: config:auth:gcp-dev
permissionDocument:
- actions:
  - read
  - update
  conditions: {}
  description: ""
  effect: allow
  id: xxxxxxxxxxxx
  meta: null
  resources:
  - config:auth:gcp-dev
  subjects:
  - users:developer1@thycotic.com
version: "0"
```

Create Reports

Case: A user needs to be able to read reports

Solution: Add a policy for the reports:query resource path

The command to create this policy is

```
dsv policy create --path report:query --subjects users:user1@organization.com --actions
create --effect allow --resources report:query
```

```
{
  "path": "report:query",
  "permissionDocument": [
    {
      "actions": ["create"],
      "conditions": {},
      "description": "",
      "effect": "allow",
      "id": "c23f8...h0hfgg",
      "meta": null,
      "resources": ["report:query"],
      "subjects": ["users:user1@organization.com"]
    }
  ],
  "version": "0"
}
```

Admin Policy and Auth Providers

In this section we will

- Define the Default Admin Policy
- Show settings for third-party authentication providers including Thycotic One, AWS, Azure, or GCP.

Commands that Act on Configuration

Command	Action
read	view the current configuration
edit	modify the configuration in an OS-native text editor such as VI, nano, or Notepad
update	upload a superseding configuration document

Read

To read out the current config, which contains the Admin policies

```
dsv config read
```

Note: In this command the `--encoding yaml` flag could be used to provide the output in YAML format.

In response, you should see a block of code containing the Default Admin Policy, similar to this.

```
{
  "created": "2019-09-18T18:38:49Z",
```

```
  "createdBy": "system",
  "lastModified": "2020-07-30T23:56:56Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "permissionDocument": [
    {
      "actions": ["<.*>"],
      "conditions": {},
      "description": "Default Admin Permissions",
      "effect": "allow",
      "id": "bm17jee33m1c72u313tg",
      "meta": null,
      "resources": ["<.*>"],
      "subjects": ["users:<thy-one:admin@company.com>"]
    },
    {
      "actions": ["<.*>"],
      "conditions": {},
      "description": "Default Deny Home Permissions",
      "effect": "deny",
      "id": "bsd72rfe1vkc72up3o1g",
      "meta": null,
      "resources": ["home:<.*>"],
      "subjects": ["users:<thy-one:admin@company.com>"]
    }
  ],
  "tenantName": "company",
  "version": "1"
}
```


The initial User possesses full administrator rights and is federated through Thycotic One. This is indicated by the `thy-one` prefix on the users's email. This enables self-service password reset through Thycotic One.

In keeping with best practices, you should set up a less privileged User policy for routine use. The highly privileged initial Admin account should be used only when a task requires its privileges.

The first section of the Admin policy with the description "Default Admin Permission" is what allows the Admin full rights to everything in DSV.

The second section with the description "Default Deny Home Permissions" denies the Admin permission to access the Home feature where users have a place for their own secrets. If required, the Admin can remove his/her name and then get access to the Home secrets (API only in Beta)

Edit

Note: Delinea recommends against changing the Default Admin Policy other than to add a User as a back-up admin. Even then, best practices would be to create a separate policy for specific access for Users.

Note: For adding and editing policies beyond the Default Admin Policy, see the Policy article.

Note: Delinea recommends against changing the Thycotic One provider because it provides for the initial User and any others you add that federate to Thycotic One. However, you can add providers.

Use *edit* to open your configuration in the OS's default editor (typically **VI**, **nano**, or **Notepad**).

```
dsv config edit --encoding YAML
```

The editor directly updates the configuration in the vault when you save your work.

Update

Use *update* to change a config by uploading JSON data.

The value of the `--data` parameter for *update* accepts JSON entered directly at the command line, or the path to a JSON file.

```
dsv config update --data '{"tenantName":"company", ...}'
```

or

```
dsv config update --data @configfilename.json
```

Grant Admin Access Rights to All Home Vaults

If it is rquired that the Admin have access to all individual Home vaults, then edit the Home Vault Permissions and change the *effect* field to "allow"

```
dsv config edit --encoding YAML
```

The editor will open the OS default editor and you can modify the *effect* field.

Authentication Providers

Add an Authentication Provider

The general command to add an Authentication Provider is:

```
dsv config auth-provider create --name <name> --type <type> --<properties>
```

in which:

- name is the friendly name used in DSV to reference this provider. It is separate from `type` because it allows multiple auth providers of the same type (for example several AWS accounts).
- type is the authentication provider type; valid values are aws, azure, gcp and thycoticone
- properties are configuration settings specific to the authentication provider

Usage

- AWS flag is *--aws-account-id*
- Azure flag is *--azure-tenant-id*
- Thycotic One requires three flags *--baseURI*, *--clientID*, and *--clientSecret*
- GCP has two options for federation, GCE metadata and service accounts.
  - For GCE metadata, use *--gcp-projcet-id*
  - Flags are not provided for a service account so a file is required.

Note: The account identifiers for third-party authentication are a top level setting that allow you or other Users to authorize specific security principals within that account. They do not automatically grant access to any User or Role within the provider.

See the Authentication section for examples of using AWS, Azure, GCP, and Thycotic One for authentication.

To see a list of all Auth-providers:

```
dsv config auth-provider search
```

Initially, your tenant will only have a Thycotic One connection

```
{
  "created": "2019-09-18T18:38:49Z",
  "createdBy": "",
  "id": "bm17jee33m1c72u313u0",
  "lastModified": "2020-05-10T02:25:04Z",
  "lastModifiedBy": "users:admin@company.com",
  "name": "thy-one",
  "properties": {
    "baseUri": "https://login.thycotic.com/",
    "clientId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "clientSecret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "sendWelcomeEmail":false
  },
  "type": "thycoticone",
  "version": "1"
}
```

Edit an Authentication Provider

Make changes to an authentication provider using the `edit` command and the name (path) of the authentication provider:

```
dsv config auth-provider edit <name>
```

Audit Command

DSV audit logs can be searched with the `dsv audit` command followed by the **required** `--startdate YYYY-MM-DD` flag.

Flags

Flag	Function	Example
`--actions`	Searches within audit logs for a specific CRUD action. Use the values: POST, GET, PUT, PATCH, DELETE. If omitted, all actions will return.	`--actions PUT`
`--cursor`	A cursor value is given when the number of events returned exceed the display limit. Include the returned cursor value in the next query to continue viewing the log. See below for example usage.	
`--enddate`	Along with `--startdate`, sets the time frame for search. If omitted, `enddate` will return all events from the startdate to the search date. **Make sure to use the YYYY-MM-DD format. You must include a zero before single-digit dates.**	`--enddate 2021-02-01`
`--limit`	Sets the maximum number of results per cursor. If omitted, `limit` will default to 25.	`--limit 10`
`--path`	Searches for actions within a given path. If omitted, all `paths` will return.	
`--principal`	Searches for a specific principal or user within DSV. If omitted, all principals will return.	`--principal users:thy-one:your.username@organization.com`
`--startdate`	Along with `--enddate`, sets the time frame for search. **Make sure to use the YYYY-MM-DD format. You must include a zero before single-digit dates.** This flag is required.	`--startdate 2020-08-21`

Usage Examples

Basic Unfiltered Query

```
dsv audit --startdate 2021-01-01
```

An audit log of all actions in every path from January 1st, 2021 to the present date is returned.

Simple Limited Query

```
dsv audit --startdate 2021-01-01 --enddate 2021-02-02 --limit 5
```

An audit log of the most recent five actions in every path from January 1st, 2021 to February 2nd, 2021 is returned.

Cursor Query

If the logs are longer than the limit, the CLI will return a long `--cursor` string. Copy the cursor value and repeat the previous input with the addition of `--cursor <returned string>` to continue listing the logs.

Initial Input Returning a Cursor Value

```
dsv audit --startdate 2021-01-01 --enddate 2021-02-02 --limit 2
```

Example Output Returning Cursor


```
"cursor": "MGJiYmYxZmItZjlhMS00NjY1LWEyN2YtNDgwM2E3MjExMjRh.AT0HNoBK4m4rE_
XkhWoXImQyjbX8hrSHQiXM06qRIQ8KgZAU21Kdb-bmur6kK85N34z2e5LEhSoEIAV3a5bhgkFbE5a9W78iwg",
    "data": [
    {
      "action": "POST",
      "created": "2021-01-26T16:48:53.502387353Z",
      "id": "",
      "ipaddress": "11.111.11.***",
      "message": "user attempting login: example@thycotic.com",
      "path": "token",
      "principal": "",
      "principalItemId": "",
      "status": 0,
      "tenant": "tenantIDstring",
      "tenantName": "yourorg"
    },
    {
      "action": "POST",
      "created": "2021-01-26T16:48:53.839114046Z",
      "id": "",
      "ipaddress": "11.111.11.***",
      "message": "unable to find provider with specified name.",
      "path": "token",
      "principal": "",
      "principalItemId": "",
      "status": 0,
      "tenant": "tenantIDstring",
      "tenantName": "yourorg"
```

Example Input with Cursor

```
dsv audit --stardate 2021-01-01 --enddate 2021-02-02 --limit 2 --cursor
MGJiYmYxZmItZjlhMS00NjY1LWEyN2YtNDgwM2E3MjExMjRh.AT0HNoBK4m4rE_
XkhWoXImQyjbX8hrSHQiXMO6qRIQ8KgZAU21Kdb-bmur6kK85N34z2e5LEhSoEIAV3a5bhgkFbE5a9W78iwg
```

Report Command

The **report** command acts on secrets and groups. All users can generate a report containing their own secrets and groups. Only **administrators** and users with a **policy allowing access to reports/query** can generate a report for other users. See **Create Reports** in policy for an example.

Secret Reporting

Use the **secret** subcommand to retrieve a list of secrets and secret actions available to a user, group, or role. Running a secret report without flags will generate a list of every secret and action available to the user running the query. Secrets are sorted by the most recent modification.

Command/Flags	Function	Example
`report secret`	Retrieves the secrets and secret actions available to a user, group, or role.	`dsv report secret`
`--group`	Searches for secrets available to a specified group.	`dsv report secret --group engineers`
`--path`	Searches for available secrets within a specified path.	`dsv report secret --path us-east/server01:<.*>`
`--role`	Searches for secrets available to a specified role.	`dsv report secret --role automation`
`--user`	Searches for secrets available to a specified user.	`dsv report secret --user john`
`--limit`	Sets the number of retrieved secrets.	`dsv report secret --limit 25`

Example Secret Queries

Personal Secret Query

The following input will return a list of the secrets available to the user performing the query.

Input:

```
dsv report secret
```

Output:

Usage

```
{
  "data": {
    "UserName": "user",
    "Provider": "thy-one",
    "Created": "2021-01-11T16:07:59Z",
    "LastModified": "2021-01-11T16:07:59Z",
    "CreatedBy": "",
    "LastModifiedBy": "",
    "Version": "0",
    "Secrets": [
      {
        "Actions": ["<.*>"],
        "ID": "",
        "Path": "us-east/server01",
        "Created": "2021-03-25T13:12:13Z",
        "LastModified": "2021-03-25T13:12:13Z",
        "LastModifiedBy": "users:thy-one:user@organization.com",
        "Version": "0"
      }
    ],
    "Home": []
  }
}
```

User Secret Query

The following input will return a list of secrets available to the specified user. Note that this query is only available to administrators and users with **reports/query** permission.

Input:

```
{
  dsv report secret --user john
}
```

Output:

```
{
  "data": {
    "UserName": "john",
    "Provider": "thy-one",
    "Created": "2021-01-11T16:07:59Z",
    "LastModified": "2021-01-11T16:07:59Z",
    "CreatedBy": "",
    "LastModifiedBy": "",
    "Version": "0",
    "Secrets": [
      {
```

```
        "Actions": ["<.*>"],
        "ID": "",
        "Path": "us-east/server01",
        "Created": "2021-03-25T13:12:13Z",
        "LastModified": "2021-03-25T13:12:13Z",
        "LastModifiedBy": "users:thy-one:john@example.com",
        "Version": "0"
      }
    ],
    "Home": []
  }
}
```

Group Reporting

Use the **group** subcommand to retrieve a list of groups associated with a user or role. Running a group report without flags will generate a list of groups associated with the user running the query.

Command/Flags	Function	Example
`report group`	Retrieves a list of groups associated with a user or role.	`dsv report group`
`--role`	Searches for the groups associated with a specified role.	`dsv report group --role automation`
`--user`	Searches for the group memberships of a specified user.	`dsv report group --user john`
`--limit`	Sets the number of retrieved groups.	`dsv report group --limit 25`

Example Group Queries

Personal Group Query

The following input will return a list of the groups to which the user performing the query belongs.

Input:

```
dsv report group
```

Output:

```
{
  "data": {
    "UserName": "user",
```

```
    "Provider": "thy-one",
    "Created": "2021-01-11T16:07:59Z",
    "LastModified": "2021-01-11T16:07:59Z",
    "CreatedBy": "",
    "LastModifiedBy": "",
    "Version": "0",
    "group": [
      {
        "Name": "accountadmins",
        "Since": ""
      }
    ]
  }
}
```

User Group Query

The following input will return a list of groups to which the specified user belongs. Note that this query is only available to administrators and users with **reports/query** permission.

Input:

```
{
  dsv report group --user john
}
```

Output:

```
{
  "data": {
    "UserName": "john",
    "Provider": "thy-one",
    "Created": "2021-01-11T16:07:59Z",
    "LastModified": "2021-01-11T16:07:59Z",
    "CreatedBy": "",
    "LastModifiedBy": "",
    "Version": "0",
    "group": [
      {
        "Name": "engineers",
        "Since": ""
      }
    ]
  }
}
```

Home Vault

Home provides Users with a separate space to store secrets. No Users can access another User's Home values. As soon as a User is created in DSV, they are given access to their own Home vault without an explicit policy granting access.

The Home value will list a path like `users:<username>:<secretname>` DSV will determine which username based on whomever authenticated. So if joesmith@company.com authenticates, then a creates a Home value, that vaule will be in Joe Smith's Home vault.

Even the Admin does not have access by default, though they can give themselves access for "breakglass" purposes. If the admin is given access to read users' Home values, it can only be done through the API in the Beta version.

Home follows the familiar syntax: `dsv home (command) (flags and parameters)` with the commands being `create, read, delete, update, describe, edit, search` The difference between `read` and `describe` is that read shows both data and metadata, while describe only shows metadata.

Examples

Create

The `create` command uses the `--data` flag to pass data into the secret. This flag accepts JSON entered directly into the command line or by a path (absolute or relative) to a JSON file.

Bash examples

```
dsv home create secret1 --data '{"username":"administrator","password":"bash-secret"}'
```

```
dsv home create secret2 --data @/home/user/secret.json
```

```
dsv home create secret2 --data @../secret.json
```

Powershell examples

```
PS C:> dsv home create --path secret1 --data '
{\"username\":\"administrator\",\"password\":\"powershell-secret\"}'
```

```
dsv home create secret2 --data '@/home/user/secret.json'
```

```
dsv home create secret2 --data '@../secret.json'
```

Usage

CMD Examples

```
PS C:> dsv home create secret1 --data "{\"username\":\"administrator\",\"password\":\"cmd-secret\"}"
```

```
dsv home create secret2 --data @/home/user/secret.json
```

```
dsv home create secret2 --data @../secret.json
```

The `--attributes` flag can be used to add user-defined metadata in the same way that data is added.

The `--desc` flag can be used to add a simple string. If the string has any spaces, then it should be enclosed in double quotes.

As a Bash example:

```
dsv home create secret1 --attributes '{"priority":"high"}'  --desc "Covert Secret" --data '{"username":"administrator","password":"bash-secret"}'
```

Update

`update` is similar to `create` but operates on an existing Home value. Only the specified values change unless the `--overwrite' flag is used, in which case all unspecified values are deleted.

If you have this Home value:

```
{
  "attributes": {
    "attr": "add one"
  },
  "created": "2019-09-20T16:12:57Z",
  "createdBy": "users:user@company.com",
  "data": {
    "host": "server01",
    "password": "badpassword"
  },
  "description": "update description",
  "id": "c893b4f8-9425-4fa4-acbf-2806d6f1fa82",
  "lastModified": "2020-01-17T15:43:27Z",
  "lastModifiedBy": "users:dsv-one:admin@company.com",
  "path": "users:user@company.com:secret1",
  "version": "12"
}
```

This Bash command will only change the value for *host* in the data section.

Usage

```
dsv home update secret1 --data '{\"host\":\"unknown\"}'
```

```
{
  "attributes": {
    "attr": "add one"
  },
  "created": "2019-09-20T16:12:57Z",
  "createdBy": "users:user@company.com",
  "data": {
    "host": "unknown",
    "password": "badpassword"
  },
  "description": "update description",
  "id": "c893b4f8-9425-4fa4-acbf-2806d6f1fa82",
  "lastModified": "2020-08-03T17:58:29Z",
  "lastModifiedBy": "users:user@company.com",
  "path": "users:user@company.com:secret1",
  "version": "13"
}
```

The flag `--overwrite`, if added to the above command would wipe-out the description and any other data KV pairs. So this flag requires caution.

```
dsv home update secret1 --data '{\"host\":\"unknown\"}' --overwrite
```

Read

The `read` command shows both the Secret data and metadata.

```
dsv home read secret1
```

Flags

`--encoding` or `-e` converts the output to JSON (default) or YAML.

`--out` or `-o` can send the read response to `stdout` (default), the clipboard (`clip`), or a file (`file:<filename>`)

`--filter` or `-f` filters to a specific KV pair. So data.password would only output the password value.

This example would send the password value only to the clipboard.

```
dsv home read secret2 -o clip -f data.password
```

Describe

The command `describe` only shows the metadata of a Home value

Usage

```
dsv home describe secret1
```

Search

You can search for Secrets by path or attribute

Some examples

```
dsv home search server

dsv home search --query server

dsv home search --query aws --search-field attributes.type

dsv home search --query 900 --search-field attributes.ttl --search-type number

dsv home search --query production --search-field attributes.stage --search-comparison
equal
```

flags

`--query`, `-q` Query of secrets to fetch (required)

`--limit` Set the maximum number of search results that will display per page (cursor)

`--cursor` Accepts the element used to get the next page of results

`--search-comparison` Specify the operator for advanced field searching, can be 'contains', 'equal', or 'begins_with' Defaults to 'contains' (optional)

`--search-field` Advanced search on a secret field such as 'attribute.type' or 'description'. Defaults to 'path'. (optional)

`--search-type` Specify the value type for advanced field searching, can be 'number' or 'string'. Defaults to 'string' (optional)

For a search where there are more results than returned in the first set, the API returns a cursor—a large piece of text. You pass that back to get the next set of results.

For example, if the command `dsv secret search -q admin --limit 10` matched 12 Secrets with admin in the name, the CLI would return the first 10 plus a cursor. To obtain the next two results, you would use this command:

```
dsv secret search -q admin --limit 10 --cursor AFSDFSD...DKFJLSDJ=
```

Cursors may be lengthy:


```
dsv secret search -q resources --limit 10 --cursor
eyJpZCI6ImEwOTFjOWIzLWE4MmQtNGRiYy1hYThiLTYxMDY0NDZhZjA3MSIsInBhdGgiOiIiLCJ2ZXJzaW9uIjoidi
1jdXJyZW5OIiwidHlwZSI6IiIsImxhdGVzdCI6MH0=
```

Edit

Use `edit` to open the Secret data in the default text editor for bash, such as **vi**, **nano**, or **Notepad**.

- Saving in the editor updates the Secret in the vault, except in the case of Notepad, in which case the update happens when you save and then exit Notepad. Your interim saves are to the working copy.

```
dsv home edit --path us-east/server02
```

Delete

To `delete` a Home value, simply specify its name.

```
dsv home delete secret1
```

When you delete a Secret, it will no longer be usable. However, with the soft delete capacity of DSV, you have 72 hours to use the *restore* command to undelete the Secret. After 72 hours, the Secret will no longer be retrievable.

Should you want to perform a hard delete, precluding any restore operation, you can use the *delete* command's `--force` flag.

Restore

The `delete` command is a soft delete for about 72 hours before the delete become permanent. During that time, the secret can be brought back using the `restore` command. After the ~72 hours, the secret is permanently deleted and can't be restored.

```
dsv home restore secret1
```

GetByVersion

The `--version` flag determines how many past versions are displayed along with the current version.

```
dsv home secret1 --version 3
```

Rollback

To return a secret to a past version, use the `rollback` command and a `--version` flag to determine which version to return to. The original version is 0.

```
dsv home rollback secret1 --version 2
```

DSV UI Reference

DevOps Secrets Vault provides a user interface (UI) for viewing Home Vaults and Shared Vaults, as well as managing users, user groups, and roles defined in your DevOps Secrets Vault tenant. Refer to the Quick Start for signing up for a tenant.

The DSV has been refreshed with a new design and navigation to enhance user experience. Learn more about the Delinea experience here.

> **Note:** Functionality for creating, updating, and deleting UI objects is dependent on the permission granted for the user's role.

This section provides instructions for:

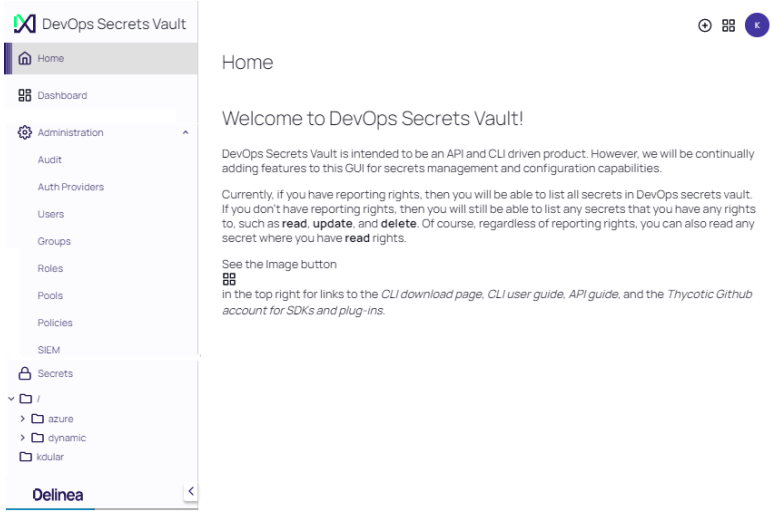
- Viewing Vaults
- Viewing Engines and Pools
- Viewing, Creating, and Deleting Secrets
- Viewing, Creating, and Deleting Users
- Viewing, Creating, and Deleting Groups and their Members
- Viewing, Creating, and Deleting a Role

Navigating the UI

The UI consists of the following functional areas:

- Left Navigation Panel - provides a fixed reference for accessing DSV functionality. Click **Administration** to mange DSV users, user groups, and roles. Secrets are managed in either a Shared Vault (team access) or a Home Vault (private access).
- Content Container - this main central area of the page updates with details for the selected feature.
- Four Square Icon - accesses additional features for DSV that include links to: **CLI Download Page**, **User Guide**, **REST API Guide**, and **Delinea GitHub (SDKs & Plugins)**.
- Create Secret (**+**) - allows the creation of a new secret.
- User Profile - accesses general information that includes and controls for adjusting the look and feel of the UI, a

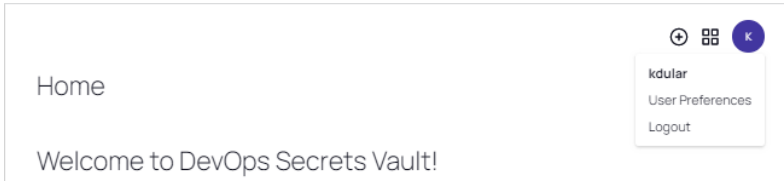
change password feature, date and time settings, and logout.



Customizing the UI

To adjust the look and feel of the UI:

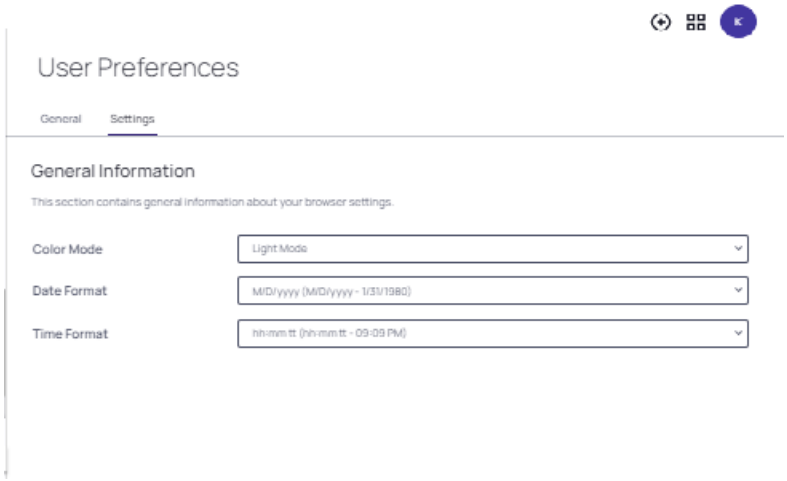
1. Click your User Profile and select **User Preferrences**.
2. On the User Preferences page, select the **Settings** tab.



3. Adjust any of the following UI controls:

- **Color Mode** can be switched between **Light Mode** and **Dark Mode**.
- **Date Format** can be set as a US (M/D/yyy) or international (yyy.MM.dd) format.

- **Time Format** allows selection of the time format (hh.mm.tt, H:mm).



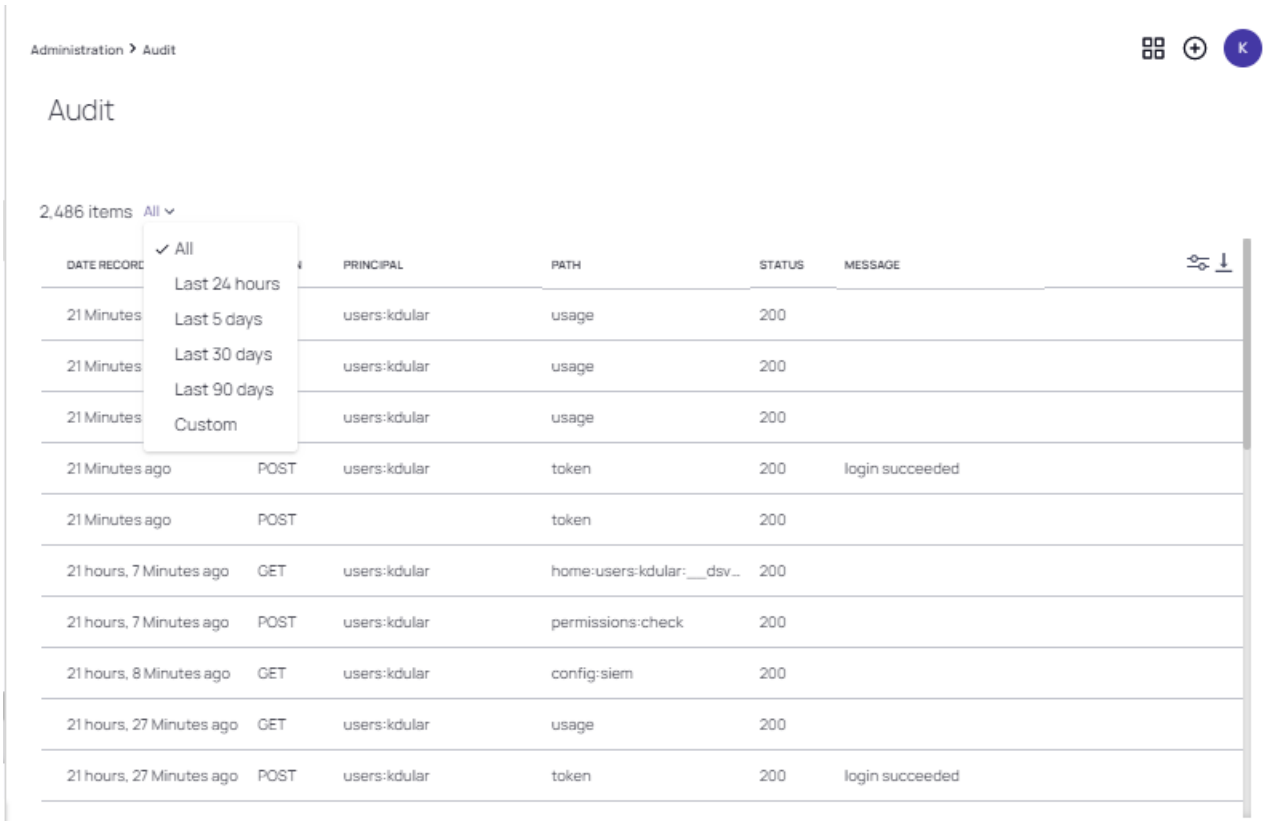
Audit

The Audit page displays details for each action in the application, including: date recorded, the type of action (**GET** or **PUT**), the user associated with the action (**Principal**), the type of action or area of interaction with the application (**Path**), the status code produced, and any message returned.

> **Note:** Depending on the permissions granted to your role, audit details displayed may be limited to your user, or include other users.

Use the interval drop-dow in the chart header to adjust the data displayed to a specific range of days. The **items** count at the top of the chart indicated the total number of audit items in the selected interval.

Usage

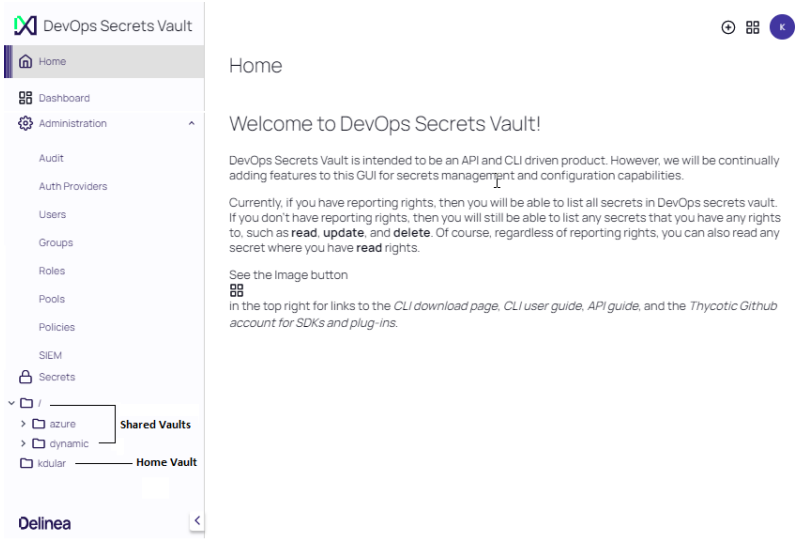


Viewing Vaults

Users are able to view Shared Vaults, as well as their Home Vault in the left navigation panel, displayed under **Secrets**, at a top level.

The display of both Home Vaults and the Secrets folder is optimized in the left navigation panel. When expanding the Secrets folder, the Home Vaults folder automatically collapses to avoid confusion between the two folders and maximize space for the subfolders.

Select a vault to view the path created for its secrets.

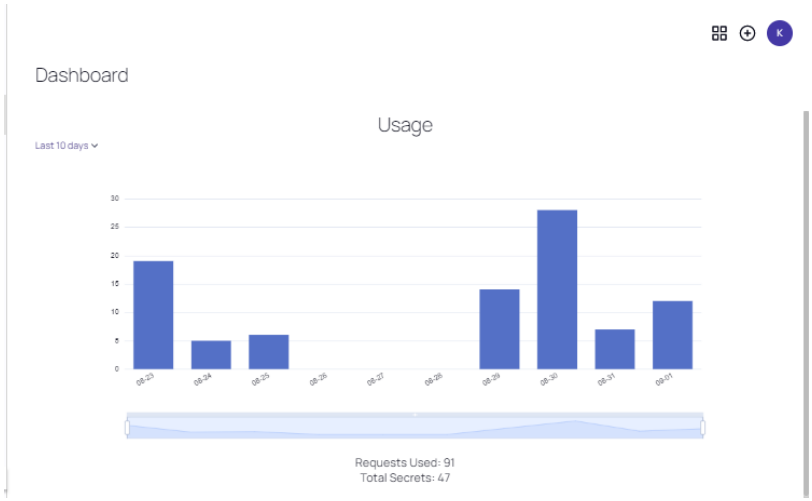


Select any path created for its secrets. Refer to Secrets for detailed secret functionality.

Dashboard

The dashboard, available from the left navigation panel, provides a real-time view of request processing over time, as well as the total secrets stored across all secrets vaults.

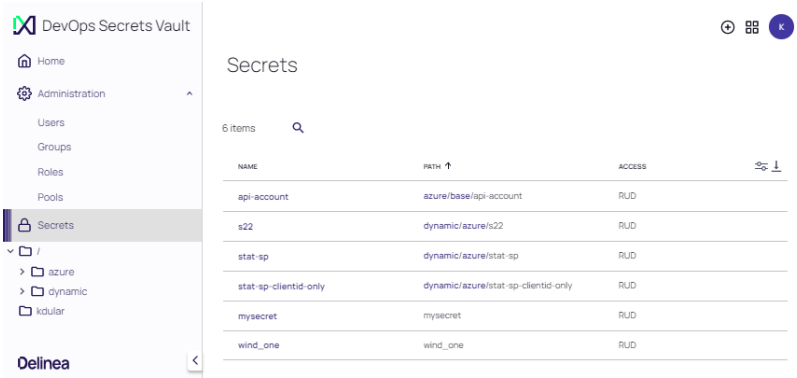
Select a time interval at the pull-down (**Last 10 days**, **Last 30 days**, **Lst 60 days**, **Last 90 days**, or **Custom**), to view individual requests/day in the given interval. Use the interactive slider below the chart to focus on a specific date range within the selected interval.



Secrets

Users are able to view their secrets in the left navigation panel. Click **Secrets** for a list view of all secrets in the application. The name, path and access rights to the secret are displayed. Access rights include **R** (read), **U** (update) and **D** (delete).

Note: Secrets can also be accessed from their respective vaults. Refer to Viewing Vaults.

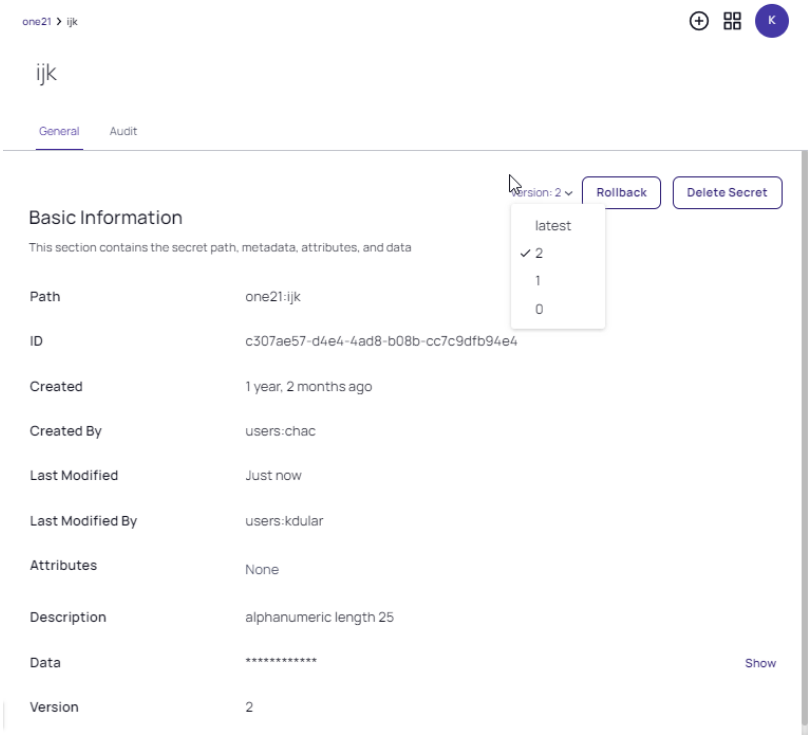


Viewing Secrets Metadata

Click any secret to display the metadata for the secret. Metadata includes: the ID and path name, any attributes defined, version, as well as the times and dates when secrets were created or last modified by a user.

Note: The Update permission is required for an account, in order to edit any of the values for a secret.

Hover over the **Created** and **Last Modified** fields to see the exact date and time the action was performed.



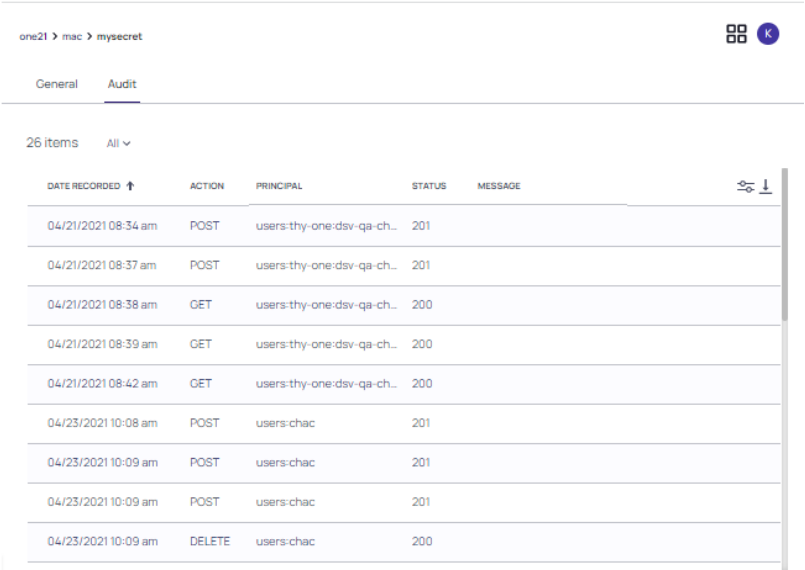
Rolling Back a Secret's Version

Any editable parameter for a secret can be updated. When updated, a new version of the secret is created (0, 1, 2 etc.).

The version of any secret can be selected and instituted as the current version. To do so, select the desired version at the Version pulldown, then click **Rollback**.

Accessing Audit Details

Click **Audit** to access the audit trail for the secrets.



Audit details include the following information:

Parameter	Value
DATE RECORDED	The date and time an action was taken.
ACTION	The action performed for the secret as either: PUT or GET.
PRINCIPAL	The user performing the action.
STATUS	The HTTP status code returned to the caller as the result of the action.
MESSAGE	Any message text created when the action was performed.

Creating and Deleting Secrets

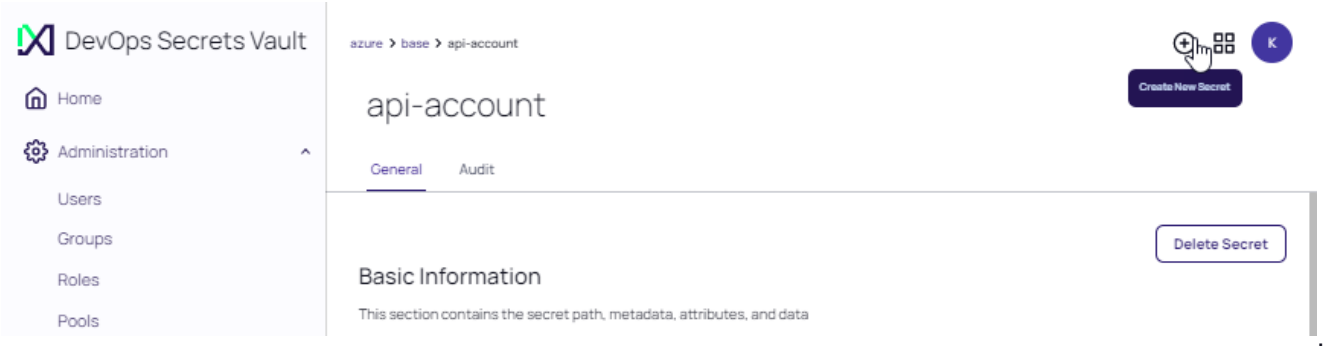
Note: The Create and Delete permission are required for an account, in order to create or delete a Role.

Secrets are deleted from the General tab for that secret's metadata. Click **Delete**, then click **Delete** again at the confirmation prompt.

Usage

Secrets are created at any page in the application, using the Add icon (**+**)in the top right corner. Click **+** and supply the requested parameters for the new secret at the Create New Secret dialog, then click **Save**.

> **Note:** Additionally, the **Create New Secret** button is available at the folder level of any Shared or Home Vault.



Parameters for creating a new secret include:

Parameter	Value
Save To	The name of the vault where the secret is saved.
Path	The path to the secret in that vault.
Data	Any data defined for the secret by a JSON string. Refer to secrets in the CLI Command Reference.
Attributes	The attributes defined for the secret by a JSON string. Refer to secrets in the CLI Command Reference.
Description	Narrative information that identifies the secret.

Auth Providers

Select **AuthProviders** in the left navigation panel to obtain the AuthProviders page. This page lists the currently defined authentication providers for use in the application. The name and dates the provider was created and modified is shown, along with the type of provider. **TYPE** can include: certificate, ThycoticOne, Azure and AWS.

Usage

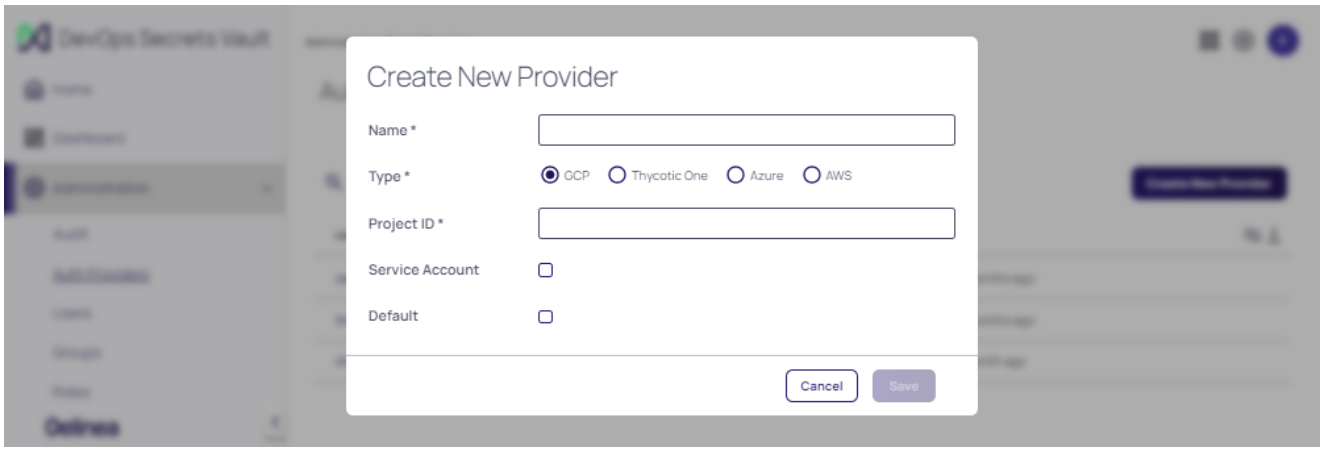
Downloading AuthProvider Information

Click the download control in the top right of the list to download a CSV file that contains the parameters of the currently defined authentication providers. Provide a name and select a date format for the information, then click **Download**.

Create a New Authentication Provider

Click **Create** in the top right of the list. Provide the following parameters for the new authentication provider, then click **Create**.

Parameter	Definition
Name	The label used to identify the authentication provider in the application.
Type	The type of authentication used with the provider as GCP (authentication using a General Certificate Provider), ThycoticOne (authentication using OIDC compliant authentication providers configured to work with Thycotic One account credentials), Azure (authentication using Azure account credentials), and AWS (authentication using Amazon Web Services credentials).



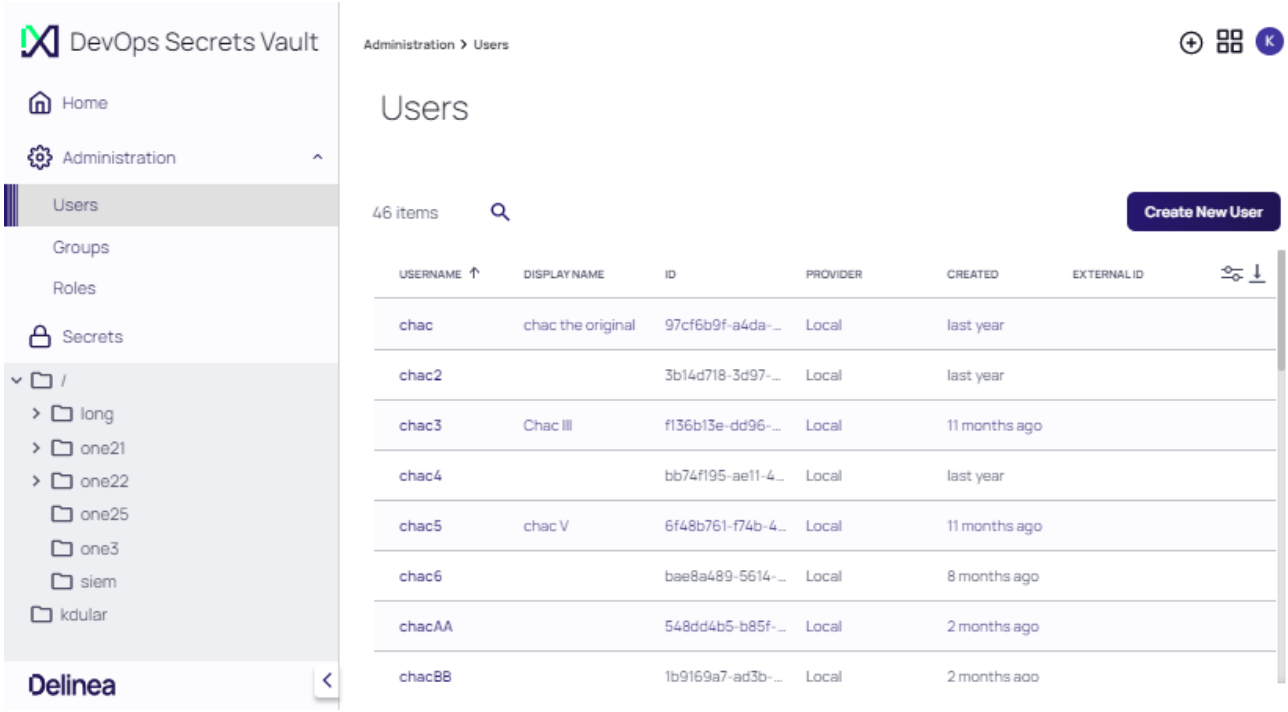
Users

Sign into your DSV tenant. On the Home page, open the **Administration** drop-down to access **Users**.

Viewing Users

At the **Administration** drop-down, select **Users**. A table of the currently defined Users is displayed.

Use the filter icon at the far right of the table header to enable or disable columns included in the table.

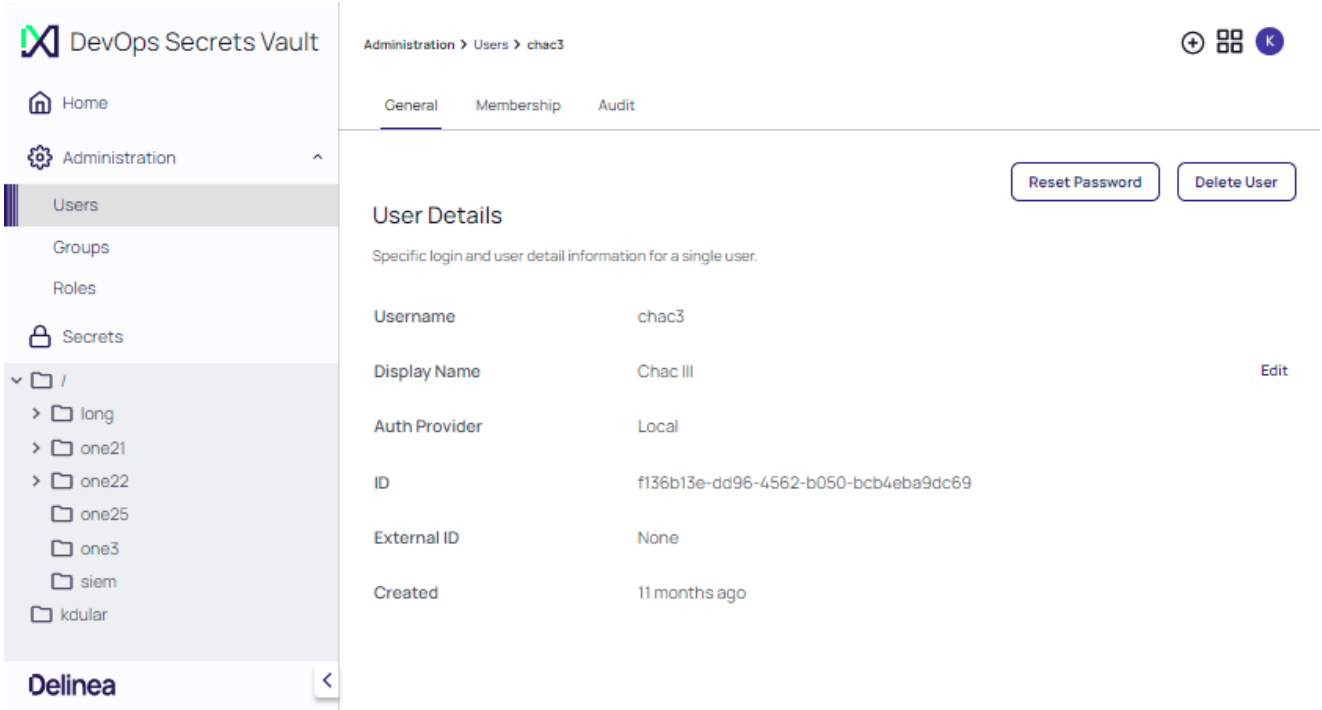


The following parameters are displayed for each User account.

Usage

Parameter	Value
USERNAME	Local username; required; supports local authentication by username and password; need not match that used by a federated authentication provider (if present)
DISPLAY NAME	Locally used display name for identifying the User in DSV
ID	Unique identifier used for this User
PROVIDER	Matches the name attribute of the authentication provider in the settings section of the config
CREATED	When the User account was created
EXTERNAL ID	Identifier recognized by third-party federated authentication providers, such as AWS or ARN

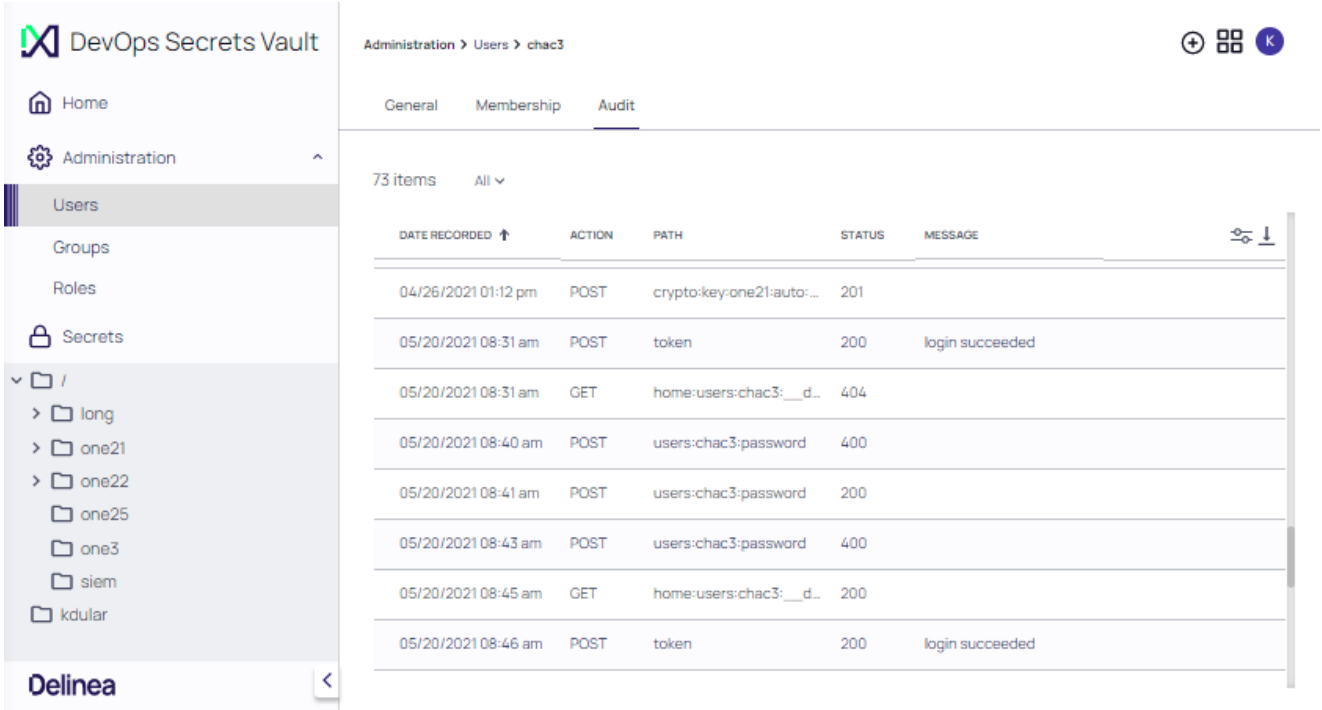
Click any User account in the table to access the User Details page for that User. The User Details page provides a **Reset Password** and **Delete User** option, as well as the ability to edit the **Display Name**.



By default the **General** tab displays for a selected user. The information on this tab presents specific login and user detail information. Click **Membership** to display any User Groups that the user belongs to.

Click **Audit** to display the audit log for the user account. Audit details include the following information:

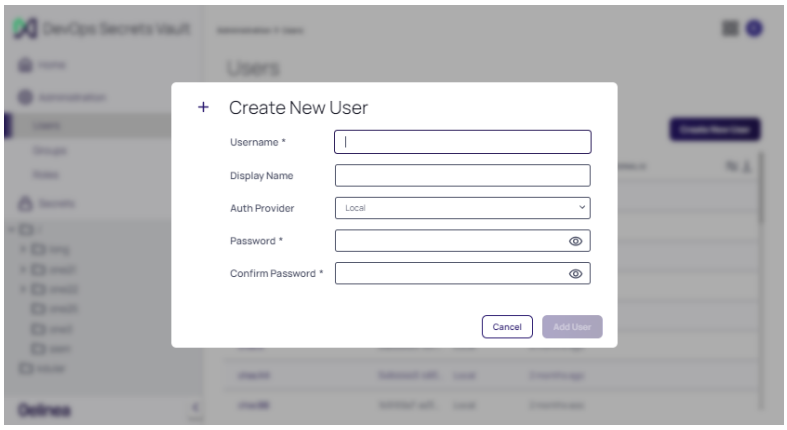
Parameter	Value
DATE RECORDED	The date and time an action was taken
ACTION	The action performed for the user as either: POST or GET
PATH	The path to where the user account is located
STATUS	The HTTP status code returned to the caller as the result of the action
MESSAGE	Any message text created when the action was performed



Creating Users

To create a User:

1. At the Administration > Users page, click **Create User**.
2. Enter values for the requested fields and click **Add User**. Required fields are indicated by an asterisk (*).



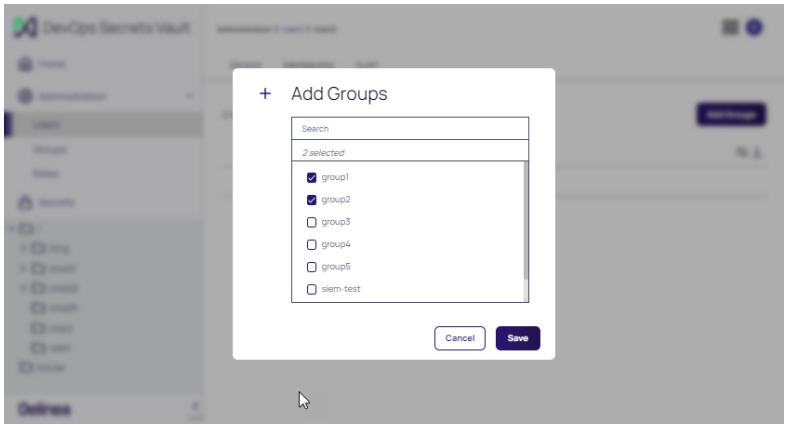
3. The User Details page for the newly created User is displayed.
4. Click **Administration** > **Users** to return to the Users page.

Assigning Group Membership

Any User account can be associated with a Group. A Group determines the policies and permissions enabled for the User. The Group Memberships available for selection for a User are predefined by the Administrator.

To create a Group Membership:

1. At the Administration > Users page, click the User account to be associated with a Group.
2. Click the **Membership** tab.
3. Click **Add Groups**.
4. At the **Add Groups** modal, select the desired Group Memberships. One or more Memberships can be selected. If Memberships already exist for a User, they can be disabled if needed. Click **Save** when complete.



5. The page is updated with User Group Memberships.

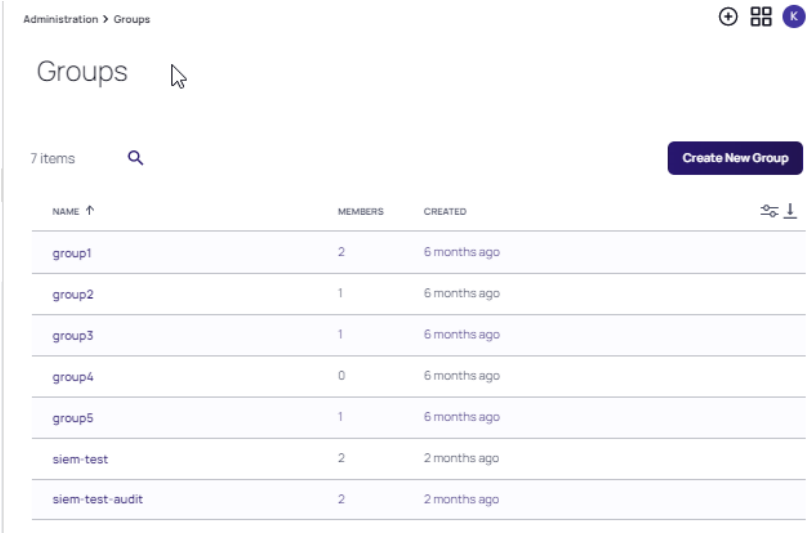
Groups

Sign into your DSV tenant. On the Home page, open the **Administration** drop-down to access **Groups**.

Viewing Groups

At the **Administration** drop-down, select **Groups**. A table of the currently defined Groups is displayed.

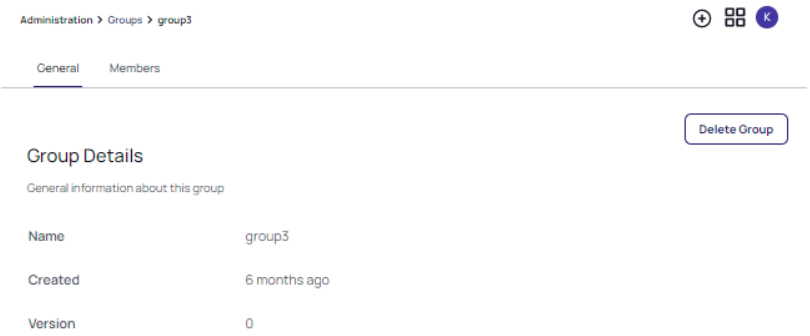
Use the filter icon at the far right of the table header to enable or disable columns included in the table. The search icon can be used to identify a specific Group for display.



The following parameters are displayed for each Group account.

Parameter	Value
NAME	Locally used display name for identifying the Group
MEMBERS	The number of Members in the group
CREATED	When the Group was created

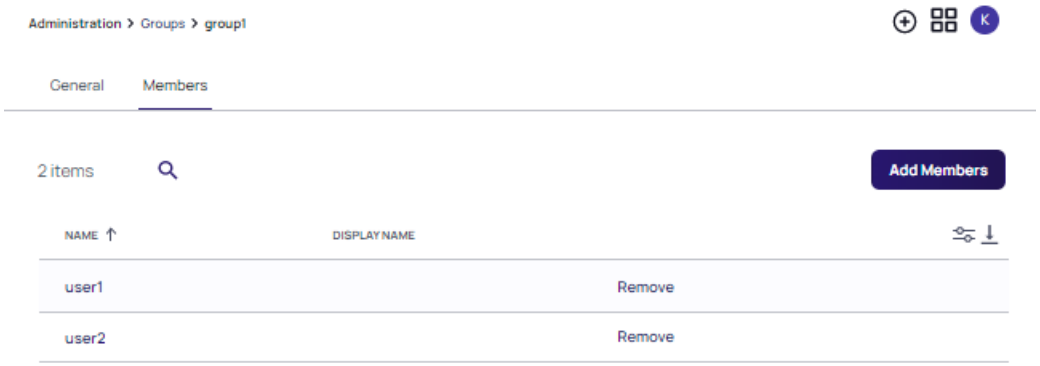
Click any Group in the table to access the Group Details page. Details include the name, date created and version for the group.



Managing the Members in a Group

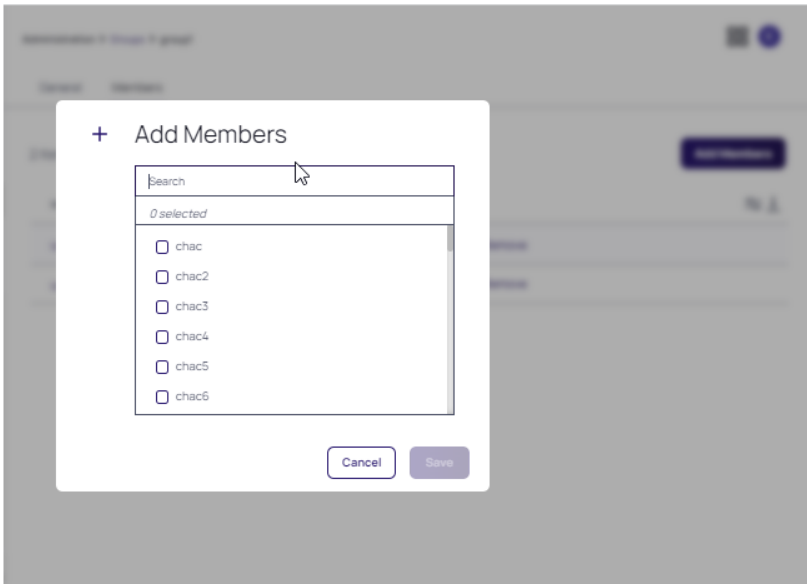
On the Group details page, click the **Members** tab to view the Members currently assigned to the Group. Use the filter icon at the far right of the table header to enable or disable columns included in the table. The search icon can be used to identify a specific Member for display.

The **Members** tab also allows Members to be added or removed.



To delete a Member from the Group, click **Remove**. The Member list is updated with the selected Members removed,

To add a Member to the Group, click **Add Members**. Use the checkboxes to select the desired Members to be added. The Search field can be used to quickly identify a Member to add. Click **Save** when all Members are selected.

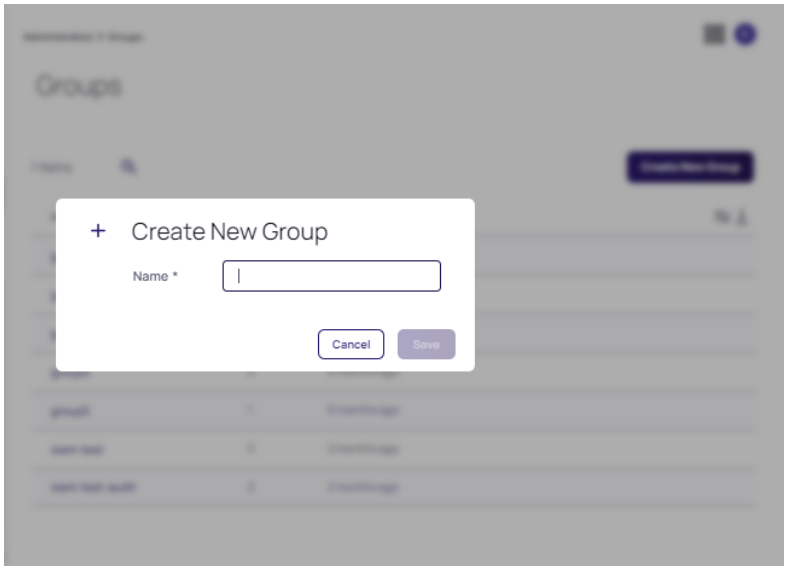


Creating Groups

Note: The Create permission is required for an account, in order to create a Group.

To create a Group:

1. At the Administration > Groups page, click **Create New Group**.
2. Enter a name for the Group and click **Create New Group**.



3. The Administration > Groups page is refreshed with the newly added Group.

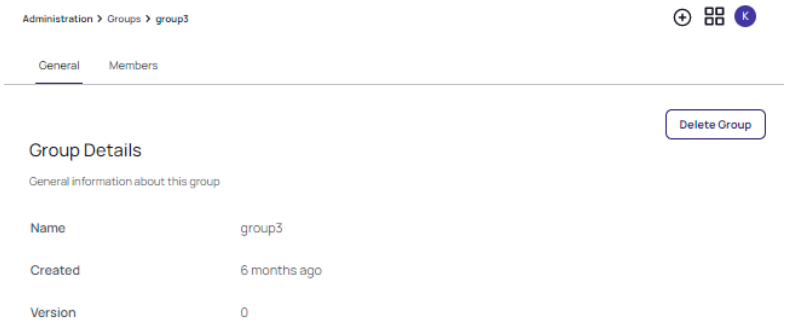
Deleting a Group

Note: The Delete permission is required for an account, in order to delete a Group.

To delete a Group:

1. At the Administration > Groups page, select the Group to be deleted.
2. At the **Group details** page, click **Delete Group**.

Usage



3. The Administration > Groups page is refreshed. The Group no longer appears in the Group list.

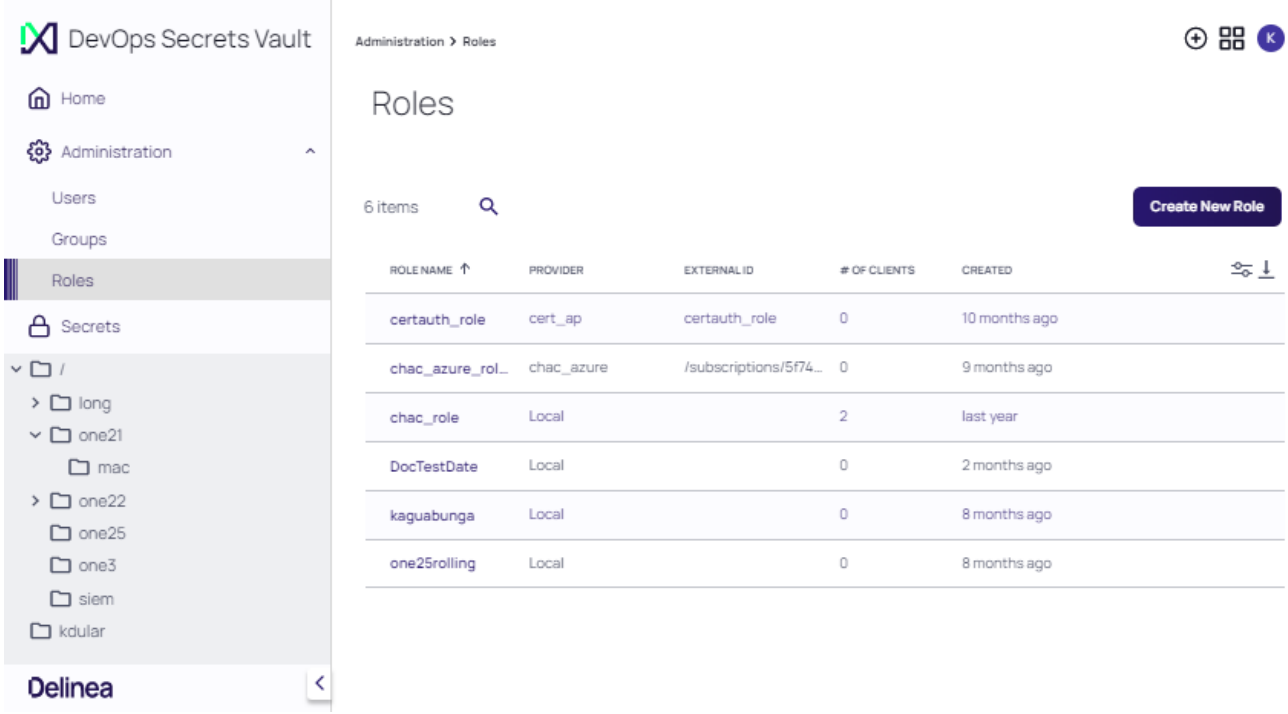
Roles

Sign into your DSV tenant. On the Home page, open the **Administration** drop-down to access **Roles**.

Viewing Roles

At the **Administration** drop-down, select **Roles**. A table of the currently defined Roles is displayed.

Use the filter icon at the far right of the table header to enable or disable columns included in the table. The search icon can be used to identify a specific Role for display.



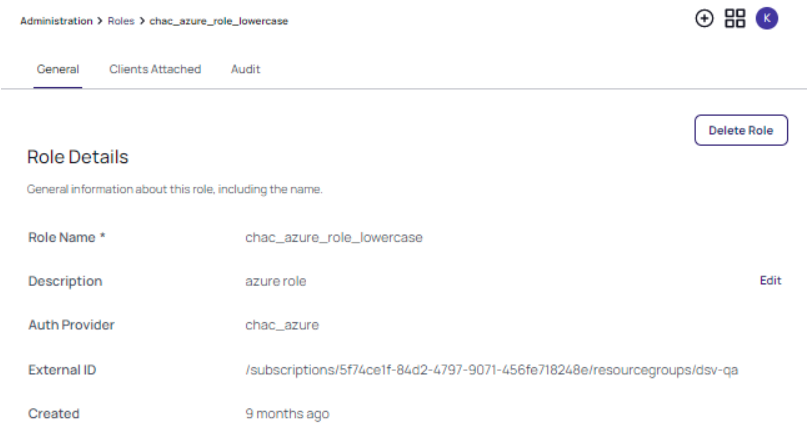
The following parameters are displayed for each Role.

Usage

Parameter	Value
ROLE NAME	Locally used display name for identifying the Role
PROVIDER	The mechanism used to authenticate the Role
EXTERNAL ID	The name that identifies the Role in the integration
# OF CLIENTS	The number of clients associated with the Role
CREATED	The time elapsed (in months) since the role was created. Hover over the value to display the date and time the Role was created.

Viewing Role Details

Click any Role in the table to access the Role Details page.

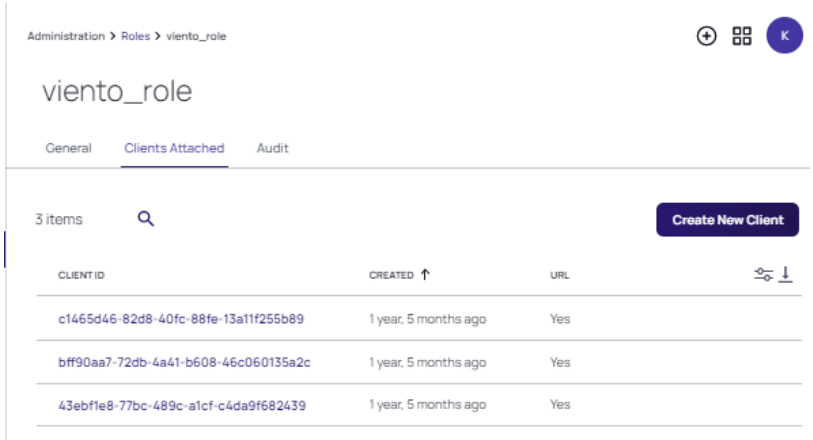


The tabs at the top of the Role Details page provides the following functionality.

Tab	Functionality
General	Displays role details that include the name, description, authentication provider, external ID, and date created for the Role
Clients Attached	Displays the clients attached to the Role and allows filtering and searching of clients. The date displayed indicates when the client was created in the application. If the **URL** column indicates **YES**, the client link represents and active URL for login.
Audit	Displays all events for the given Role

Attached Clients

With the **Clients Attached** tab selected, click any Client ID to view its Client Details page.



The following parameters are displayed for each client.

Parameter	Definition
Client ID	The unique identifier assigned to the client.
Description	A narrative summary that identifies the client.
Role	The Role associated with the client.
URL	When enabled, login via a URL is enabled. When disabled, user credentials are required for login.
Client TTL	This field appears when a URL is selected. The Time To Live (TTL) is the time the client will exist. When set to **0**, the client will exist indefinitely.
Created	A reference to when the client was created. Hover over the entry to view the creation date and time.

Click **Delete Client** to remove an attached client. The client is no longer displayed on the **Clients Attached** list.

Click **Create New Client** to attach a new client, and supply the requested parameters at the **Create New Client** dialog.

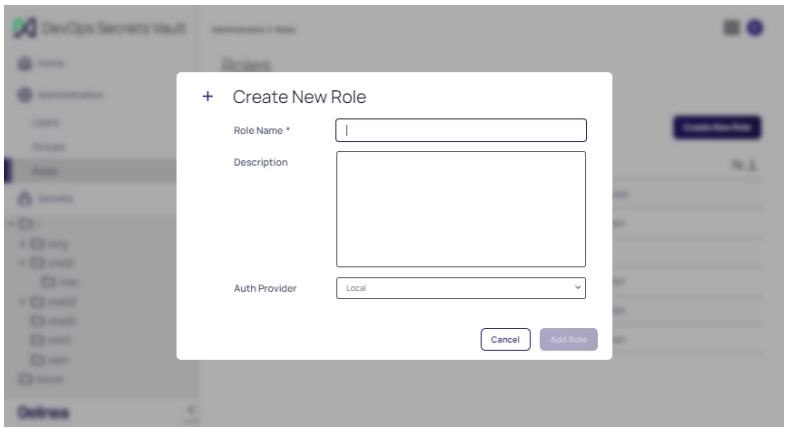
When creating a client, a **Description**, **Client TTL (sec)**, and **Client Uses** value is required. **Client Uses** is the number of times the client credential can be used. When set to **0**, the client can be used indefinitely. If login is enabled with a URL, enable the **URL** checkbox. Click **Save**.

Creating Roles

Note: The Create permission is required for an account, in order to create a Role.

To create a Role:

1. At the Administration | Roles page, click **Create Role**.
2. Enter a **Role Name** and **Description** for the Role. Select the **Auth Provider** and click **Add Role**.



3. The Roles page is refreshed with the newly added Role.

Deleting a Role

Note: The Delete permission is required for an account, in order to delete a Role.

To delete a Role:

1. At the Administration | Roles page, select the Role to be deleted.
2. At the Role details page, click **Delete Role**.
3. The Roles page is refreshed. The Role no longer appears in the Role list.

Policies

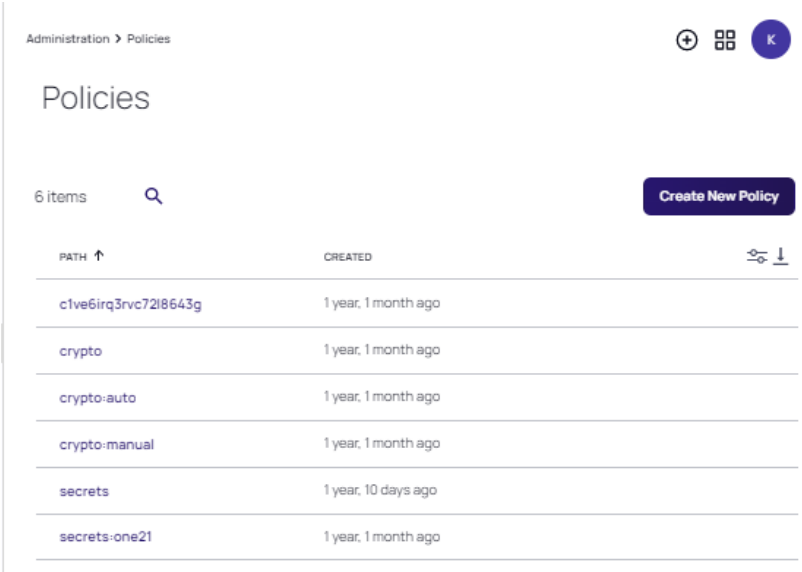
Policies determine access to resources and the authority to work on resources. Sign into your DSV tenant. On the Home page, open the **Administration** drop-down to access **Policies**.

Viewing Policies

At the **Administration** drop-down, select **Policies**. A table of the currently defined Policies is displayed.

Use the filter icon at the far right of the table header to enable or disable columns included in the table. The search icon can be used to identify a specific Policy for display.

Usage



The following parameters are displayed for each Policy.

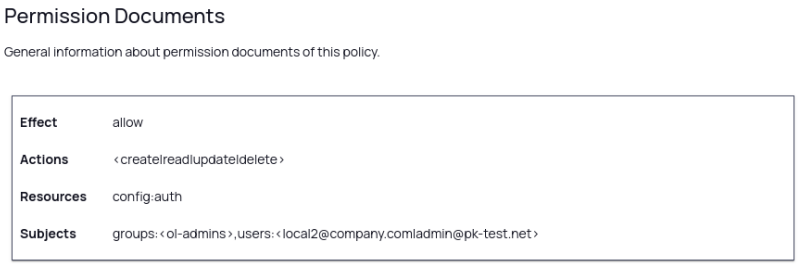
Parameter	Value
PATH	The path to the policy.
CREATED	The time elapsed (in year/months/days/hours; or "just now") since the Policy was created. Hover over the value to display the date and time the Policy was created.

Viewing Policy Details

Click any Policy in the table to access the Policy Details page.

Click **Delete Policy** to remove the currently displayed Policy.

In addition to the basic information regarding the policy identifiers and dates, the Policy includes information for Permission Documents.



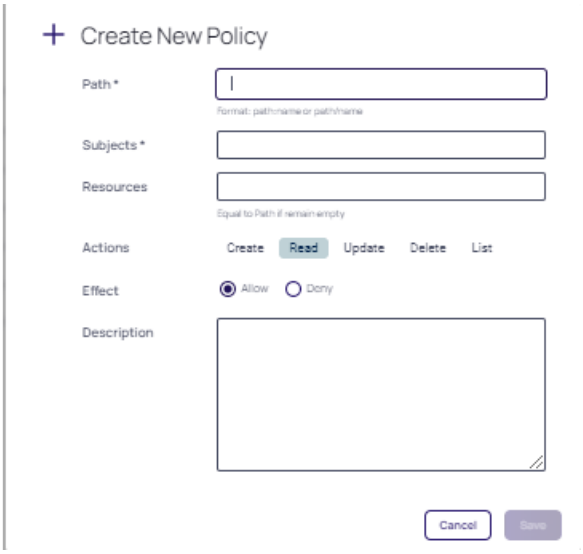
Parameter	Value
Effect	Determines whether the permissions for the Policy are allowed (**allow**) or denied **deny**).

Parameter	Value
Actions	The actions (**create**, **read**, **update**, **delete**, **list**, **assign**) that are supported by the Policy. **<.*>** indicates all actions. **Note**: Basic policy functionality is currently supported, however, future enhancements to the UI for Policies will provide full functionality. Refer to the CLI Reference for robust Policy customization.
Resources	The resources affected by the Policy. Typically, this is the same value as the **PATH**, unless a more specific instance of resource applies.

Creating a New Policy

1. From the Policies page, click **Create New Policy**.

   Note: The Create permission is required for an account, in order to create a Policy.



2. At the **Create New Policy** dialog, provide the following parameters:

Parameter	Value
Path	The path to the Policy.
Subjects	The users (groups or roles) that the Policy affects.
Resources	The Resources that the Policy affects. Typically, this is the same value as the **PATH**, unless a more specific instance of resource applies.

Parameter	Value
Actions	The actions (**create**, **read**, **update**, **delete**, **list**, **assign**) that are supported by the Policy. **<.*>** indicates all actions.
Effect	Controls whether the permissions for the Policy are allowed (**allow**) or denied (**deny**).
Description	A short narrative that identifies the Policy.

3. Click **Save**.

Engines and Pools

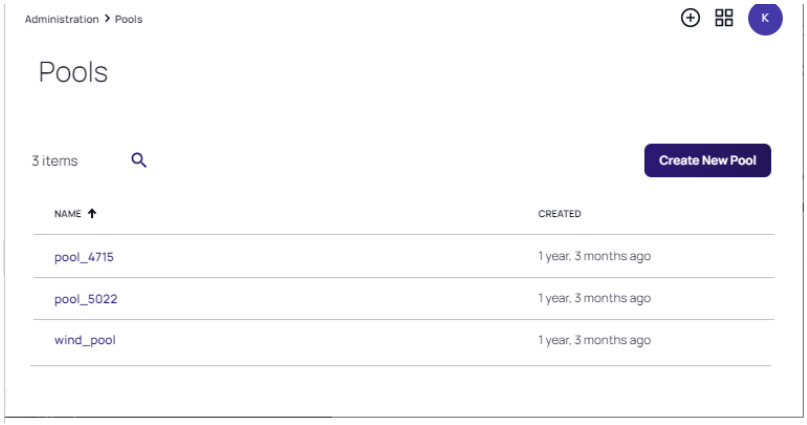
Log onto your DSV tenant. On the Home page, open the **Administration** drop-down.

Viewing and Pools

At the **Administration** drop-down, select **Pools**. A table of the currently defined Pools is displayed, along with the name of the pool and how long ago it was created.

Hover over the **CREATED** entry to view the date and time the pool was created.

Use the filter icon at the far right of the table header to enable or disable columns included in the table.



Creating a Pool

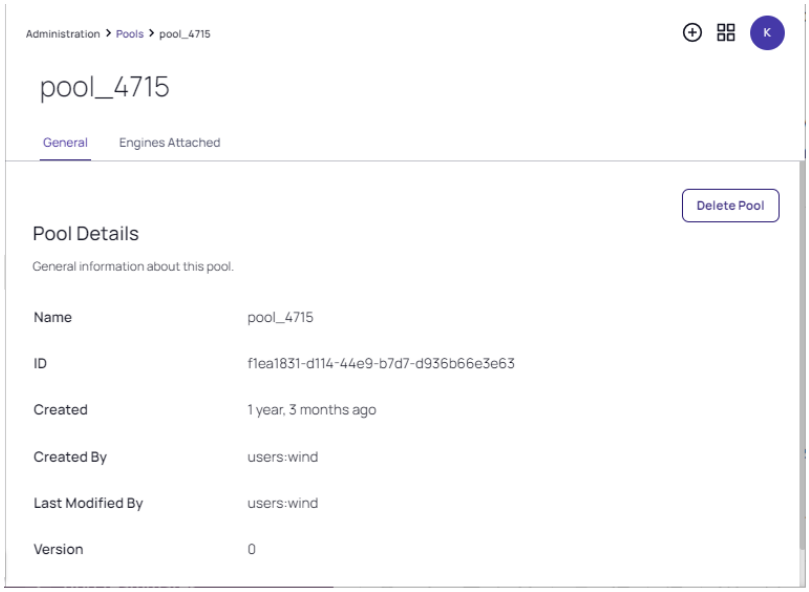
On the Pools page, click **Create New Pool**. Supply a **Name** for the Pool and click **Save**. The Pool Details page is displayed for the newly created Pool.

Viewing Pool Details

Click any Pool in the list of Pools. The following parameters are displayed on the **General** tab on the Pool Details page.

Usage

Parameter	Value
Name	Locally used display name for identifying the Pool in DSV
ID	Unique identifier used for this Pool
Created, **Created By**	When the Pool was created and by what user
Last Modified By	The user that made the last update to the pool
Version	The current version of the pool.



Use **Delete Pool** to remove the pool from the application. Click **Delete** again at the confirmation prompt.

Viewing Attached Engines

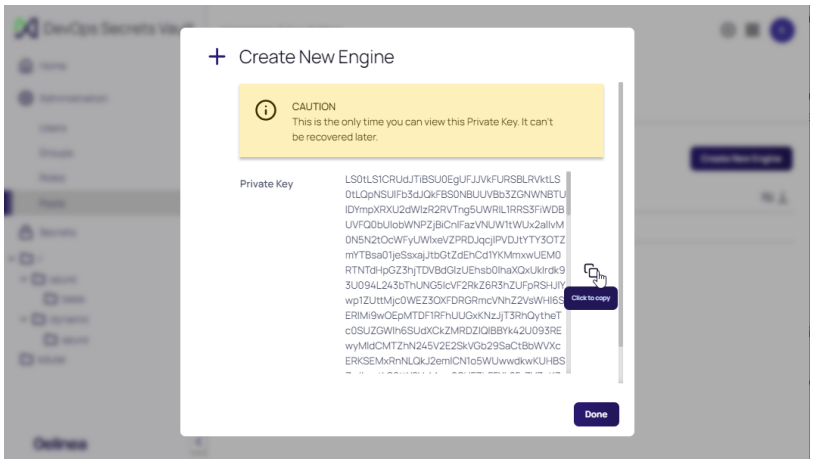
Click the **Engines Attached** tab on the Pool Details page. The name of the engine, how long ago it was created, and its last known heartbeat are displayed.

Hover over the **CREATED** and **LAST HEARTBEAT** values to display the associated date and time.

Creating a New Engine

To create a new engine, click **Create New Engine**. Supply a **Name** for the Engine, or select an existing engine to attach. Click **Save**.

The application displays the Private Key for the engine. Use this prompt to copy the key, as it cannot be recovered again.



Click **Done**. The Engine Details page is displayed for the newly created Engine.

Viewing Engine Details

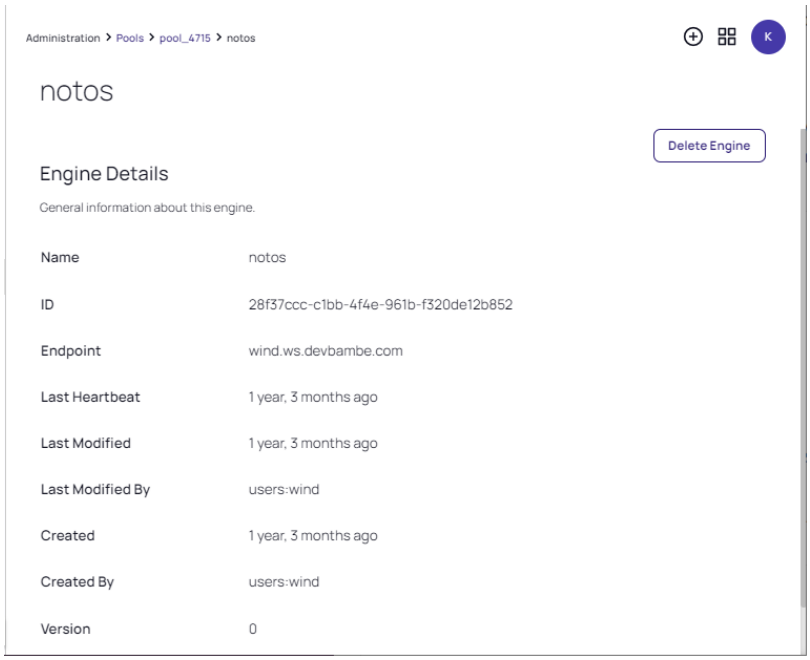
On the Pool Details page, click **Engines Attached**. Select any engine to view its details.

The following parameters are displayed for the associated Engine.

Parameter	Value
Name	Locally used display name for identifying the Engine in DSV
ID	Unique identifier used for this Engine
Endpoint	
Last Heartbeat	The last known signal generated by the engine
Last Modified, **Last Modified By**	The user that made the last update to the pool and when it was made
Created, **Created By**	When the Engine was created and by what user
Version	The current version of the engine.

Use **Delete Engine** to remove the engine from the application. Click **Delete** again at the confirmation prompt.

Usage



SIEM

SIEM integrations are viewable and actionable from the Administration menu on the Home page. SEIM integrations produce the audit logs of captured actions that are sent to registered Security Information and Event Management (SIEM) endpoints in near real time.

Note: SIEM actions are also supported in the CLI. Refer to SIEM Integrations.

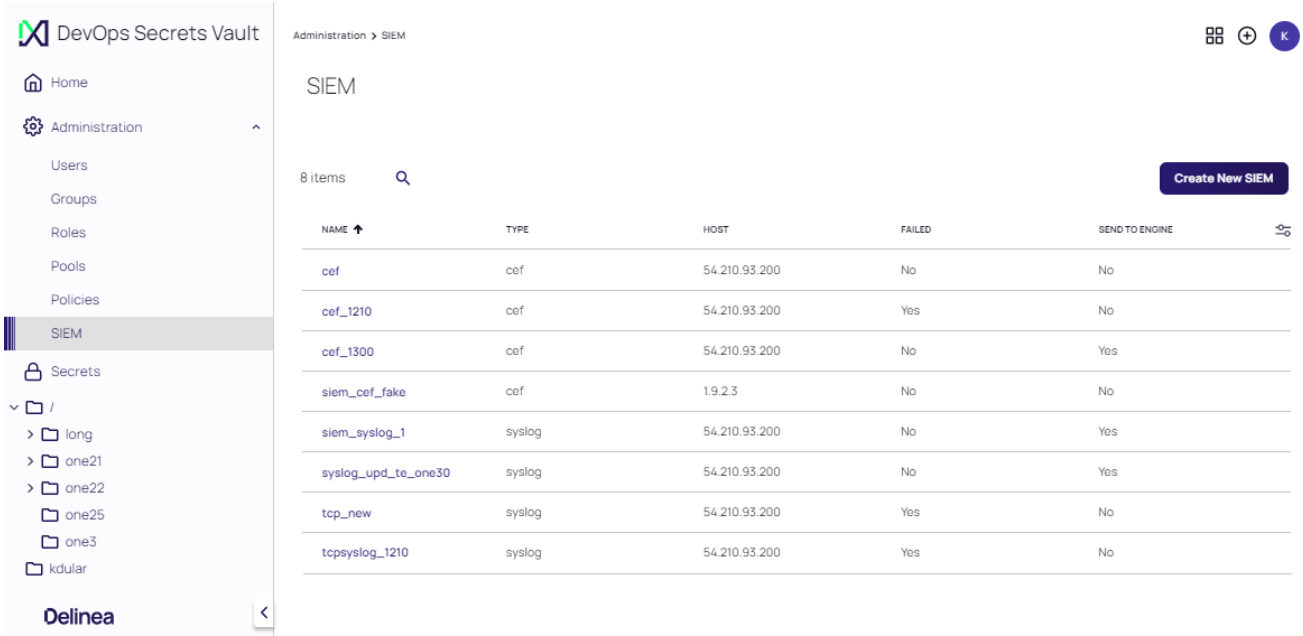
Viewing SIEM Integrations

To access a list of the currently defined SIEM integrations, select **SIEM** from the Administration drop-down.

Note: For every audit action, DSV will try twice to reach the endpoint. If the endpoint is unresponsive after ten actions and retries, DSV will deregister the endpoint and mark it as failed (**FAILED** yes). The endpoint must be recreated or updated to be used again.

At the SIEM page, click any SIEM to view its details. In addition to the parameters defined when the SIEM integration was created (refer to Creating a SIEM Integration for Auditing), the following information is provided:

- ID: The internal audit ID associated with the protocol.
- Failed Events: The number of times a send to the endpoint failed.



Creating a SIEM Integration for Auditing

1. From the Home page, select the SIEM folder, then click **Create New SIEM**. Supply the following information at the Create New SIEM dialog box.

Field	Description
Name (required)	The label in the UI used to identify the SIEM configuration.
SIEM Type (required)	The logging output format used to register an endpoint.
Protocol (required)	transport protocol expected by endpoint.
Logging Format (required)	The format for Syslog messages. Currently, messages must be in RFC 5424-compliant format.
Host (required)	The URL of the server that hosts the configuration.
Port (required)	The port number used in the protocol.

Field	Description
Auth (required)	The authentication method used in the protocol.
Endpoint	The endpoint on the network that SIEM logs are generated for.
Send to Engine	Enabling this control allows audit logs to be sent through a DSV engine to a server that isn't accessible to the outside internet. An engine and pool must be already configured.
Pool (required)	If **Send to Engine** is enabled, this field allows selection of an engine pool. A message will appear if a pool does not exist for selection or a network delay occurs.

2. Click **Save**.

Deleting a SIEM Integraion

To delete a SIEM integration, select the SIEM integration in the list on the SIEM page to access its details.

Click **Delete SIEM**.

The integration is removed from the SIEM list.

Authentication

DSV supports several authentication methods.

Password

Password authentication relies directly on individual User accounts. It requires an initial administrator account with username and password authentication.

DSV encrypts the password in the config on successful authentication. This prevents users from accidentally disclosing the password by sending the config to someone or by giving access to the computer to another person.

Routine activities associated with this authentication method include:

- creating a new user
- entering the username and password of the new user
- adding the new user to the DSV config

See the users portion of the CLI Reference for details.

Client Credentials

In this method, you authenticate via a client id and a secret generated by the vault. This suits situations requiring application or server access when no third party trust is feasible.

Client credentials tie to roles, not user accounts, the significance being that roles have a one-to-many relationship with user accounts. Using roles-based authentication allows you to efficiently apply uniform authentication requirements to collections of users.

Usage

Routine activities associated with the client credentials authentication method include:

- creating a new role
- adding the new role to the DSV config
- creating new client credentials using the new role
- invoking the **init** command and supplying those client credentials

See the Roles portion of the CLI Reference for more information.

Thycotic One Authentication

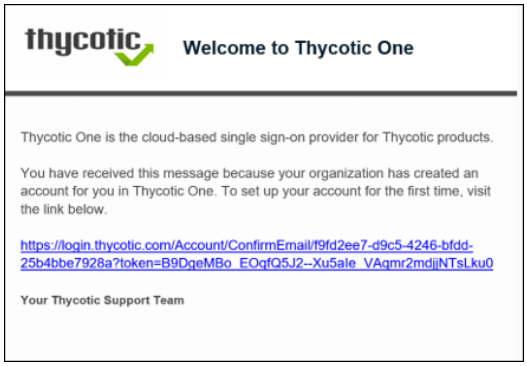
Users can authenticate into DSV using a Thycotic One account. To add a User with Thycotic One authentication:

> **Note:** Thycotic One authentication provides the option of sending a welcome email directly to new users with a link to create their login. To enable welcome emails, use the `dsv config auth-provider update` command and set the `sendWelcomeEmail` value to `true`.

1. Create a user and assign credentials using the following format:

   ```
   dsv user create --username thyoneuser@yourorganization.com --provider thy-one
   ```

2. If you have set `sendWelcomeEmail` to `true` in your auth-provider configuration, the user will receive an email with a link to both confirm their email address and setup a password.



3. Once the Thycotic One user follows the link and sets a password, they will be ready to authenticate to DSV.

Third Party Authentication

Besides Thycotic One, DevOps Secrets Vault works with third party authentication providers, including:

AWS IAM: DSV uses the current AWS profile to generate a signed request which the vault validates against AWS. You can use this with EC2 instances and with a Lambda that is assigned an IAM Role or an IAM User account. See Authentication: AWS

Azure MSI: DSV uses the assigned Azure Managed Service Identity (MSI). See Authentication: Azure

GCP Service Accounts: DSV uses GCP's service accounts to enable secrets access to just about anything that can be assigned a service account. Google Compute Engines (GCE) may also be assigned service accounts and authenticated through GCE metadata. See Authentication: GCP.

OIDC Provider DSV connects to Thycotic One, which in-turn may connect to any OIDC provider. See Authentication: OIDC.

Profiles

On initial configuration, your DevOps Secrets Vault config will have just one profile with the choices you specified for credentials storage, authentication type, and cache strategy for secrets.

However, DSV supports creating other profiles, potentially with different credentials, and adding them to the config. Once the config has more than one profile, you can set which one DSV will use by default.

Add a Profile to a Config

DSV syntax gives you two ways to add a profile to the config.

- Run `dsv init` and type `add` or `a` at the prompt. Then enter the name of a new profile.
- To do it with one command, run `dsv init --profile [name]`.

See the Config Contents

If you want to verify the profile has been added, output the updated config contents.

```
dsv cli-config read
```

Using an Alternate Profile for a Specific CLI Action

For a config with more than one profile, the profile used by default for any command will be the first profile created. However, you can override the default by specifying the profile to be used for a command as a parameter.

```
dsv secret read --path mySecret --profile developer
```

So commanded, the CLI will try to auth as the User specified in the *developer* profile and attempt to read the secret as that user.

The CLI does not have a command to set the default for all commands moving forward. For that, you should edit the *.thy.yml* file in the home directory to change the profile set as the default.

Authentication: AWS

Use `dsv config auth-provider search -e yaml` to see all of your current authentication providers.

Initially, the only authentication provider is Thycotic One, similar to this:

```
created: "2019-11-11T20:29:20Z"
createdBy: users:thy-one:admin@company.com
id: xxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z"
```

```
lastModifiedBy: users:thy-one:admin@company.com
name: thy-one
properties:
 baseUri: https://login.thycotic.com/
 clientId: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 clientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
type: thycoticone
version: "0"
```

AWS Authentication Provider

To add an AWS account to act as an authentication provider:

- `dsv config auth-provider create --name <name> --type aws --aws-account-id <AWS account ID>`

in which:

- name is the friendly name used in DSV to reference this provider.
- type is the authentication provider type; in this case, aws.
- the property flag for AWS is `--aws-account-id` then include the account ID

To view the resulting addition to the config file, you would use:

`dsv config auth-provider <name> read -e yaml` where the example name we will use here is aws-dev

The readout would look similar to this:

```
created: "2019-11-12T18:34:49Z"
createdBy: users:thy-one:admin@company.com
id: xxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z"
lastModifiedBy: users:thy-one:admin@company.com
name: aws-dev
properties:
  accountId: "xxxxxxxxxxxx"
type: aws
version: "0"
```

AWS User Example

When you create a User in AWS, remember that the username serves as a friendly name within DSV. It does not have to match the Identity Access Management (IAM) username, but the provider must match the provider name previously configured.

```
dsv user create --username test-admin --external-id arn:aws:iam::xxxxxxxxxxx:user/test-
admin --provider aws-dev
```

After creating the User, modify the config to give that User access to the default administrator permission policy.

Usage

> **Note:** Adding a user to the admin policy is not security best practices. This is for example purposes only. Ideally, you would create a separate policy for this AWS user with restricted access. For details on limiting access through policies, see the Policy section.

```
dsv config edit -e yaml
```

Add *test-admin* as a User subject to the **Default Admin Policy**. Third party accounts must be prefixed with the provider name; in this case, the fully qualified username would be *aws-dev:test-admin*.

```
<snip>
- actions:
 - <.*>
 conditions: {}
 description: Default Admin Policy
 effect: allow
 id: xxxxxxxxxxxxxxxxxxxx
 meta: null
 resources:
 - <.*>
 subjects:
 - users:<aws-dev:test-admin|admin@company.com>
<snip>
```

Next, on a machine with the AWS CLI installed and configured with an AWS IAM user, download the DVS CLI executable appropriate to the OS of the machine, and initialize the CLI:

```
dsv init
```

When prompted for the authorization type, choose *AWS IAM (federated)*.

```
Please enter auth type:
       (1) Password (local user)(default)
       (2) Client Credential
       (3) #{ThycoticOne}# (federated)
       (4) AWS IAM (federated)
       (5) Azure (federated)
       (6) GCP (federated)
       (7) OIDC (federated)
```

DSV will prompt for the specific AWS profile to use if you are authenticating using a non-default AWS profile.

Please enter aws profile for federated aws auth (optional, default:default)

Read an existing Secret to verify you can authenticate to DSV and access data.

Usage

```
dsv secret read --path <path to secret>
```

AWS Role Example

This example assumes that you:

- have your own CLI configured locally with an admin account
- created an IAM Role in the AWS Console
- launched an EC2 instance using the IAM Role
- downloaded the CLI onto the EC2 instance

Create a corresponding Role in DSV with the external-id of the IAM Role's ARN.

```
dsv role create --name test-role --external-id arn:aws:iam::xxxxxxxxxxx:role/testlogin --
provider aws-dev
```

You should see a result similar to this:

```
{
"description": "",
"externalId": "arn:aws:iam::xxxxxxxxxxx:role/testlogin",
"name": "test-role",
"provider": "aws-dev"
}
```

Add the Role *aws-dev:test-role* to the **Default Admin Policy** in your vault config to grant the new Role admin access.

> **Note:** Adding a role to the admin policy is not security best practices. This is for example purposes only. Ideally, you would create a separate policy for this AWS role with restricted access. For details on limiting access through policies, see the Policy section.

Use the command `dsv config edit -e yaml`

```
<snip>
- actions:
 - <.*>
 conditions: {}
 description: Default Admin Policy
 effect: allow
 id: bgn8gjei66jc7148d9i0
 meta: null
 resources:
 - <.*>
```

```
  subjects:
  - users:<aws-dev:test-admin|admin@company.com>
  - roles:<aws-dev:test-role>
 <snip>
```

On the EC2 instance, configure the CLI by running `dsv init` and choosing AWS IAM as the authentication type.

Once configured, ensure you can read an existing Secret to verify the EC2 instance is able to authenticate and access data.

```
dsv secret read --path <path to secret>
```

Authentication: Azure

Use `dsv config auth-provider search -e yaml` to see all of your current authentication providers.

Initially, the only authentication provider is Thycotic One, similar to this:

```
created: "2019-11-11T20:29:20Z"
createdBy: users:thy-one:admin@company.com
id: xxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z"
lastModifiedBy: users:thy-one:admin@company.com
name: thy-one
properties:
 baseUri: https://login.thycotic.com/
 clientId: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 clientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
type: thycoticone
version: "0"
```

Azure Authentication Provider

To add an Azure account to act as an authentication provider:

- `dsv config auth-provider create --name <name> --type azure --azure-tenant-id <Azure tenant ID>`

where:

- name is the friendly name used in DSV to reference this provider
- type is the authentication provider type; in this case, azure
- the property flag for Azure is `--azure-tenant-id`

To view the resulting addition to the config file, you would use:

`dsv config auth-provider <name> read -e yaml` where the example name we will use here is azure-prod

The readout would look similar to this:

```
created: "2019-11-12T18:34:49Z"
createdBy: users:thy-one:admin@company.com
-id: xxxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z"
lastModifiedBy: users:thy-one:admin@company.com
name: azure-prod
properties:
 tenantId: xxxxxxxxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
type: azure
version: "0"
```

Azure User Assigned MSI Example

First you will need to configure the User that corresponds to an Azure User Assigned MSI.

The username is a friendly name within DSV. It does not have to match the MSI username, but the provider must match the resource id of the MSI in Azure.

```
dsv user create --username test-api --provider azure-prod --external-id
/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/build/providers/Microsoft.ManagedIdentity/userAssignedIdentiti
es/test-api
```

Modify the config to give that User access to the default administrator permission policy.

> **Note:** Adding a user to the admin policy is not security best practices. This is for example purposes only. Ideally, you would create a separate policy for this Azure user with restricted access. For details on limiting access through policies, see the Policy section.

```
dsv config edit --encoding yaml
```

Add the User as a subject to the **Default Admin Policy**. Third party accounts must be prefixed with the provider name; in this case the fully qualified username will be *azure-prod:test-api*.

```
<snip>
- actions:
 - <.*>
 conditions: {}
 description: Default Admin Policy
 effect: allow
 id: xxxxxxxxxxxxxxxxxxxx
 meta: null
 resources:
 - <.*>
 subjects:
 - users:<azure-prod:test-api|admin@company.com>
<snip>
```

Usage

On a VM in Azure that has the User MSI assigned as the identity, download the DVS CLI executable appropriate to the OS of the VM and initialize the CLI.

```
dsv init
```

When prompted for the authorization type, choose the `Azure (federated)` authentication option.

```
Please enter auth type:
       (1) Password (local user)(default)
       (2) Client Credential
       (3) #{ThycoticOne}# (federated)
       (4) AWS IAM (federated)
       (5) Azure (federated)
       (6) GCP (federated)
       (7) OIDC (federated)
```

Read an existing secret to verify you can authenticate and access data.

```
dsv secret read --path <path to a secret>
```

Azure Resource Group

If you want to grant access to a set of VMs in a resource group that use a System assigned MSI rather than a User assigned MSI, you can create a role that corresponds to the resource group's resource ID.

```
dsv role create --name identity-rg  --external-id /subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/build --provider azure-prod
```

Modify the config to give that role access to the default administrator permission policy.

> **Note:** Adding a role to the admin policy is not security best practices. This is for example purposes only. Ideally, you would create a separate policy for this Azure role with restricted access. For details on limiting access through policies, see the Policy section.

```
dsv config edit --encoding yaml
```

Add the User as a subject to the **Default Admin Policy**. Third party accounts must be prefixed with the provider name; in this case the fully qualified role name will be *azure-prod:identity-rg*.

```
<snip>
- actions:
```

```
  - <.*>
  conditions: {}
  description: Default Admin Policy
  effect: allow
  id: bgn8gjei66jc7148d9i0
  meta: null
  resources:
  - <.*>
  subjects:
  - users:<azure-prod:test-api|admin@company.com>
  - roles:<azure-prod:identity-rg>
<snip>
```

On a VM in Azure that is part of the resource group and has a system-assigned MSI, download the DVS CLI executable appropriate to the OS of the VM and initialize the CLI.

```
dsv init
```

When prompted for the authorization type, choose the `Azure (federated)` option.

```
Please enter auth type:
       (1) Password (local user)(default)
       (2) Client Credential
       (3) #{ThycoticOne}# (federated)
       (4) AWS IAM (federated)
       (5) Azure (federated)
       (6) GCP (federated)
```

Read an existing secret to verify you are able to authenticate and access data.

```
dsv secret read --path <path to a secret>
```

Authentication Google Cloud Platform (GCP)

DevOps Secrets Vault provides two ways to authenticate using GCP. One is through a Google service account and the other is through Google Compute Engine (GCE) metadata.

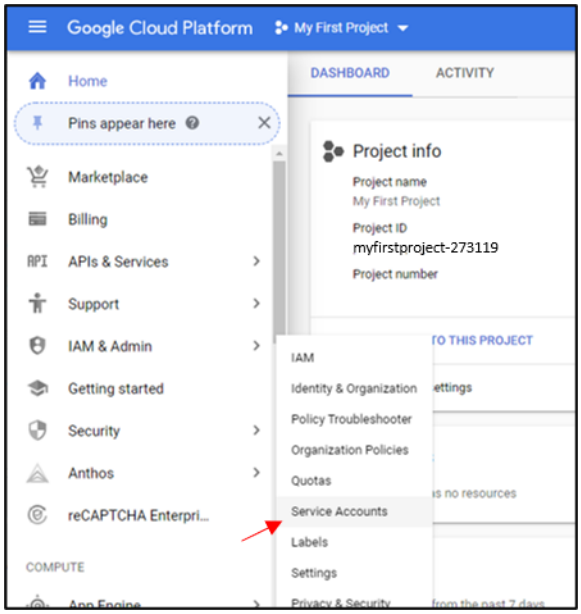
Google Service Account Authentication

To setup GCP authentication using service accounts in DSV, a GCP service account must be provided that DSV can use as the authentication provider. This service account must be assigned to the project you are working in, have the role **Service Account Key Admin** so that it can issue and manage service account tokens, and a key must be generated.

These steps can be done programmatically, but we will use the GCP Console.

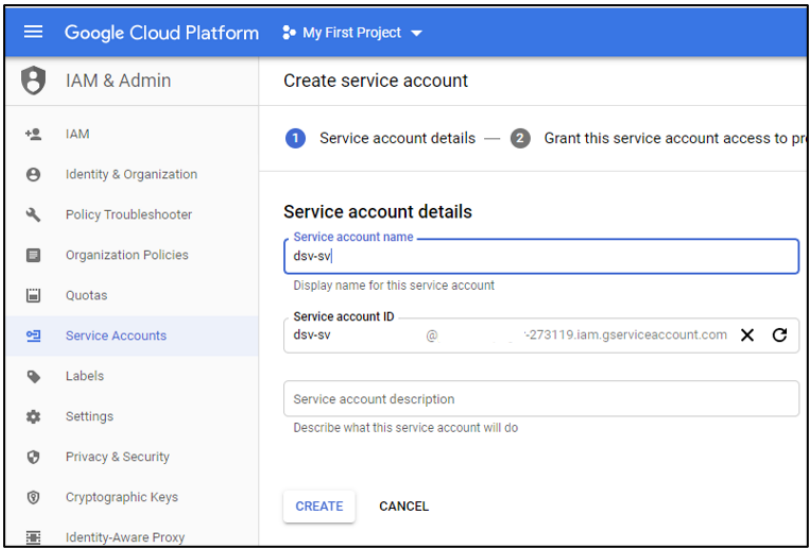
GCP Service Account Setup

In the GCP Console Home page, go to your project, hover **IAM & Admin**, and then click **Service Accounts**.



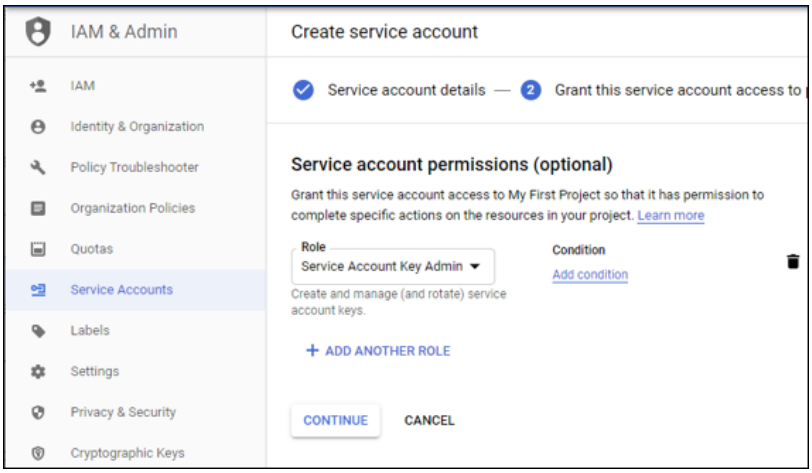
At the top, click **CREATE SERVICE ACCOUNT**.

For the first step, enter an account name. We will use `dsv-svc` in this example. Click **CREATE**.

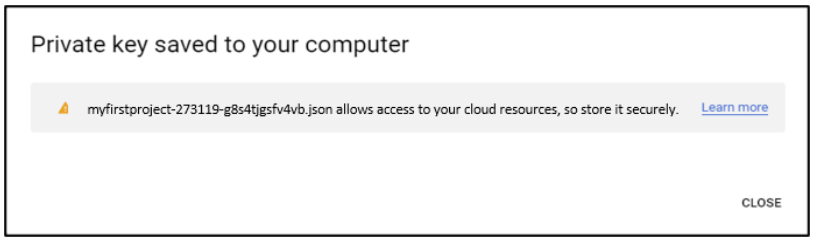


In the second step, click the drop-down arrow in the **Select a role** box, type `service account key admin` in the filter and select **Service Account Key Admin**. Then click **Continue**.

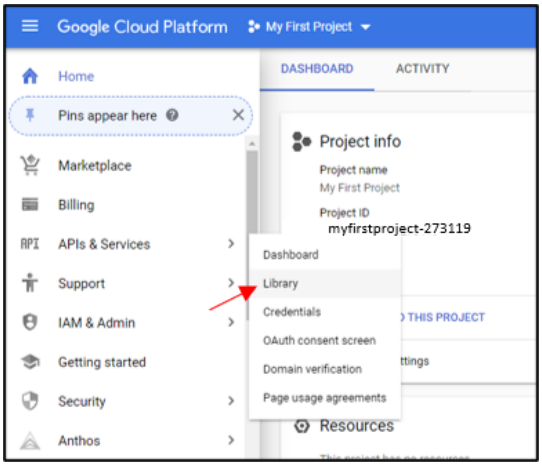
Usage



In the third step, click **CREATE KEY** and when the option to generate a file slides in from the right, select **json** and click **CREATE**. A file will be downloaded that will have all the information needed to setup the DSV authentication provider.

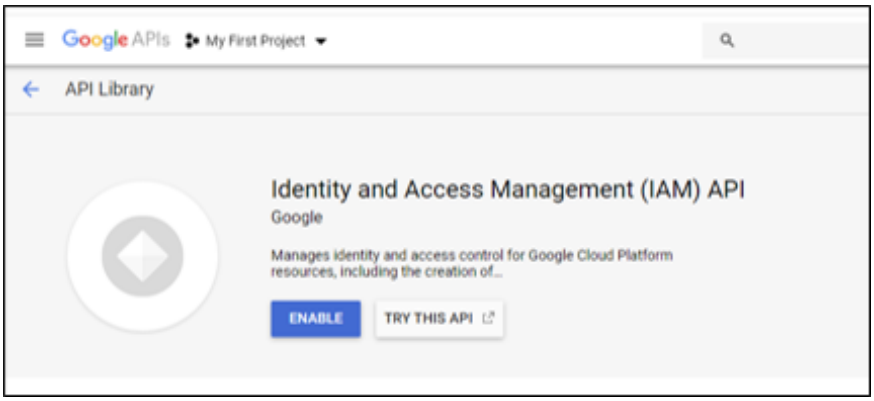


The Goolge API for IAM must be enabled. To do this in the Google Console, go to the relevant project and on the left navigation, hover **APIs & Services** then select **Library**.



In the search, type `Identity and Access` and in the results, select the **Identity and Access Management (IAM) API**. Click **Enable**.

Usage



DSV Authentication Provider Setup

Go back to the terminal (DevOps Secrets Vault CLI).

Use `dsv config auth-provider search -e yaml` to see all of your current authentication providers.

Initially, the only authentication provider is Thycotic One, similar to this:

```
created: "2019-11-11T20:29:20Z"
createdBy: users:thy-one:admin@company.com
id: xxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z"
lastModifiedBy: users:thy-one:admin@company.com
name: thy-one
properties:
 baseUri: https://login.thycotic.com/
 clientId: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 clientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
type: thycoticone
version: "0"
```

Setup the DSV authentication provider. Create a json file named `auth-gcp.txt` with the following format, substituting the `dsv-svc` service account values in the key file you downloaded from the GCP console.

```
{
  "name": "gcloud",
  "type": "gcp",
  "properties": {
    "ProjectId": "{project-id}",
    "type": "service_account",
    "PrivateKeyId": "{private-key-id}",
    "PrivateKey": "-----BEGIN PRIVATE KEY-----{private-key}-----END PRIVATE KEY-----\n",
    "ClientEmail": "{clientemail}",
    "TokenURI": "https://oauth2.googleapis.com/token"
  }
}
```

In the DSV CLI, run `dsv config auth-provider create --data @auth-gcp.txt` to create the GCP authentication provider.

`dsv config auth-provider <name> read -e yaml` where the example name we will use here is gcloud

```
created: "2019-11-12T18:34:49Z"
createdBy: users:thy-one:admin@company.com
id: bq4ce17cj2bc72qun8vg
lastModified: "2020-05-18T03:58:15Z"
lastModifiedBy: users:thy-one:admin@company.com
name: gcloud
properties:
clientEmail: dsv-svc@myfirstproject-xxxxxx.iam.gserviceaccount.com
privateKey: |
-----BEGIN PRIVATE KEY-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX=
-----END PRIVATE KEY-----
privateKeyId: 9xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx3
projectId: myfirstproject-xxxxxx
tokenUri: https://oauth2.googleapis.com/token
type: service_account
type: gcp
```

DSV Service Account/User Mapping

Now the service account that is going to access DSV is required. For this example, we will name this account `client-svc` The setup in GCP is the same as above for the `dsv-svc` account except that when the role is assigned,

it must be **Service Account Token Creator** so that this account can request tokens. Also, after generating the key, make sure to save the file to the local machine that will access DSV and note the location.

In the DSV CLI, create a User called `gcp-test` referring to the `client-svc` service account with `gcloud` as the authentication provider using `dsv user create --username gcp-test --provider gcloud --external-id client-svc@myfirstproject-273119.iam.gserviceaccount.com`.

```
{
  "cursor": "",
  "data": [
    {
      "created": "2020-04-04T17:56:33Z",
      "createdBy": "users:thy-one:admin@company.com",
      "externalId": "client-svc@myfirstproject-xxxxxx.iam.gserviceaccount.com",
      "id": "d6a8e1e5-5554-4fc8-a4ca-1c1a653f9095",
      "lastModified": "2020-04-04T17:56:33Z",
      "lastModifiedBy": "users:thy-one:admin@company.com",
      "provider": "gcloud",
      "userName": "gcp-test",
      "version": "0"
    }
  ],
  "length": 1,
  "limit": 25
}
```

Set an environmental variable named GOOGLE_APPLICATION_CREDENTIALS to the path of the key file for `client-svc` that was just downloaded.

In Linux or Mac, this might look like:

```
export GOOGLE_APPLICATION_CREDENTIALS="/home/user/Downloads/[FILE_NAME].json"
```

Windows Powershell

```
$env:GOOGLE_APPLICATION_CREDENTIALS="C:\Users\username\Downloads\[FILE_NAME].json"
```

Windows Command Line

```
set GOOGLE_APPLICATION_CREDENTIALS=C:\Users\username\Downloads\[FILE_NAME].json
```

After creating the User, modify the `config` to give that User access to the default administrator permission policy.

> **Note:** Adding a User to the admin policy is not security best practices. This is for example purposes only. Ideally, you would create a separate policy for this GCP service account with restricted access. For details on limiting access through policies, see the Policy section.

```
dsv config edit
```

Add *gcloud:gcp-test* as a User to the **Default Admin Policy**. Third party accounts must be prefixed with the provider name; in this case, the fully qualified username would be *glcoud:gcp-test*.

```
<snip>
- actions:
 - <.*>
 conditions: {}
 description: Default Admin Policy
 effect: allow
 id: xxxxxxxxxxxxxxxxxxxx
 meta: null
 resources:
 - <.*>
 subjects:
 - users:<gcloud:gcp-test|admin@company.com>
<snip>
```

Run `dsv init` filling out the desired values and selecting **6** GCP (federated) when prompted for the auth type.

```
Please enter auth type:
       (1) Password (local user)(default)
       (2) Client Credential
       (3) #{ThycoticOne}# (federated)
       (4) AWS IAM (federated)
       (5) Azure (federated)
       (6) GCP (federated)
       (7) OIDC (federated)
```

Run `dsv auth` to verify authentication. A token will be displayed.

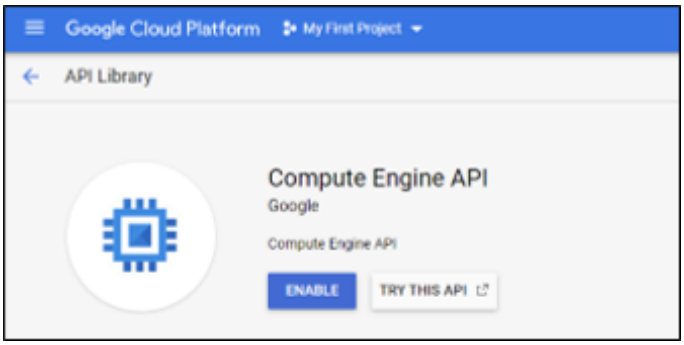
Run `dsv secret read <path to any secret>` to verify secret access.

Google Compute Engine (GCE) Metadata Authentication

The idea behind GCE Metadata authentication is to enable a GCE instance to gain access to DevOps Secrets Vault.

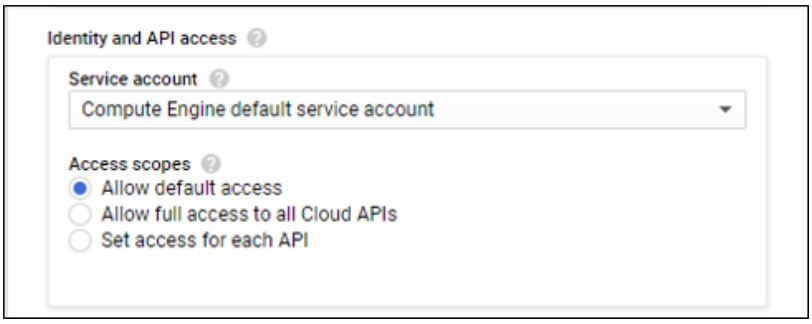
In this example we assume you have created a Linux Google Compute Instance and have the Google Compute Engine API enabled.

Usage



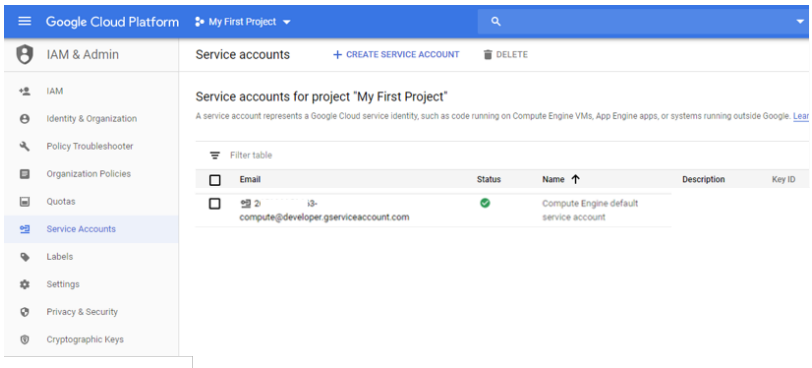
It is further assumed that the **Compute Engine default service account** is used. However, you can assign a different service account to the Compute instance if desired.

> **Note:** Using the GCE default service account is generally not best practices because it is defaulted to every GCE that is created, violating the idea of least privileges. This is for illustration purposes.



To find the **Compute Engine default service account** email, from the GCP Console Home, hover **IAM** and then click **Service Accounts**.

The name will say "Compute Engine default service account". Copy and store the email for later.



DSV GCE Authentication Provider setup

Using any computer with Admin DSV access, we now want to setup the DSV Authentication Provider.

Create a file named 'auth-gcp.txt' in the following format and substituting your ProjectID.

Usage

```
{
  "name": "gcloud-gce",
  "type": "gcp",
  "properties": {
    "ProjectId": "myfirstproject-273119"
  }
}
```

Run `dsv config auth-provider create --data @auth-gcp.txt` to implement the Authentication Provider.

To view the resulting addition to the config file, you would use:

`dsv config auth-provider <name> read -e yaml` where the example name we will use here is *gcloud-gce*.

```
- ID: bq71e5co19js72ppv140
name: gcloud-gce
properties:
projectId: myfirstproject-273119
type: gcp
tenantName: company
```

```
created: "2019-11-12T18:34:49Z"
createdBy: users:thy-one:admin@company.com
id: xxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z"
lastModifiedBy: users:thy-one:admin@company.com
name: gcloud-gce
properties:
projectId: myfirstproject-xxxxxx
type: gcp
version: "0"
```

DSV GCE Metadata Service Account/DSV User Mapping

Run `dsv user create --username gce-test --provider gcloud-gce --external-id {default compute service account email}` using the default service account email we saved earlier.

```
{
  "created": "2020-04-09T12:59:44Z",
  "createdBy": "users:thy-one:admin@company.com",
  "externalId": "2XXXXXXXXXX3-compute@developer.gserviceaccount.com",
  "id": "19709b4e-2a13-4164-a930-81997b568036",
  "lastModified": "2020-04-09T12:59:44Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "provider": "gcloud-gce",
  "userName": "gce-test",
```

```
  "version": "0"
}
```

After creating the User, modify the config to give that User access to the default administrator permission policy.

> **Note:** Adding a user to the admin policy is not security best practices. This is for example purposes only. Ideally, you would create a separate policy for this GCP service account with restricted access. For details on limiting access through policies, see the Policy section.

```
dsv config edit
```

Add *gcloud:gce-test* as a user to the **Default Admin Policy**. Third-party accounts must be prefixed with the provider name; in this case, the fully qualified username would be *glcoud-gce:gce-test*.

> **Note:** Adding a user to the admin policy is not security best practices. This is for example purposes only. Ideally, you would create a separate policy for this AWS user with restricted access. For details on limiting access through policies, see the Policy section.

```
dsv config edit -e yaml
```

```
<snip>
- actions:
 - <.*>
 conditions: {}
 description: Default Admin Policy
 effect: allow
 id: xxxxxxxxxxxxxxxxxxxx
 meta: null
 resources:
 - <.*>
 subjects:
 - users:<gcloud-gce:gce-test|admin@company.com>
<snip>
```

GCE Authentication

SSH into the GCE and download the latest DSV CLI from this website DSV CLI.

For example, `curl: https://dsv.secretsvaultcloud.com/downloads/cli/1.31.0/dsv-linux-x64 -o dsv`

You may need to give yourself permissions to run the DSV binary, and it is also easier if you set the path.

Run `dsv init` filling out the desired values and selecting **6** GCP (federated) when prompted for the auth type.

```
Please enter auth type:
```

```
(1) Password (local user)(default)
(2) Client Credential
(3) #{ThycoticOne}# (federated)
(4) AWS IAM (federated)
(5) Azure (federated)
(6) GCP (federated)
(7) OIDC (Federated)
```

Run `dsv auth` to verify authentication. A token will be displayed.

Run `dsv secret read <path to any secret>` to verify secret access.

Google Kubernetes Engine (GKE) Authentication

It follows that if you can have a GCE (aka a virtual server) authenticate to DSV, that there would be a similar way to do that with a Google Kubernetes Engine (GKE) node.

Here is an example where we deploy a simple app in GKE that is able to authenticate to DSV.

In the GCE example above, we used the **Compute Engine default service account**. Here we suggest you create a service account with at least the `storage.objectViewer` role for the project which will enable the ability to pull an image from GCP registry. In this example, we created a service account named `dsv-gce`.

DSV Authentication provider

Using any computer with Admin DSV access, setup the DSV Authentication Provider.

Create a file named 'auth-gcp.txt' in the following format and substituting your GCP `<ProjectID>`.

```
{
  "name": "gcloud-gce",
  "type": "gcp",
  "properties": {
    "ProjectId": "myfirstproject-273119"
  }
}
```

Run `dsv config auth-provider create --data @auth-gcp.txt` to implement the Authentication Provider.

DSV User mapped to the GKE service account

Run `dsv user create --username gce-test --provider gcloud-gce --external-id {dsv-gce service account email}` using the default service account email we saved earlier. You will get a response like this.

```
{
  "created": "2020-04-09T12:59:44Z",
  "createdBy": "users:thy-one:admin@company.com",
  "externalId": "dsv-gce@gcp-project-id.iam.gserviceaccount.com",
  "id": "19709b4e-2a13-4164-a930-81997b568036",
```

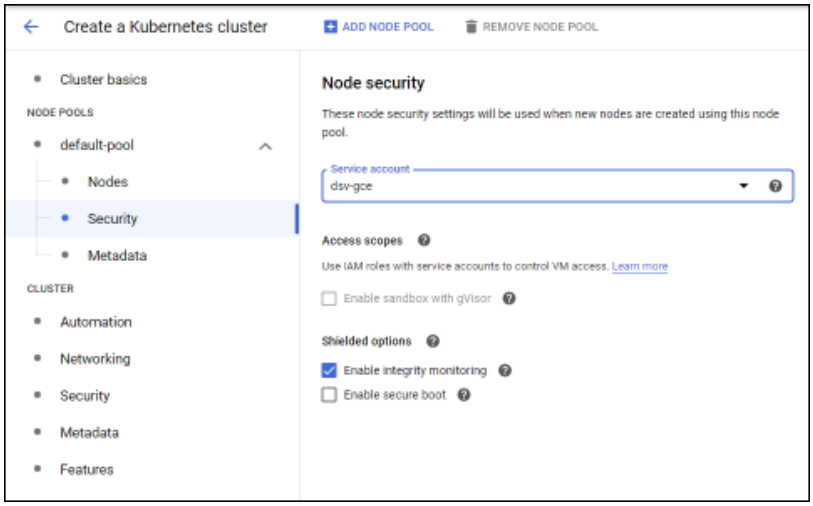
```
  "lastModified": "2020-04-09T12:59:44Z",
  "lastModifiedBy": "users:thy-one:admin@company.com",
  "provider": "gcloud-gce",
  "userName": "gce-test",
  "version": "0"
}
```


Back to GCP to setup a GKE cluster

From the **GCP Home** page, in the left menu, hover over **Kubernetes Engine** and select **Clusters**. Then **Create Cluster**. If this is the first one, then GCP will enable the GKE API for you.

When the form comes up, the default values can be used with the exception of the service account. To change this, in the left navigation, select **default-pool** then **Security** where you will select the service account `dsv-gce` just mentioned.

Click **Create**. It takes a few minutes for the cluster to be built.

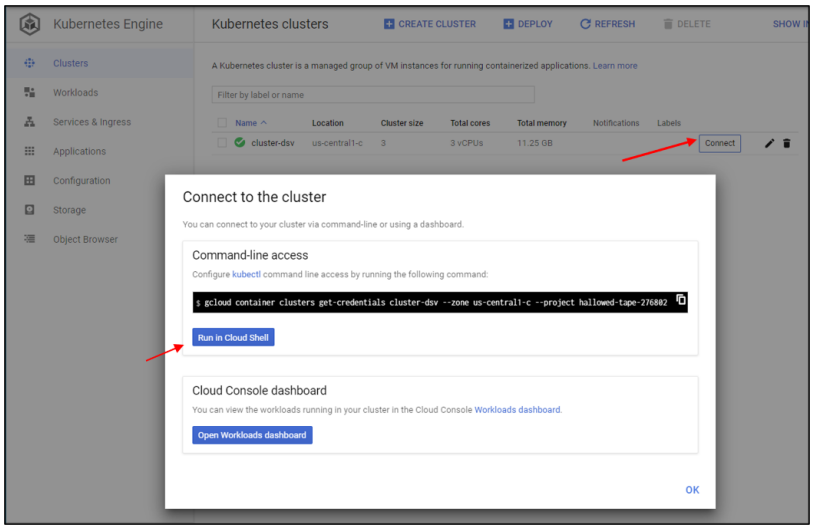


Hello-App

Now create and deploy this Go-based hello app in this cluster node.

We will use the built-in GCP Cloud shell to connect since it comes with Docker, Kubectl, and connectivity to GCP all setup. It even has a nice editor for the files we will create. To do this, go to the **Kubernetes Engine** then **Clusters** page. From the list, there is a **Connect** button that opens a modal pop-up. In the modal, select **Run in Cloud Shell**.

Usage



A terminal opens in the browser. Run the following steps.

```
mkdir hello-app
cd hello-app
cat >  main.go
```

Now you can copy the code below into the terminal, but substitute the `tenant_url` to your URL, which will look something like `https://mycompany.secretsvaultcloud.com`.

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
)

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/", hello)

    port := os.Getenv("PORT")
    if port == "" {
        port = "8080"
    }

    log.Printf("Server listening on port %s", port)
    log.Fatal(http.ListenAndServe(":"+port, mux))
```

```
}

func hello(w http.ResponseWriter, r *http.Request) {
    log.Printf("Serving request: %s", r.URL.Path)

    fmt.Println("-----------computeMetadata-----------")

    client := &http.Client{}
    req, err := http.NewRequest("GET",
"http://metadata.google.internal/computeMetadata/v1/project/project-id", nil)
    if err != nil{
        fmt.Fprintf(w, "Error creating Metadata Request: %s\n", err.Error())
        return
    }
    req.Header.Add("Metadata-Flavor", Google)
    resp, err := client.Do(req)
    if err != nil{
        fmt.Fprintf(w, "Error creating Metadata : %s\n", err.Error())
        return
    }

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil{
        fmt.Fprintf(w, "Error parsing body computeMetadata: %s\n",  err.Error())
        return
    }
    fmt.Fprintf(w, "Response computeMetadata: %s\n", string(body))

    fmt.Println("-----------computeMetadata-service-accounts-----------")

    tenant_url := "{tenant url}"
    client2 := &http.Client{
    }
    req2, err := http.NewRequest("GET",
"http://metadata.google.internal/computeMetadata/v1/instance/service-
accounts/default/identity", nil)
    if err != nil{
        fmt.Fprintf(w, "Error creating service-accounts Metadata Request: %s\n", err.Error
())
        return
    }
    req2.Header.Add("Metadata-Flavor", Google)
    q := req2.URL.Query()
    q.Add("audience", tenant_url)
    q.Add("format", "full")
    req2.URL.RawQuery = q.Encode()
    resp2, err := client2.Do(req2)
    if err != nil{
        fmt.Fprintf(w, "Error creating service-accounts Metadata : %s\n", err.Error())
        return
    }

    body2, err := ioutil.ReadAll(resp2.Body)
```

```
    if err != nil{
        fmt.Fprintf(w, "Error parsing body service-accounts computeMetadata:
%s\n",  err.Error())
        return
    }
    fmt.Fprintf(w, "Response service-accounts computeMetadata: %s\n", string(body2))

    fmt.Println("-----------DSV-----------")

    reqBody, _ := json.Marshal(map[string]string{
        "grant_type" : "gcp",
        "jwt" : string(body2),
    })

    dsvResp, err := http.Post(tenant_url+"/v1/token","application/json", bytes.NewBuffer
(reqBody))
    if err != nil || dsvResp == nil{
        if err!= nil {
            fmt.Fprintf(w, "Error creating dsv Request: %s\n", err.Error())
        }
        return
    }

    dsvBody, err := ioutil.ReadAll(dsvResp.Body)
    if err != nil{
        fmt.Fprintf(w, "Error parsing body dsv: %s\n",  err.Error())
    } else{
        fmt.Fprintf(w, "Response from DSV: %s\n", string(dsvBody))
    }
}
```

Use `ctrl+c` to escape out.

Now create the docker file.

```
cat > Dockerfile
```

Copy the commands below in.

```
FROM golang:1.13-alpine
ADD . /go/src/hello-app
RUN go install hello-app

FROM alpine:latest
COPY --from=0 /go/bin/hello-app .
ENV PORT 8080
CMD ["./hello-app"]
```

Use `ctrl+c` to escape out.

Usage

Run these commands to build and push the app to GKE. Substitute your `project ID` in.

```
docker build -t gcr.io/{PROJECT_ID}/hello-app:v1 .
docker push gcr.io/{PROJECT_ID}/hello-app:v1
```

The docker image is in GCP registry, so now create the Kubernetes deployment.

```
cat > k8.yml
```

Substitute your `project id` and paste the following.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      name: my-app
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app
          image: gcr.io/{PROJECT_ID}/hello-app:v1
          volumeMounts:
            - name: certs
              mountPath: /etc/ssl/certs
      volumes:
        - name: certs
          hostPath:
            path: /etc/ssl/certs
```

Use `ctrl+c` to escape out.

And deploy:

```
kubectl apply -f k8.yml
```

Make sure the pod is in running status.

Usage

```
kubectl get pod
```

Now expose the app to the internet.

```
kubectl expose deployment my-app --type=LoadBalancer --port 80 --target-port 8080
kubectl get service
```

You should see:

```
root@THY-01-0250-LT:/mnt/c/Users/masres/repo/hello-app#  kubectl get service
NAME         TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.8.0.1     <none>        443/TCP        10d
my-app       LoadBalancer   10.8.0.130   <pending>     80:32628/TCP   3s
```

It will take a few minutes for the `<pending>` to turn to an IP address.

Retry `kubectl get service` until you see IP address in EXTERNAL-IP.

```
NAME         TYPE           CLUSTER-IP   EXTERNAL-IP    PORT(S)        AGE
kubernetes   ClusterIP      10.8.0.1     <none>         443/TCP        10d
my-app       LoadBalancer   10.8.0.130   34.66.218.89   80:32628/TCP   65s
```

Copy the EXTERNAL-IP for my-app and paste in your browser. You should a get DSV token.

At this point you are successfully logged into DSV from GKE. There are two tokens, the first one is the GKE metadata token. The second one is the DSV authentication token. If you parse the DSV token at the jwt.io website you should see the username `gcloud-gce:gce-test` to confirm.

Authentication: OIDC

Use `dsv config auth-provider search --encoding yaml` to see your current authentication settings.

The initial auth settings after your tenant is provisioned should look like this:

```
data:
- created: "2020-04-27T18:04:52Z"
  createdBy: ""
  id: bqjhth447csc72i4sm8g
  lastModified: "2020-04-27T18:04:52Z"
  lastModifiedBy: ""
  name: thy-one
```

```
    properties:
      baseUri: https://login.thycotic.com/
      clientId: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      clientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    type: thycoticone
    version: "0"
  length: 1
  limit: 25
```

OIDC Providers

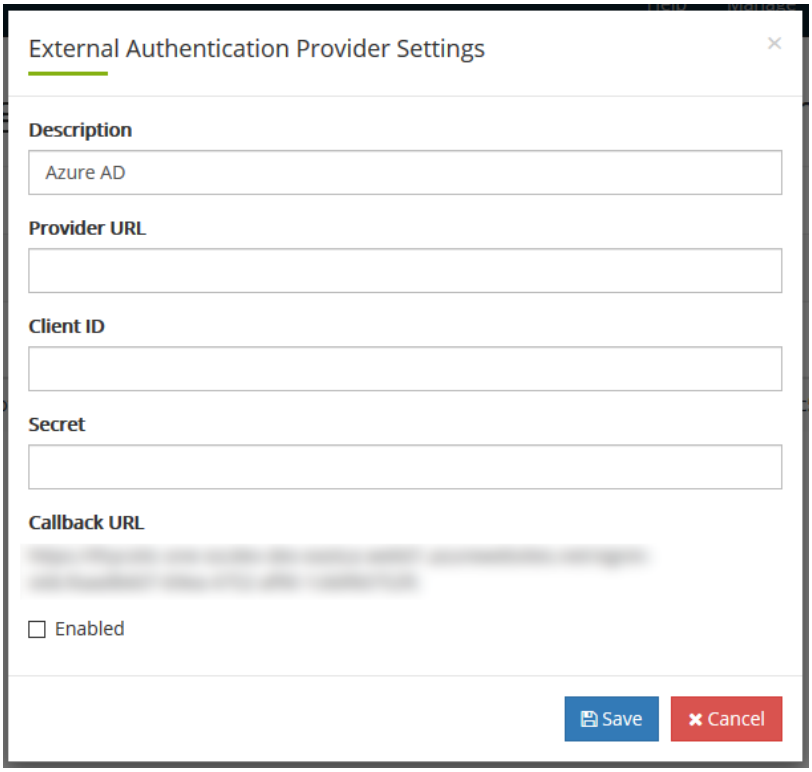
Any OIDC compliant authentication provider should be configurable to work with Thycotic One and DevOps Secrets Vault. Documented integrations are below.

Common Steps

For all OIDC authentication providers you will need to get their provider URL, client id, and client secret. You will need to set in the authentication provider the callback URL that it will redirect to once authentication is complete.

To get your callback URL:

1. Sign into the cloud manager portal and go to Manage | Teams and click **Organizations** for your team.
2. Click **Auth Providers** and then click **New**. This opens a dialog.
3. Give it a name and copy the Callback URL provided. Do not save or cancel. You will be coming back to fill out the rest of the fields.



Post OIDC Configuration Steps

Creating a User in Thycotic One and DSV

In order to log in using OIDC, the user must exist in the external provider, Thycotic One, and in DSV.

If your current user, such as your initial admin already exists in all places, then skip this section. If you want to add another user to Thycotic One and DSV simultaneously, do the following steps.

1. In the DSV CLI run `dsv user create --username useremail@company.com --provider thy-one`.
2. This creates a user record in DSV and syncs it to Thycotic One. The User will get an email with a link to establish their password.
3. In the cloud manager portal, you can see your users by logging in and clicking the **Users** link.

Logging In

Initialize the CLI.

```
dsv init
```

Add a new profile if you want to retain your default `dsv` profile.

When prompted for the authorization type, choose *OIDC (federated)*.

```
Please enter auth type:
       (1) Password (local user)(default)
       (2) Client Credential
       (3) #{ThycoticOne}# (federated)
       (4) AWS IAM (federated)
       (5) Azure (federated)
       (6) GCP (federated)
       (7) OIDC (federated)
```

When prompted for the authentication provider, press Enter to accept the default of `thy-one`

If you are on Windows or Mac OS, the CLI should automatically open a browser to the Google login page, otherwise it will print out a URL that you can copy and paste into a browser to complete the process.

Log in using your Google credentials and your browser will redirect to `http://localhost:8072/callback`. The CLI is listening on that port and will submit the returned authorization code to DSV to finish the login process.

Verify the login by running (omit the --profile flag if you overwrote your config).

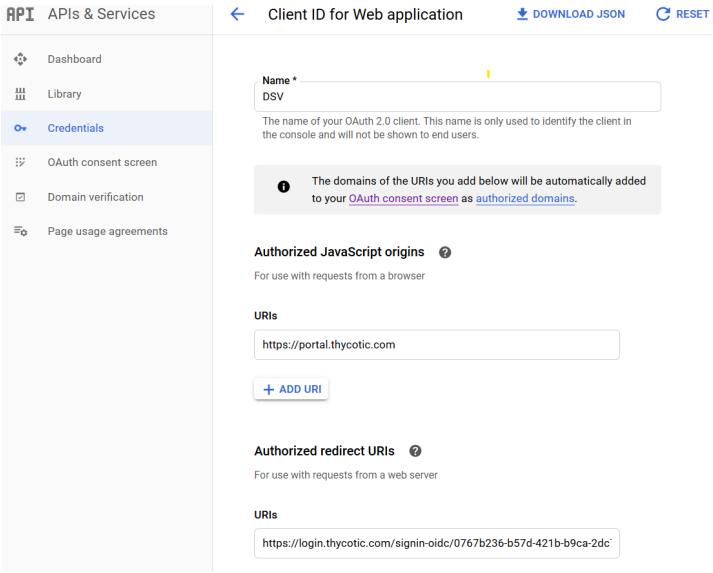
```
dsv auth --profile profilename
```

Google Identity Provider Example

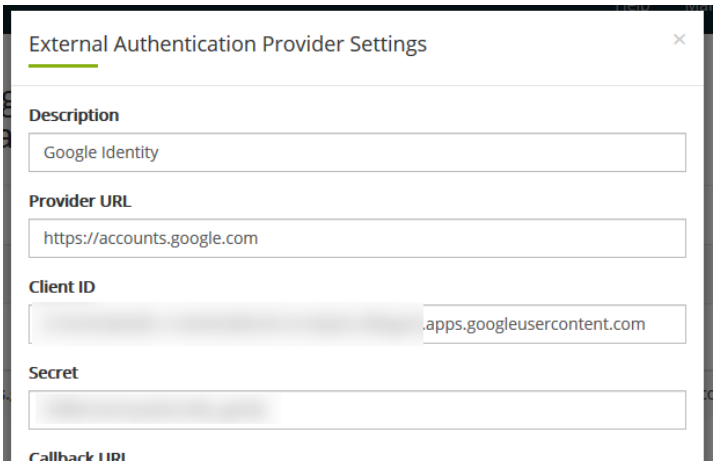
Configure Auth Providers

This example uses the Google Cloud Identity service.

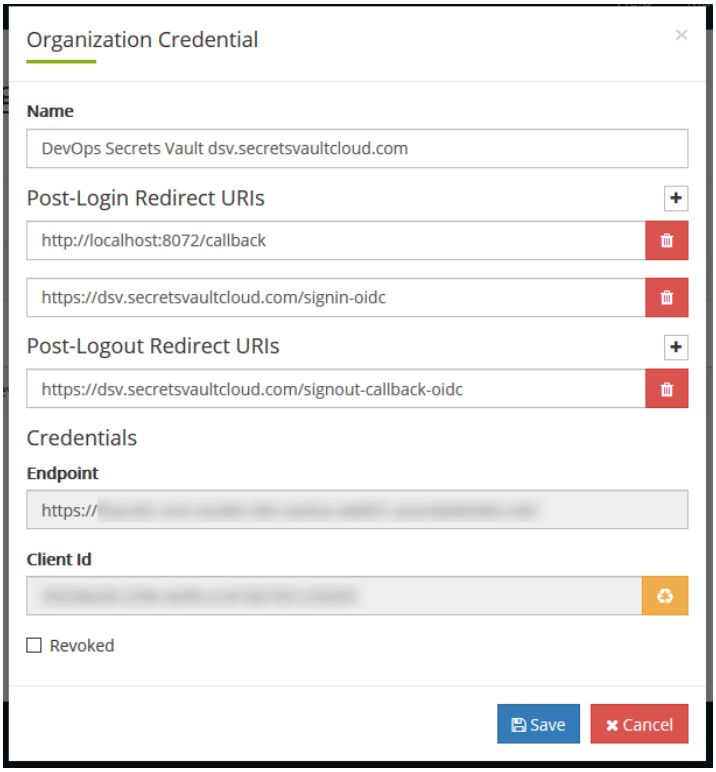
1. Get the callback URL from Thycotic One# following the directions at Authentication:OIDC.
2. Go to the Google Cloud API Console and select a project if needed.
3. Select Credentials and click Create Credentials and click OAuth Client ID.
4. Choose **Web Application**.
5. Enter the information, setting the Authorized origin as `https://portal.thycotic.com/` and Authorized redirect as the callback URL copied from the Thycotic cloud manager portal. Follow the instructions to add these URL's to the OAuth consent screen.



6. Save and copy the client id and client secret from the dialog into the credentials create dialog in Cloud Manager. Your **Provider URL** in cloud manager should be set to `https://accounts.google.com`

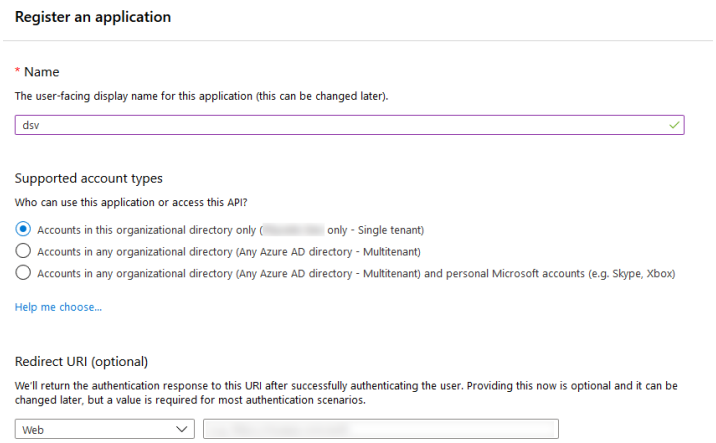


7. Save the credential create dialog in cloud manager and go back to Organizations. Click **Credentials** and then edit your Credential. This is what is used by DSV to connect to the Thycotic One identity provider for authentication.
8. Verify that there is a **Post-Login** Redirect URI for `http://localhost:8072/callback`. If there isn't, add one. This is the callback used when logging into DSV with the CLI.

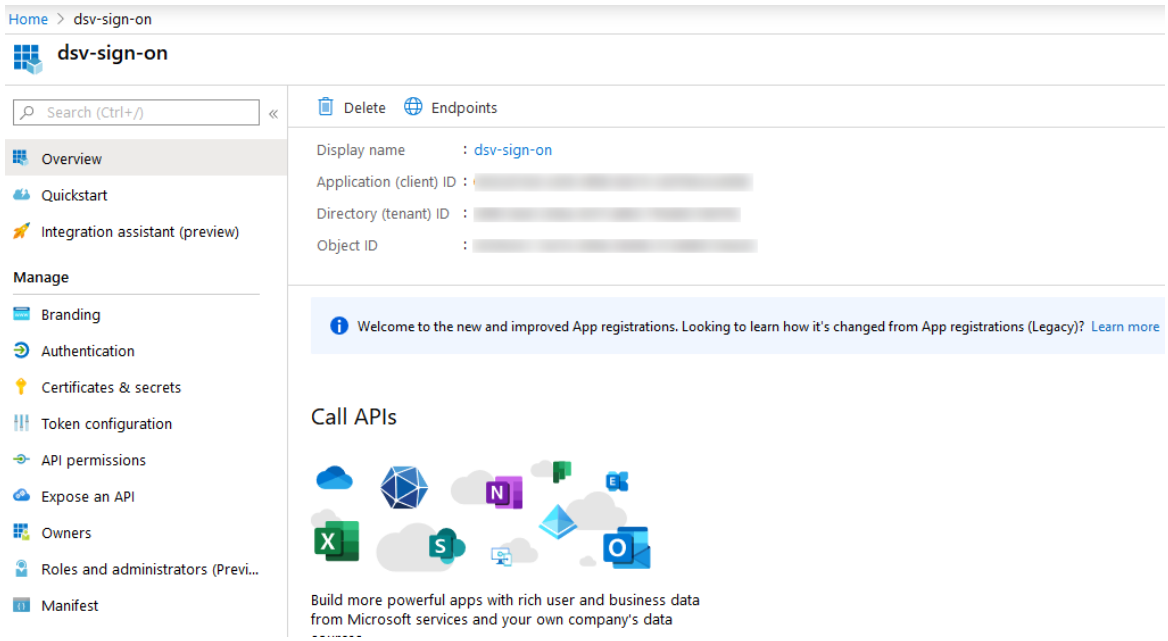


Azure AD OIDC Example

1. Get the callback URL from Thycotic One following the directions at Authentication:OIDC
2. In your azure portal go to **Azure Active Directory** and then go to the App Registrations.
3. Click **New Registration**
4. Give your app a name and add the Callback URL from Thycotic One as the Redirect URI.



5. Click **Register** to save your app.
6. Go to your app's **Certificates and Secrets** and click **New Client Secret**.
7. Set the time period for the secret and click **Add**.
8. Copy the client secret, note that it will not be available after you leave the page.
9. Go to **Authentication** and check the box for **ID Tokens** in the implicit grant section and save.
10. Navigate to **Overview** and note the Application ID and Directory ID. The Application ID is your Client ID for Thycotic One and the Directory ID will be part of your provider URL in the format https://login.microsoftonline.com/{directory id}.



11. Go back to the open dialog in Thycotic One and enter the Application ID for the Client ID, the generated secret for Client Secret, and fill in the Provider URL and click **Save** Thycotic One.
12. When you sign into Thycotic One again you should now see an option for logging in with Azure AD.

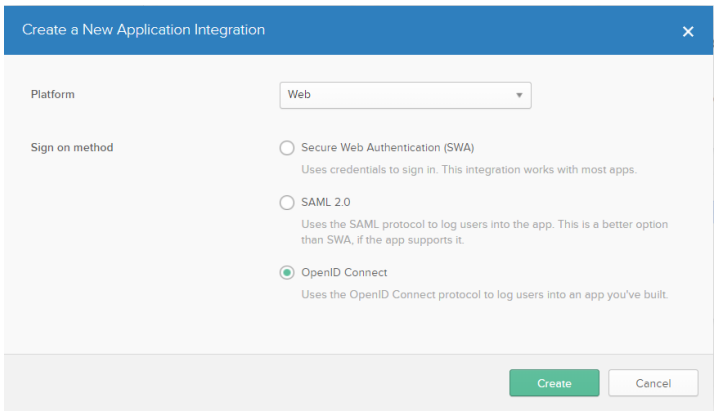
Okta Identity Provider Example

This example uses Okta as a OIDC identity provider.

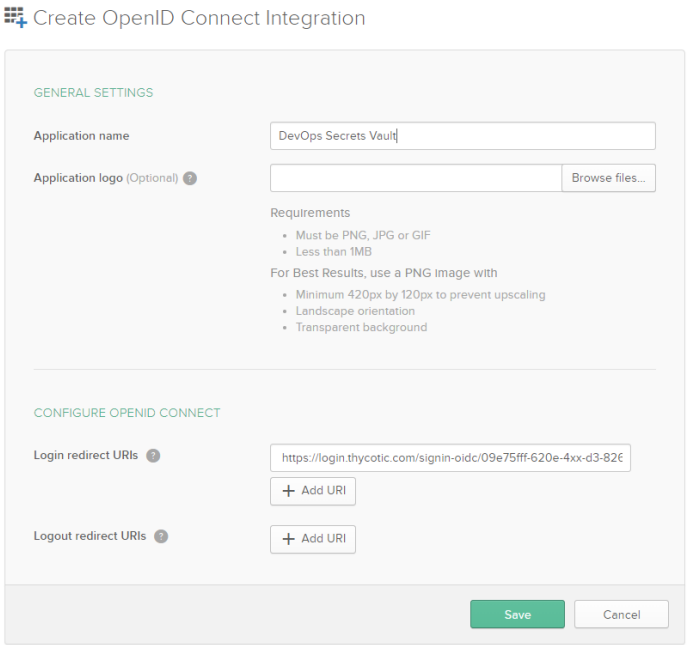
Okta OIDC connection

1. Get the callback URL from Delinea's Cloud Manager portal following the directions at Authentication:OIDC.
2. Log in to your Okta Admin console.
3. From the top menu bar, select **Applications**.
4. Select **Add Application**.
5. At the top right, select **Create New App**. A window opens.
6. For platform, select **Web** from the drop-down and the **OpenID Connect** radio button. Click **Create**.

Usage

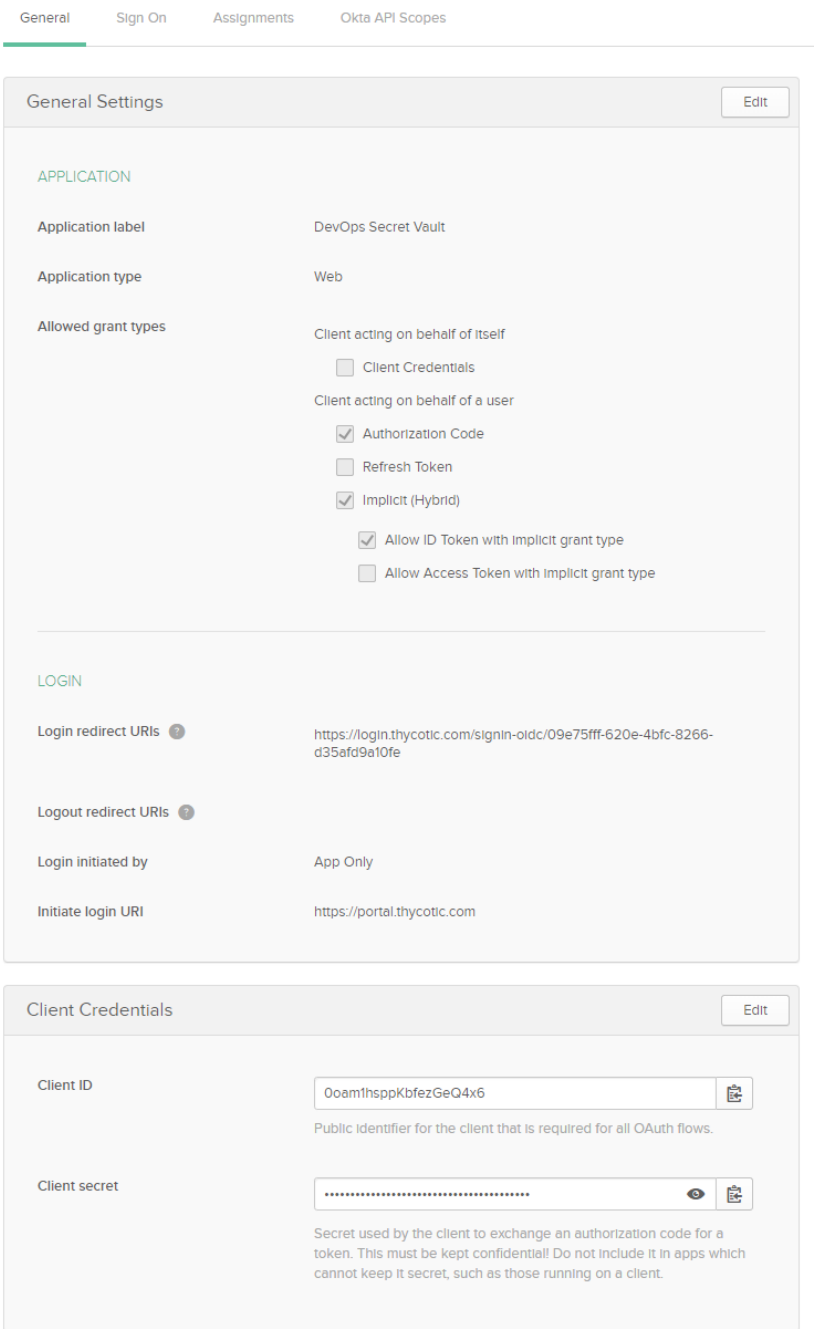


7. On the resulting screen, provide an **Application name** and optional logo. Enter the Delinea callback URL in the box labeled **Login redirect URIs**. Click **Save**.



8. To the right of General Settings click **Edit**. Check the **Implicit (Hybrid)** box and it will expand. Then check **Allow ID Token with Implicit grant type**.
9. In the **Initiate login URI**, Okta defaults to copying the Login Redirect URI, so highlight that box and copy `https://portal.thycotic.com`. Click **Save**.
10. Copy the Client ID and Client secret for entry into the Delinea Cloud portal

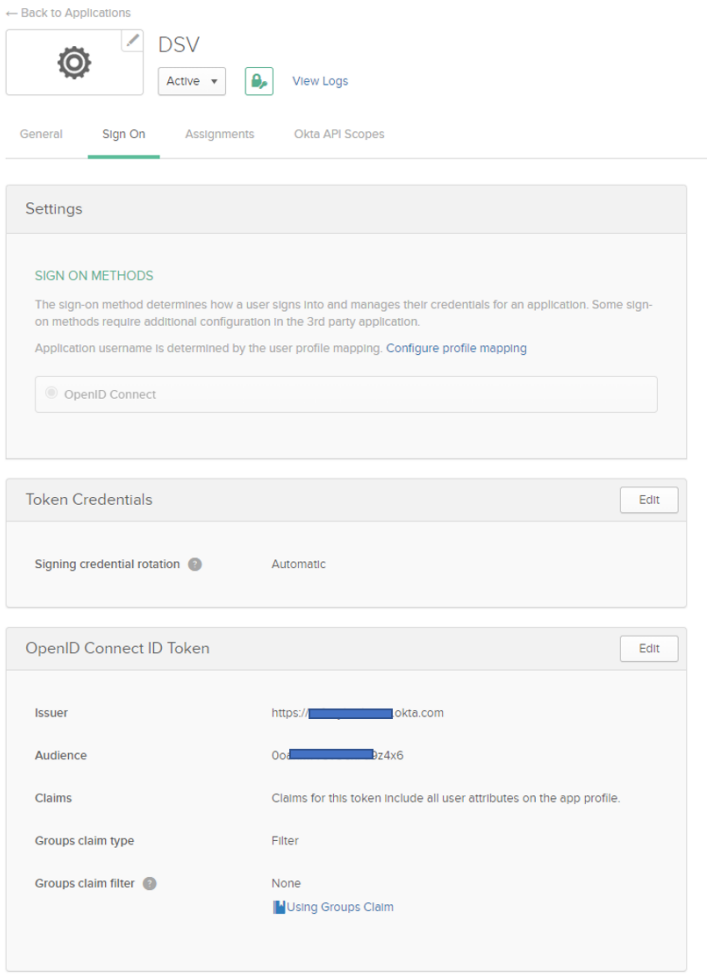
Usage



Retrieve the Issuer URL

11. In the second menu bar from the top, click **Sign On** and in the third box down, OpenID Connect ID Token, take note of the URL by **Issuer**. Enter this into the Delinea Cloud portal. It will be something like `https://company.okta.com` or `https://company.oktapreview.com`.

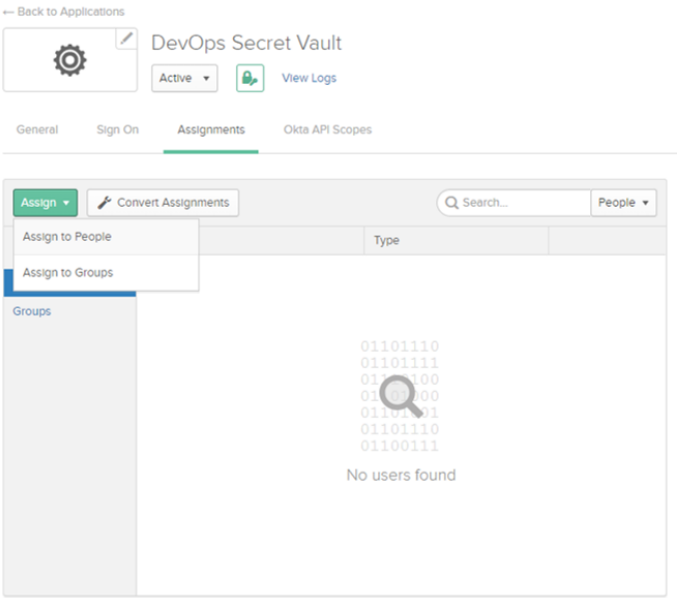
Usage



Add Okta Users and Groups to the DSV Application

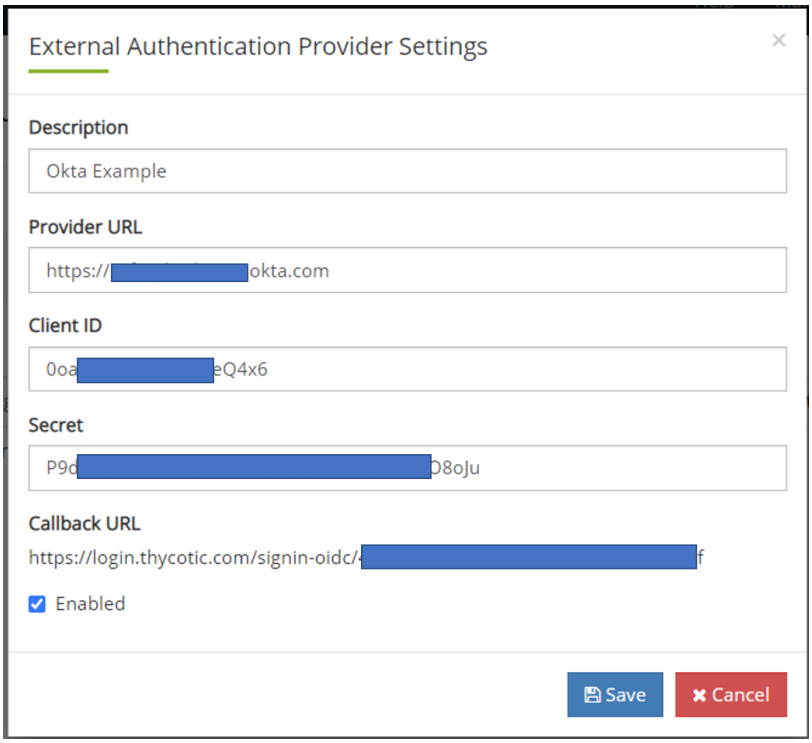
12. In second menu bar from the top, click **Assignments**
13. Click **Assign** and when it drops down, add users and/or groups that will use DevOps Secrets Vault. Of course, you can always come back and add/remove people as needed.

Usage



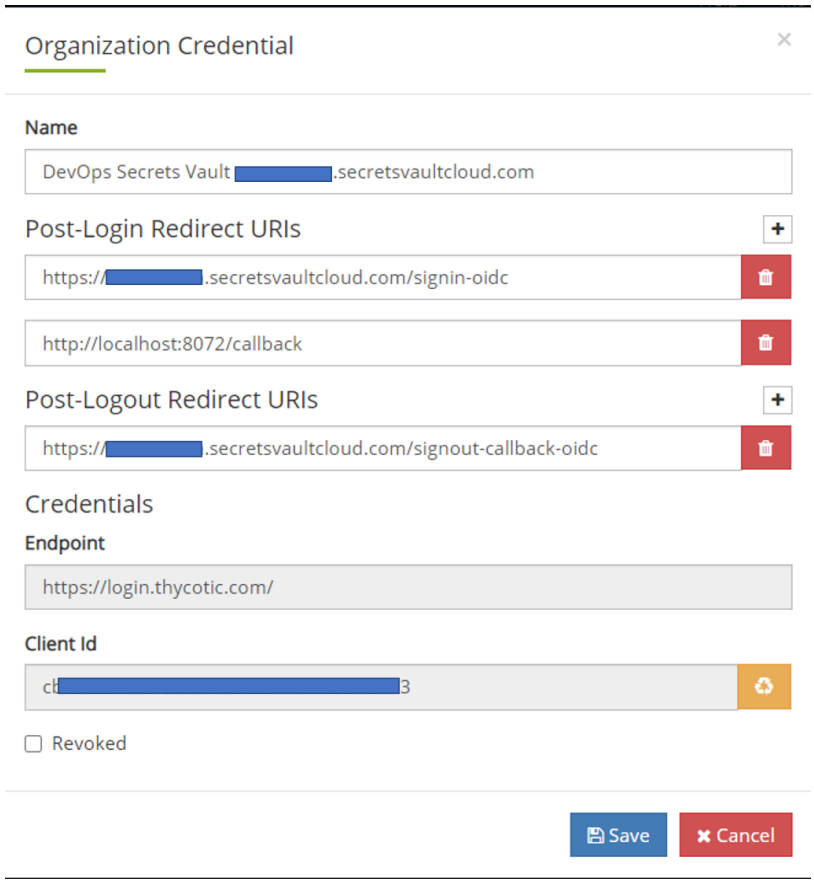
Finish the Connection on the Delinea One side

14. Go back to the Delinea Cloud Manger Portal where we started. Provide a **Description** and the issuer/provider URL from step 11.
15. Provide the **Client ID** and **Client Secret** from step 10.
16. Check **Enable**.
17. Click **Save**.



18. Click **Back to Organizations**.
19. Click **Credentials**.
20. Click **Edit**.
21. In the dialog that appears, and to the right of **Post-Login Redirect URIs**, click the **+**. In the prompt that appears, type `http://localhost:8072/callback`.

> **Note:** If you have already added this call back for another auth provider, then it should still be there so you can skip these last steps (18-21).



Authentication: Certificate

Authentication by certificate uses two API calls and does not send a private key.

Prerequisites

Authenticating with a certificate requires a certificate and a corresponding role. The user with this role can be authenticated using a leaf certificate that contains role as a description field.

Role

First, a role is needed. Use this command to create a role.

```
dsv role create --name certauth
```

Or, use an existing role.

Certificate

In this step, generate a root certificate for signing leaf certificates. The root certificate can issue leaf certificates with different roles.

Usage

```
dsv pki generate-root --rootcapath root-for-auth --common-name root.auth --domains
root.system.a,root.system.b --maxttl 168
```

After that, prepare a client certificate with a corresponding role.

```
dsv pki leaf --common-name root.system.a --rootcapath root-for-auth --description certauth
```

The output should present a generated certificate, private key and SSH public key. The certificate and the private key are required for authentication and must be saved.

CLI Configuration

After you've configured everything, you can initialize the CLI configuration. For that, run:

```
dsv init
```

> **Note:** for testing purposes I recommend to create a separate profile when running the `dsv init` command and after in all commands for testing use `--profile=your-profile-name` flag.

When prompted for the authentication type, choose **x509 Certificate**:

```
Please enter auth type:
        (1) Password (local user) (default)

        (2) Client Credential

        (3) Thvcotic One (federated)

        (4) AWS IAM (federated)

        (5) Azure (federated)

        (6) GCP (federated)

        (7) OIDC (federated)

        (8) x509 Certificate
```

When prompted, input your certificate and the private key. Note that CLI only sends the certificate for authentication. Private key will not be sent over the wire, and is used only to decrypt data to prove ownership of the private key to the server.

Dynamic Secrets

Dynamic secrets are automatically generated at the time of request. This differs from the standard Secret store read request where the credentials remain the same until changed by a user. They can be used when you need to provide credentials to a user or resource, like a configuration tool, but the access should expire after a set period of time.

Supported Types:

IaaS Dynamic Secrets

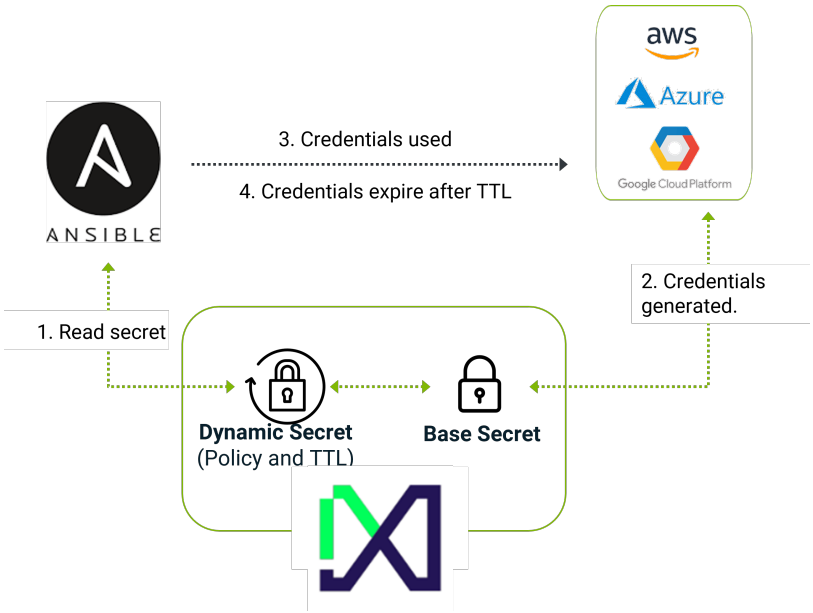
- AWS
- Azure AD Graph
- Azure MS Graph
- GCP

Database Dynamic Secrets

- MSSQL
- MySQL
- Oracle
- PostgreSQL
- MongoDB

Linking

In order for dynamic secrets to be generated, they rely on a base secret stored in DSV that contains the provider's credentials that are used to automatically generate the ephemeral access keys.



The linking is done through the `attributes` section in the secret JSON. For example, the following secret `temp-api` has no data, but is linked to a different AWS IAM secret that contains the access and secret key information. The `linkConfig` defines the type of linking and the linked secret path.

Attribute	Description
linkConfig	link type and path to the linked secret

Attribute	Description
linkConfig.linkType	the only valid value is "dynamic"
linkConfig.linkedSecret	secret path to the base credential

```
{
    "id": "cc619722-6538-4891-b0a6-2c7fa1776a67",
    "path": "dynamic:aws:creds:temp-api",
    "attributes": {
        "linkConfig": {
            "linkType": "dynamic",
            "linkedSecret": "base:aws:creds:api-account"
        }
    },
    "description": "",
    "data": {
    }
}
```

Search for linked Secrets

To get a list of all dynamic secrets linked to a base secret, issue the command.

```
dsv secret search --query <base secret path> --search-links
```

IaaS Dynamic Secrets

DSV currently supports dynamic secrets for:

- AWS
- Azure AD Graph
- Azure MS Graph
- GCP

AWS Dynamic Secrets

AWS Dynamic Secrets generate a temporary access key, secret key, and session token. AWS Security Token Service (STS) provides for either `federate` or `assumeRole`. `federate` and is ideal for assigning dynamic secrets from a single AWS account. `assumeRole` allows cross account access in AWS, so a single set of credentials in DSV can grant access to multiple AWS accounts.

These are the links to AWS documentation for each STS type.

- Federate
- Assume Role

AWS Federate

Setup the AWS IAM User

For the `federate` example, create a new IAM User and note the access key and secret key.

Assign a policy to the IAM user with `sts:GetFederationToken` permission as well as any other permissions the IAM user should have. In this example, we assign the user full CodeDeploy rights.

> **Note:** When you get temporary tokens from AWS via `GetFederationToken` the resulting token's permissions will be the intersection of the IAM User and the policy ARN specified on the dynamic decret. In other words, the dynamic secret is only allowed the permissions that are in both the IAM policies and the dynamic secret attached policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:GetFederationToken",
                "codedeploy:*"
            ],
            "Resource": "*"
        }
    ]
}
```

Create the Base Secret

Next create a secret in DSV with the AWS IAM user access key, secret key, and region.

Create a file named `secret_root.json` substituting your values.

```
{
        "accessKey": "youraccesskey",
        "region": "us-east-1",
        "secretKey": "yoursecretkey"
}
```

Create the secret via the CLI at a path of your choosing.

```
dsv secret create --path aws/base/api-account --data @secret_root.json --attributes '{"type":
"aws"}'
```

Create the Dynamic Secret

Attribute	Description
policyArn	AWS ARN of the policy to assign the federated user token. Can be customer or AWS managed.
providerType	`federate`

ttl	optional time to live in seconds of the generated token. If none is specified it will default to the minimum of 900.

Note: If the TTL is set to less than 900 seconds, AWS will fail to create the token.

Now, you need to create a dynamic secret, which points to the base secret via its attributes. The dynamic secret doesn't have any data stored in it because data is only populated when you read the secret.

Create an attributes JSON file named *secret_attributes.json*, substituting your values.

```
{
       "linkConfig": {
               "linkType": "dynamic",
               "linkedSecret": "aws:base:api-account"
       },
       "policyArn": "arn:aws:iam::aws:policy/AWSCodeDeployReadOnlyAccess",
       "providerType": "federate",
       "ttl": 1200
}
```

Create a new Dynamic Secret

```
dsv secret create --path dynamic/aws/federate-api --attributes @secret_attributes.json
```

Now, anytime you read the dynamic secret, the data is populated with the temporary AWS access credentials.

```
dsv secret read --path dynamic/aws/federate-api
```

This returns a result like:

```
{
  "attributes": {
    "linkConfig": {
      "linkType": "dynamic",
      "linkedSecret": "aws:base:api-account"
    },
    "policyArn": "arn:aws:iam::aws:policy/AWSCodeDeployReadOnlyAccess",
    "providerType": "federate",
    "ttl": 1200
  },
  "data": {
    "accessKey": "youraccesskey",
    "expiration": "2020-02-06T18:49:17Z",
    "secretKey": "yoursecretkey",
    "sessionToken": "yoursessiontoken",
    "ttl": 1200
  },
  "description": "",
  "id": "yourId",
  "version": "0"
}
```

You can validate the credentials only grant read access to Code Deploy by putting the credentials in a python script and attempting to create a Code Deploy application:

```
import boto3
import json
from botocore.exceptions import ClientError
```

```
sess = boto3.Session(
    aws_access_key_id="accesskeyid",
    aws_secret_access_key="secretaccesskey",
    aws_session_token="yoursessiontoken"
)

client = sess.client("codedeploy")
resp = client.list_applications()
print("----list code deploy apps----")
print(json.dumps(resp["applications"], indent=4))

print("----create code deploy app----")
try:
    resp = client.create_application(
        applicationName="TestApp",
        computePlatform="Server"
    )
except ClientError as e:
    print(e.response["Error"]["Code"])
```

The result should look something like this (depending on how many CodeDeploy apps exist):

```
----list code deploy apps----
[
    "ExampleApp"
]
----create code deploy app----
AccessDeniedException
```

AWS Assume Role

In this example, we assume the IAM user and the role that the user will assume are in separate AWS accounts. This is not required, but then it might make more sense to use the `sts:Federated`approach.

Setup the AWS IAM user

In the AWS account for the IAM user, create or modify an IAM user policy to include the `sts:AssumeRole` permissions as well as any other permissions the user should have. In this example, we assign the user full `CodeDeploy` rights.

> **Note:** For setting up, if you don't know the role account ID or name at this point, **Resources** could be set to all `*`, but best practices would be to come back and restrict the **Resources** to only the role once the name is known as shown here.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
```

```
            ],
            "Resource": "arn:aws:iam::{account id of role}:role/{role-name}"
        }
    ]
}
```

Setup the AWS IAM role

In the AWS account with the role that is to be used, create a new Role or identify an existing one with the proper policies (not shown here).

> **Note:** The `sts:AssumeRole` token will have permissions that intersect between the IAM user policy(ies) and the role ploicy(ies) they assume. In other words, the token can't have permissions enabled by both the user and role policies.

Additionally, this role must have a trust relationship setup between the IAM user in the first account and this role. It might look like this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{account id of user}:{iam-user}"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

Create the Base Secret

Next, create a secret in DSV with the AWS IAM user access key, secret key, and region.

Create a file named `secret_root.json` substituting your values.

```
{
        "accessKey": "youraccesskey",
        "region": "us-east-1",
        "secretKey": "secretkey"
}
```

Create the Secret via the CLI at a path of your choosing.

```
dsv secret create --path aws/base/api-account --data @secret_root.json --attributes '{"type":
"aws"}'
```

Create the Dynamic Secret

Attribute	Description
roleArn	AWS ARN of the role to assign the AssumeRole user token. Can be customer or AWS managed.

providerType	`assumeRole`
ttl	optional time to live in seconds of the generated token. If none is specified will default to 900.

Create the Dynamic Secret

Now you need to create a dynamic secret which points to the base secret via its attributes. The dynamic secret doesn't have any data stored in it. Data is only populated when you read the secret.

Create or update the attributes json file named `secret_attributes.json substituting the ARN of the role you created.

```
{
       "linkConfig": {
               "linkType": "dynamic",
               "linkedSecret": "aws:base:api-account"
       },
       "roleArn": "arn:aws:iam::{account id of role}:role/{role-name}",
       "providerType": "assumeRole",
       "ttl": 1200
}
```

Now, create the dynamic secret in the CLI using the JSON above.

```
dsv secret create --path dynamic/aws/assume-api --attributes @secret_attributes.json
```

Now, anytime you read the dynamic secret, the data is populated with the temporary AWS access credentials.

```
dsv secret read --path dynamic/aws/assume-api
```

This returns a result like:

```
{
  "attributes": {
    "linkConfig": {
      "linkType": "dynamic",
      "linkedSecret": "aws:base:api-account"
    },
    "roleArn":  "arn:aws:iam::{account id of role}:role/{role-name}",
    "providerType": "assumeRole",
    "ttl": 1200
  },
  "data": {
    "accessKey": "youraccesskey",
    "expiration": "2020-02-06T18:49:17Z",
    "secretKey": "yoursecretkey",
    "sessionToken": "yoursessiontoken",
    "ttl": 1200
  },
  "description": "",
  "id": "yourid",
  "version": "0"
}
```

AAD Graph Dynamic Secrets

Note: As of June 30th, 2020, Microsoft stopped updating Azure Active Directory in favor of Azure Microsoft Graph. Starting June 30th, 2022, all support and updates for Azure AD Graph will end, and endpoints will no longer send responses. Delinea*strongly recommends* using the MS Graph API. See: Azure Microsoft Graph to get started using DSV with MS Graph.

DevOps Secrets Vault relies on Azure service principals to provide dynamic secrets.

In order for DSV to generate dynamic secrets, a base secret must first be created using a service principal that has permissions to manage other service principals. Those permissions include:

- "Owner" role for the subscription scope
- "Read and write all applications" permission in Azure Active Directory.
- Your account must have Microsoft.Authorization/*/Write access to assign an active directory application to a role.

These permissions can be configured through the Azure Portal, CLI tool, or PowerShell. A guide to setting up the Azure service principals in the Azure portal is provided in the Azure Service Principal section.

Create the Base Secret

The base secret holds the credentials required for DSV to perform API calls to Azure to query roles and create/delete service principals.

Attribute	Description
subscriptionId	Required - The subscription ID holding the resources you wish to access using Azure Active Directory.
tenantId	Required - The tenant ID for Azure Active Directory. Azure lists it in places as *Directory (tenant) ID*.
clientId	Required - The OAuth2 client ID to connect to Azure. Azure lists it in places as *Application (client) ID*.
clientSecret	Required - The OAuth2 client secret to connect to Azure.
environment	Optional - The Azure environment. If not specified, DSV will use Azure Public Cloud.

Create a file named `secret_base.json` substituting your values.

```
{
  "subscriptionId": "yoursubscriptionId",
  "tenantId":       "yourtenantId",
  "clientId":       "yourclientId",
  "clientSecret":   "yoursecret"
}
```

Usage

Create the base Secret via the CLI substituting a path of your choosing.

```
dsv secret create --path azure/base/api-account --data '@secret_base.json' --attributes '
{"type": "azure"}' --desc "azure base credential"
```

Dynamic Secrets

In DSV you can create dynamic secrets from either an existing service principal or create a temporary service principal.

Note: Temporary vs Existing Service Principals: Azure does not use these terms, but DSV can either use a service principal that you have already setup (existing) or DSV can create a service principal on-the-fly (temporary) through Azure's role-based access control (RBAC).

If possible, a temporary service principal is preferred. Temporary service principals are independent from other service principals and provide fine grained access and auditing. However, creating temporary service principals can take up to 2 minutes before fully provisioned on Azure.

Use of an existing service principal is required in some cases when Azure services are not accessible through Azure RBAC. In these cases, an existing service principal can be set up with the necessary access and DSV can create a new client secret for this service principal each time the dynamic secret is read. One issue with this might be that Azure limits the number of passwords for a given Application object, but this can be managed by reducing the secret TTL. Also keep in-mind that Azure does not log actions related to each secret, so auditing is not a clean as with temporary service principals.

Dynamic Secret for an Existing Service Principal

Create a dynamic secret that points to the base secret via its attributes.

Note: The dynamic secret does not have any data stored in it because data is only populated when you read the secret.

Attribute	Description
roleName	Optional- Azure role name to be assigned to the existing service principal. Does not change existing principal's role.
appId	Required - Application (client) ID for an existing service principal
appObjectId	Required - Application Object ID for an existing service principal
ttl	Optional - Time to live in seconds of the generated token. If none is specified it will default to 900.

Usage

1. Create an attributes JSON file named `secret_attributes.json` substituting your values.

```
{
    "linkConfig": {        "linkType": "dynamic",        "linkedSecret":
"azure:base:api-account"    },    "roleName":       "Contributor",    "appId":
"f81b3c6d-2ce9-47d4-ad2d-fef8390792a2",    "appObjectId" : "5fe218ee-cb58-4089-ac9f-
b1b68971ad73",    "ttl": 900}
```

2. Create the dynamic secret via the CLI substituting the path of your choosing.

```
dsv secret create --path azure/dynamic/api-account --attributes '@secret_
attributes.json' --desc "azure dynamic credential"
```

3. Now anytime you read the dynamic secret, the data is populated with the temporary Azure access credentials. The input

```
dsv secret read --path azure/dynamic/api-account
```

This returns the result:


```
{
    "id": "yourId",
    "path": "dynamic:azure:sp-static",
    "attributes": {
        "clientId": "yourpaddId",
        "appObjectId": "yourappObjectId",
        "linkConfig": {
            "linkType": "dynamic",
            "linkedSecret": "azure:base:api-account"
        },
        "roleName": "Contributor",
        "ttl": 900
    },
    "data": {
        "appObjectId": "yourappObjectId",
        "clientId": "yourclientId",
        "clientsecret": "yoursecret",
        "role": "Contributor",
        "subscriptionId": "yoursubscriptionId",
        "tenantId": "yourtenantId",
        "ttl": 900
    },
    "created": "2020-02-24T16:42:34Z",
    "lastModified": "2020-03-04T19:21:04Z",
    "version": "13"
}
```

Dynamic Secret for a Temporary Service Principal

Note: Creating service principal and assigning role in same request takes tens of seconds (over a minute has been seen), The command has been broken down into two separate calls. In the first call the service principal will be returned along with the task id that fired in the background for role assignment. You will need to wait to use that temporary service principal or check via the Azure portal or via the DSV API (provided below)

Attribute	Description
roleName	Optional - If no `roleID` is assigned, DSV will try to look-up the built-in Azure role by this name.
roleId	Optional - Azure role id to be assigned to the temporary service principal. If not defined, then DSV will attempt to look up the Azure built-in role by `roleName`. However, role ID takes precedence. Either `roleName` or `roleID` required.
scope	Required - Azure resource group to be assigned to the temporary service principal
ttl	Optional - Time to live in seconds of the generated token. If none is specified it will default to 900.

Note: Azure built-in role names and IDs can be found here

1. Create an attributes JSON file named `secret_attributes.json` substituting your values.

```
{
    "linkConfig": {
        "linkType": "dynamic",
        "linkedSecret": "azure:base:api-account"
    },
    "roleName":       "Contributor",
    "roleId":         "/subscriptions/<Azure Subscription
ID>/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-
20f7382dd24c",
    "scope":          "/subscriptions/<Azure Subscription ID>/resourceGroups/<resource
group name>",
    "ttl": 36000
}
```

2. Create a new dynamic secret via the CLI substituting the path of your choosing.

```
dsv secret create --path /azure/dynamic/api-account --attributes '@secret_attributes.json'
--desc "azure dynamic credential"
```

3. Now anytime you read the dynamic secret, the data is populated with the temporary Azure access credentials.

```
{
```

```
    "id": "yourId",
    "path": "dynamic:azure:ac-api",
    "attributes": {
        "linkConfig": {
            "linkType": "dynamic",
            "linkedSecret": "azure:base:api-account"
        },
        "roleId": "/subscriptions/6ca2adeb-7b44-4c7f-93fc-
2d5b9729a8c1/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-
20f7382dd24c",
        "roleName": "Contributor",
        "scope": "/subscriptions/6ca2adeb-7b44-4c7f-93fc-2d5b9729a8c1/resourceGroups/dsv-
resource-group",
        "ttl": 36000
    },
    "description": "azure root credential",
    "data": {
        "appObjectId": "e463477c-7d90-4743-92f2-c7f44ede8ec9",
        "clientId": "945d25cb-7697-4648-b574-e8a660154269",
        "clientSecret": "yoursecret",
        "roleName": "Contributor",
        "roleId": "youroleId",
        "roleAssignmentStatus": "created",
        "roleAssignmentTaskId": "task_3da0a37c-0a1c-4ebd-8829-dbe7b988b36f",
        "servicePrincipleId": "1782611c-99c2-418b-b672-783e3cf8bd14",
        "subscriptionId": "6ca2adeb-7b44-4c7f-93fc-2d5b9729a8c1",
        "tenantId": "11f54b31-ffb9-42b5-8fda-76c734a7796c",
        "ttl": 36000
    },
    "created": "2020-02-12T20:57:44Z",
    "lastModified": "2020-03-04T19:27:45Z",
    "version": "12"
}
```


4. It takes some time for the temporary service principal to be created, so you can check using the Azure portal for the new service principal or use the DSV API.

   Use the `roleAssignmentTaskId` from above response.

method	path
GET	/v1/task/status/{roleAssignmentTaskId}

Sample Response


```
{
  "taskName": "azure_role_assignment",
  "state": "SUCCESS",
  "results": null,
```

```
  "error": "",
  "createdAt": "2020-03-04T19:28:07.420285103Z"
}
```


Azure Service Principal

This is a step-by-step guide to creating an Azure service principal with the privileges necessary to enable Azure credential generation.

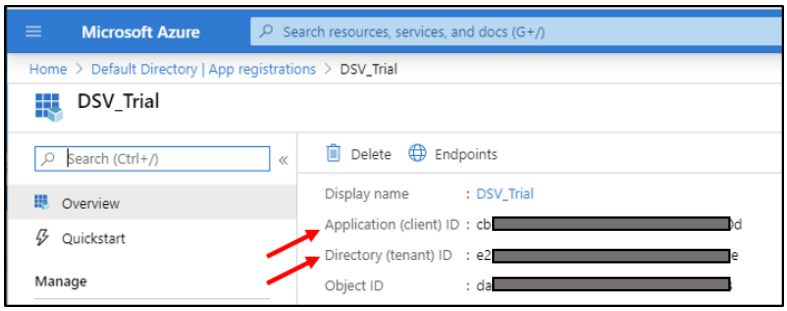
An Azure service principal is an identity created for use with applications, hosted services, and automated tools to access Azure resources.

These are the links to azure documentation on service principal:

- Service Principal
- Create Service Principal

Creating a Service Principal for the DSV Base Secret

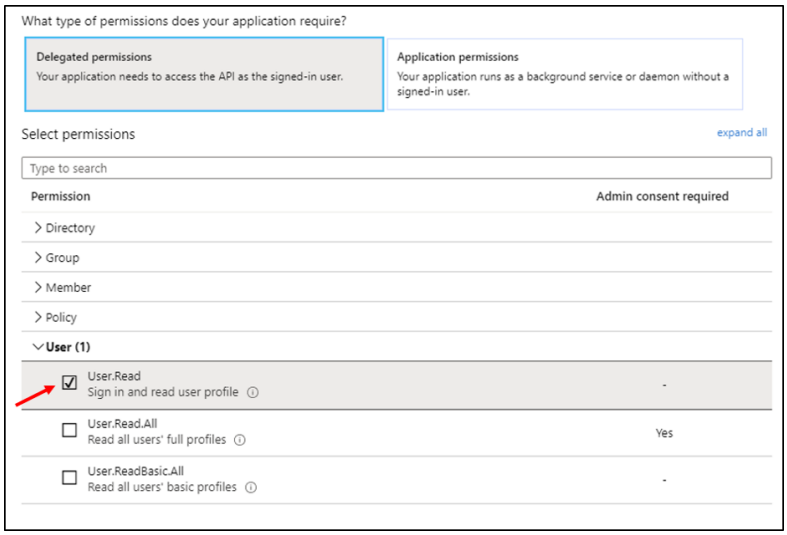
1. Go to the Microsoft Azure portal and login.
2. Go to **Azure Active Directory**.
3. Click **App registrations** then **New registration**. Enter an application name and then click **Register**.
4. Take note of the **Application (client) ID** and **Directory (tenant) ID**. They are the DSV Base secret `clientId` and `tenantId` parameters respectively.



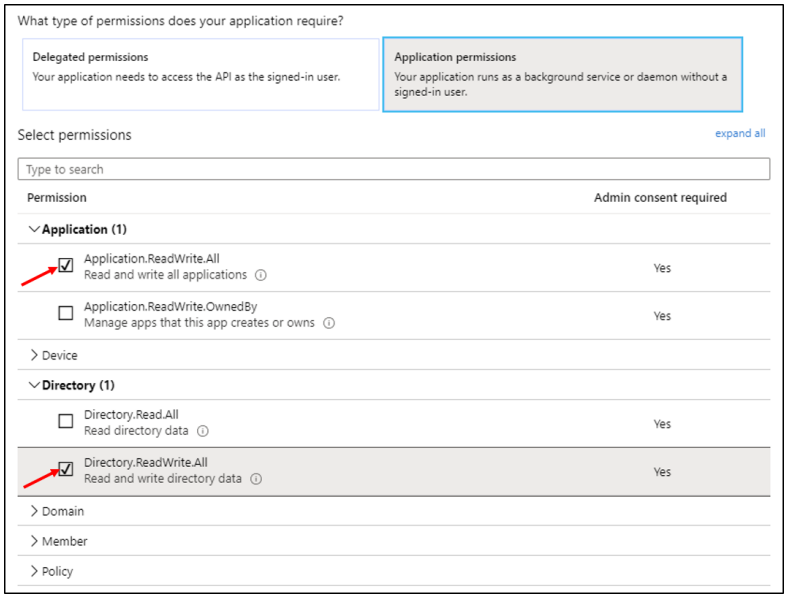
5. Select **Certifications & secrets** then **New client secret**. Enter a description and when it should expire. Click **Add**.
6. Take note of the newly generated secret which will be the `clientSecret` parameter in the DSV Base Secret.

7. Select **API permissions** and then **Add a permission**.

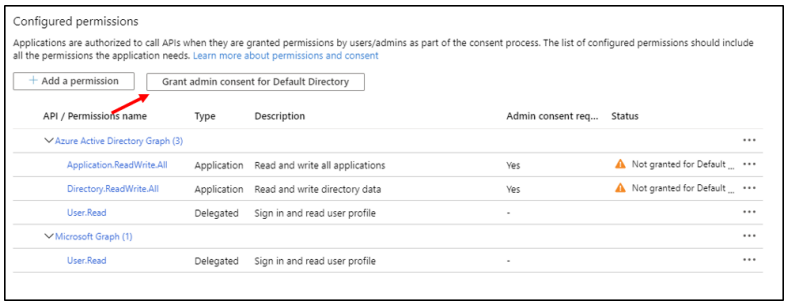
8. Under Supported Legacy APIs, select **Azure Active Directory Graph**.
9. Select **Delegated permissions**, expand the **User** accordion, and then check the **User.Read** box.



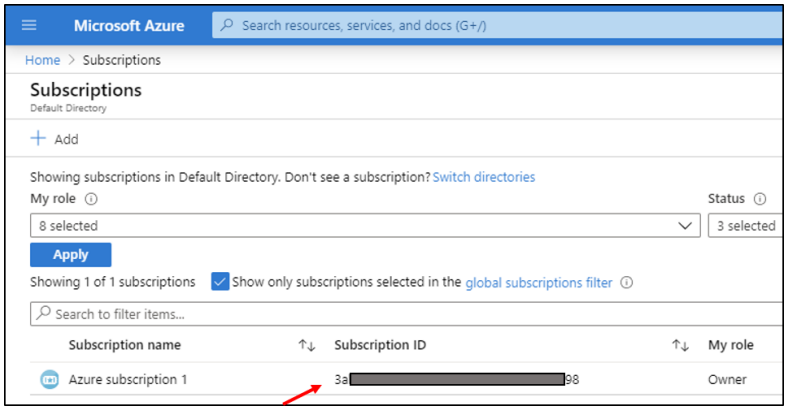
10. Select **Application permissions** and expand the **Application** and **Directory** accordions. Check the **Application.ReadWrite.All** and **Directory.ReadWrite.All** boxes.



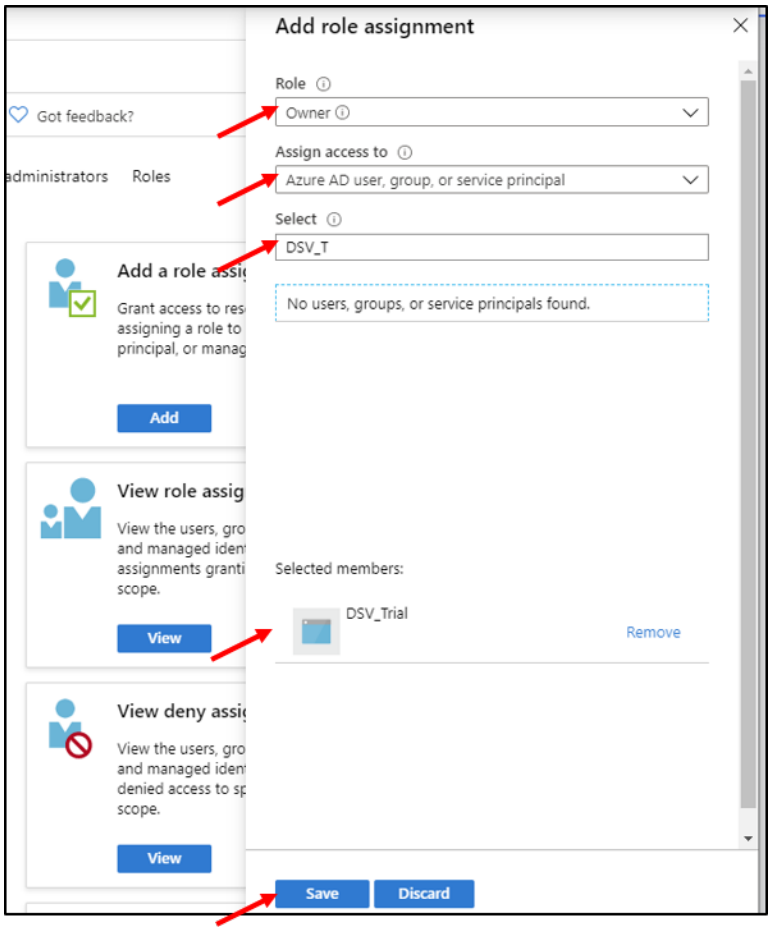
11. Select **Add permissions** at the bottom of the page. This takes you back to the API Permissions page. Notice that the Application permissions have warnings that those permissions are not yet granted.
12. Click **Grant admin consent for Default Directory** and then **Yes**. This step can be easy to miss.



13. Navigate to **Home** | **Subscriptions** and take note of the **Subscription ID** that you will be using. This is the `subscriptionId` in the DSV base secret.



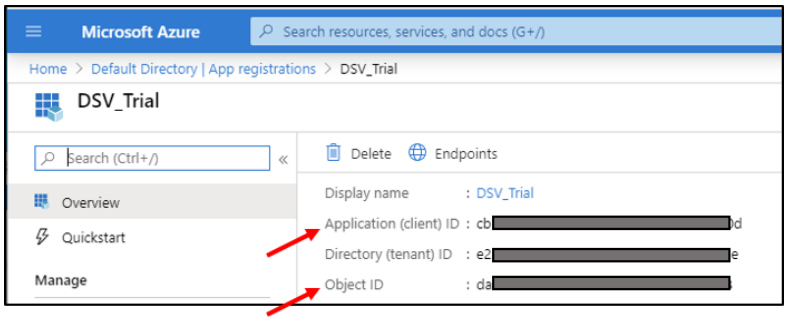
14. Click into the **Subscription ID** then **Access control (IAM)** then **Add** in the **Add role assignment** box on the right.
15. Select **Owner** in the **Role** drop-down.
16. Select **Azure AD user, group, or service principal** in the **Assign access to** drop-down.
17. In the **Select** field, enter the application name or **Application (client) ID** saved previously and select it so that it shows up under **Selected Members** below.
18. Click **Save**.



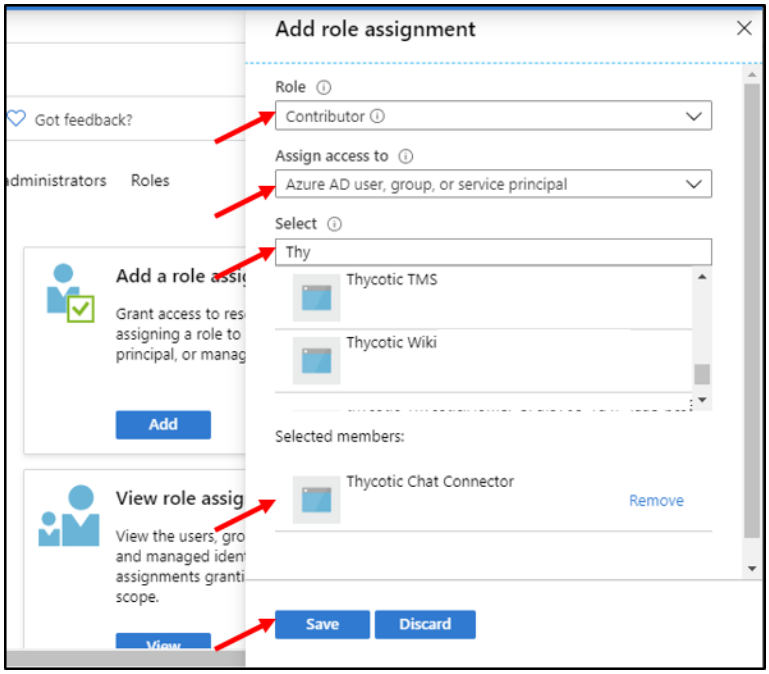
Creating a Service Principal for a DSV Dynamic Secret

In the Azure Dynamic Secrets section, we discuss DSV using an existing service principal vs DSV creating a temporary service principal. This is guidance on creating an existing service principal in the Azure portal. In the case of the temporary service principal, no guidance in Azure is needed because DSV creates them.

1. Go to the Microsoft Azure portal and login.
2. Go to **Azure Active Directory**.
3. Click **App registrations** then **New registration**. Enter an application name and then click **Register**.
4. Take note of the **Application (client) ID** and **Object ID**. They are the DSV Dynamic Secret `appId` and `appObjectId` parameters respectively.



5. Navigate to **Home > Subscriptions**.
6. Click into the **Subscription ID** that you are using and then **Access control (IAM)** then **Add** in the **Add role assignment** box on the right.
7. Select **Role** drop-down, select the role you wish to provide. In this example, we will use **Contributor**.
8. Select **Azure AD user, group, or service principal** in the **Assign access to** drop-down.
9. In the **Select** field, enter the application name or **Application (client) ID** saved previously and select it so that it shows up under **Selected Members** below.
10. Click **Save**.



Microsoft Graph Dynamic Secrets

To create dynamic secrets for Azure Microsoft Graph:

- Create an Azure Service Principal for Microsoft Graph.
- Create a base secret.

Usage

- Create the dynamic secret.

Create the Base Secret

The base Secret holds the credentials required for DSV to perform API calls to Azure to query roles and create/delete service principals.

Attribute	Description
subscriptionId	Required - The subscription ID holding the resources you wish to access using Azure Active Directory.
tenantId	Required - The tenant ID for Azure Active Directory. Azure lists it in places as "Directory (tenant) ID"
clientId	Required - The OAuth2 client ID to connect to Azure. Azure lists it in places as "Application (client) ID"
clientSecret	Required - The OAuth2 client secret to connect to Azure.
environment	Optional - The Azure environment. If not specified, DSV will use Azure Public Cloud.

Create a file named `secret_base.json` substituting your values:

```
{
  "subscriptionId": "yoursubscriptionId",
  "tenantId":       "yourtenantId",
  "clientId":       "yourclientId",
  "clientSecret":   "yoursecret"
}
```

Create the base Secret via the CLI substituting a path of your choosing:

```
    dsv secret create --path azure/base/api-account --data '@secret_base.json' --
attributes '{"type": "azure"}' --desc "azure base credential"
```

Dynamic Secret for a Temporary Service Principal

Attribute	Description
appRoleId	Required - The id of the appRole (defined on the resource service principal) to assign to the client service principal.
ResourceID	Required - The id of the resource servicePrincipal (the API) which has defined the app role (the application permission).

Usage

Attribute	Description
ttl	Optional - Time to live in seconds of the generated token. If none is specified it will default to 900.

1. Create an attributes json file named `secret_attributes.json` substituting your base secret path, resourceID, and AppRoleID.

```
{
        "linkConfig": {
        "linkType": "dynamic",
        "linkedSecret": "azure:base:msgraph"
    },
    "ttl": 360,
    "resourceId": "resourceID",
    "appRoleId": "appRoleId",
    "msApiType": "msgraph"
}
```

2. Create a new Dynamic Secret via the CLI substituting the path of your choosing.

```
dsv secret create --path /azure/dynamic/api-graph --attributes '@secret_attributes.json' -
-desc "azure dynamic credential"
```

3. Now anytime you read the dynamic Secret, the data is populated with the temporary azure access credentials.


```
{
    "id": "8247cc11-7465-49de-9d49-959f1d2e7e39",
    "path": "dynamic:azure:ac-graph",
    "attributes": {
        "appRoleId": "c6d6abd5-4021-4d46-8f18-xxxxxxxxxxx",
        "linkConfig": {
            "linkType": "dynamic",
            "linkedSecret": "azure:base:api-graph"
        },
        "msApiType": "msgraph",
        "resourceId": "8c828069-ab9c-4a3b-b30e-xxxxxxxxxxxx",
        "ttl": 360
    },
    "description": "azure root credential",
    "data": {
        "clientId": "xxxxxxxx-eaa5-4c82-a177-f526742f8881",
        "clientSecret": "secret key",
        "displayName": "dsv-7c89bba6-d61e-4de8-9c10-a4735f8eebff",
        "servicePrincipalId": "xxxxxxx-2e29-4bad-a908-a5cf0c7eaebb",
        "subscriptionId": "your subscriptionId",
        "tenantId": "your tenantId"
    },
```

```
    "created": "2020-12-13T03:31:07Z",
    "lastModified": "2021-01-12T19:50:24Z",
    "createdBy": "users:user1",
    "lastModifiedBy": "users:user1",
    "version": "15"
}
```

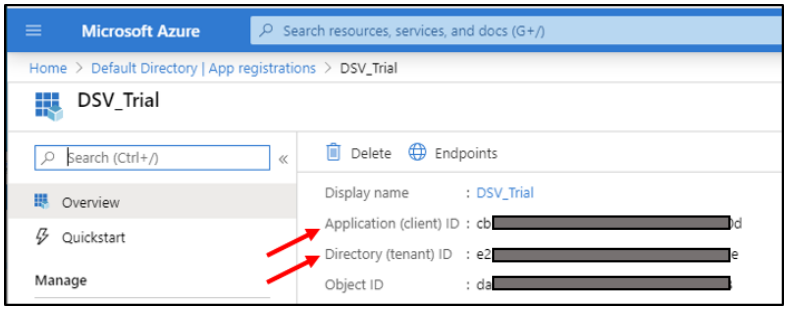

Azure Service Principal

This is a step-by-step guide to creating an Azure service principal with the privileges necessary to enable Azure Microsoft Graph credential generation.

An Azure **service principal** is an identity created for use with applications, hosted services, and automated tools to access Azure resources.

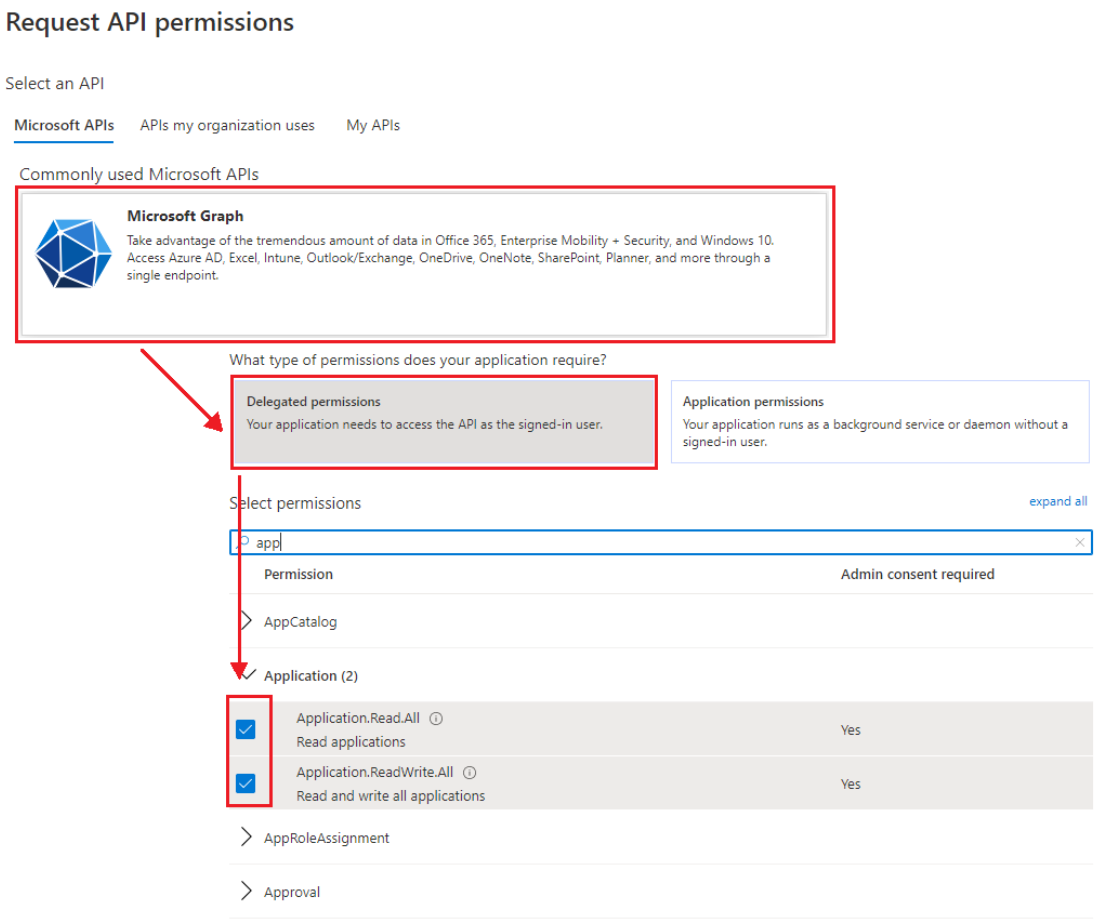
Creating a Service Principal for the DSV Base Secret

1. Login to the Microsoft Azure portal.
2. Go to **Azure Active Directory**.
3. Click **App registrations**, then **New registration**. Enter an application name and then click **Register**.
4. Take note of the **Application (client) ID** and **Directory (tenant) ID**. They are the DSV Base secret `clientId` and `tenantId` parameters, respectively.

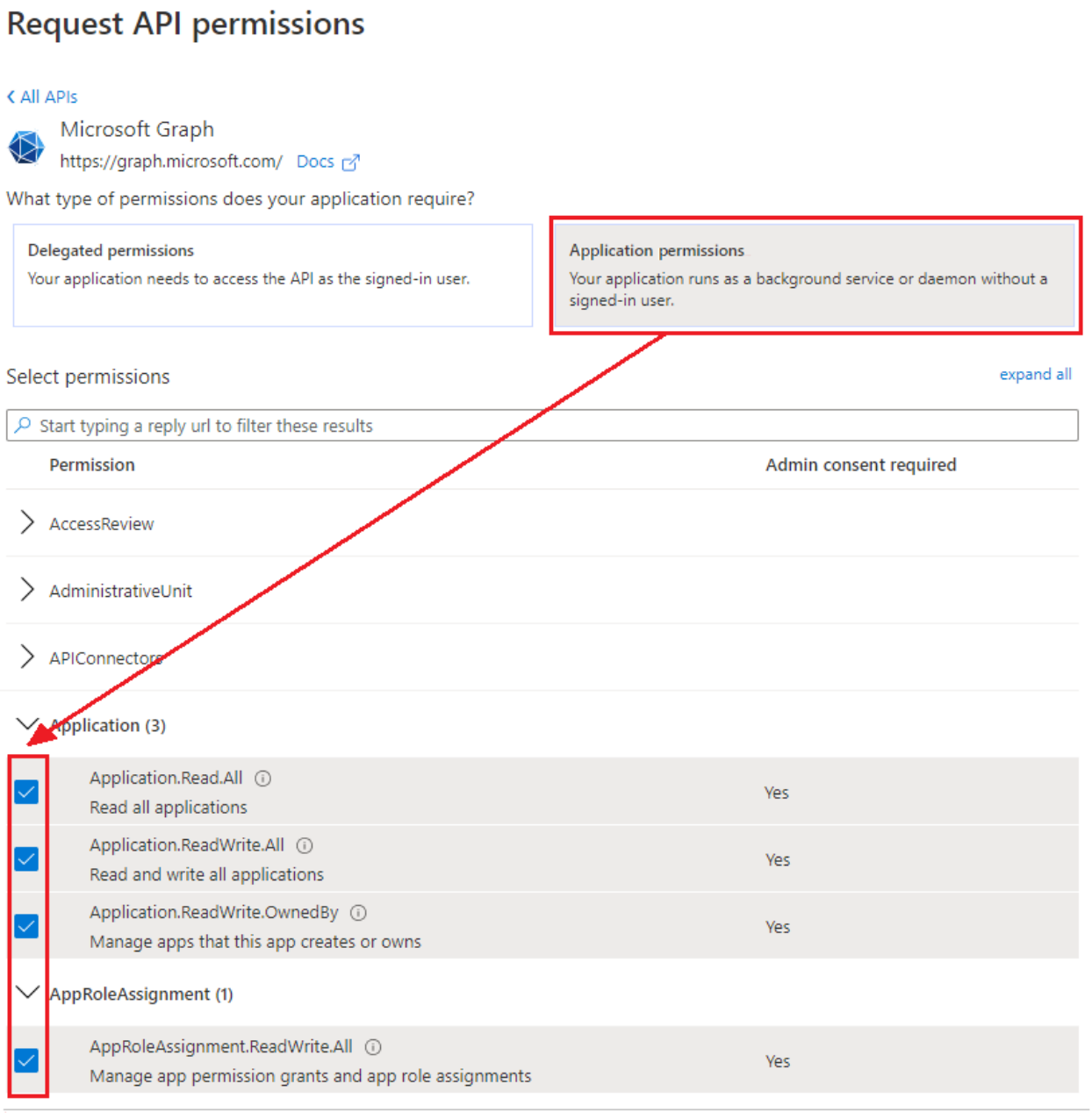


5. Select **Certifications & secrets** then **New client secret**. Enter a description and expiration date. Click **Add**.
6. Take note of the newly generated secret which will be the `clientSecret` parameter in the DSV Base Secret.

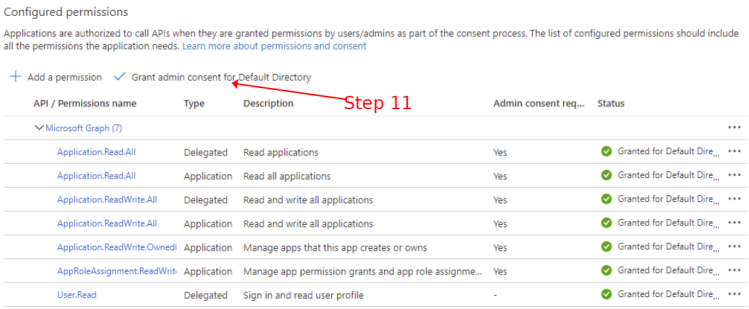
7. Select **API permissions** and then **Add a permission**.
8. Under Microsoft Graph APIs, first select **Delegated** permissions on the left. Expand the **Application** drop-down and check the **Application.Read.All** and **Application.ReadWrite.All** boxes.



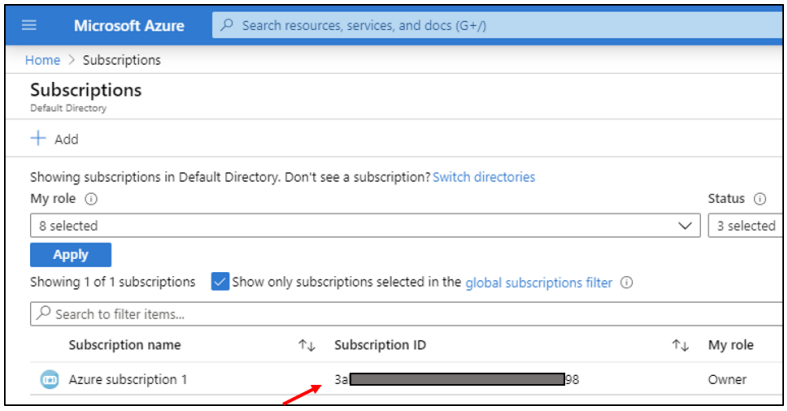
9. Now, select **Application** permissions on the right. Expand both the **Application** and **AppRoleAssignment** drop-down, and then check the **Application.Read.All**, **Application.ReadWrite.All**, and **Application.ReadWrite.OwnedBy** boxes under Application and the **AppRoleAssignment.ReadWrite.All** box under AppRoleAssignment.



10. Select **Add permissions** at the bottom of the page. This takes you back to the API Permissions page. Notice that the Application permissions have warnings that those permissions are not yet granted.
11. Click **Grant admin consent** and then **Yes** (You will need administrative privileges to complete this step).
12. The completed API permissions should look like this:



13. Navigate to **Home > Subscriptions** and take note of the **Subscription ID** that you will be using. This is the `subscriptionId` in the DSV Base Secret.

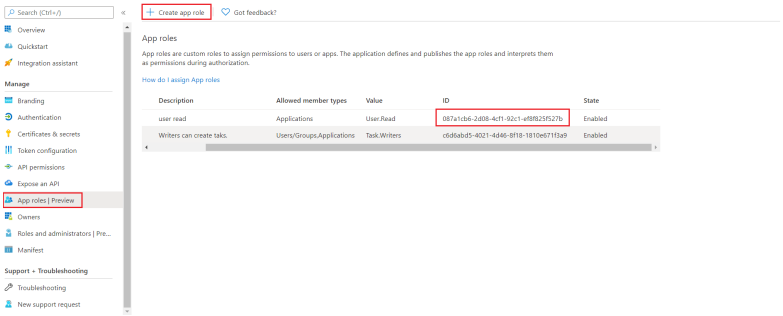


Add appRole in Root Application or Any Application

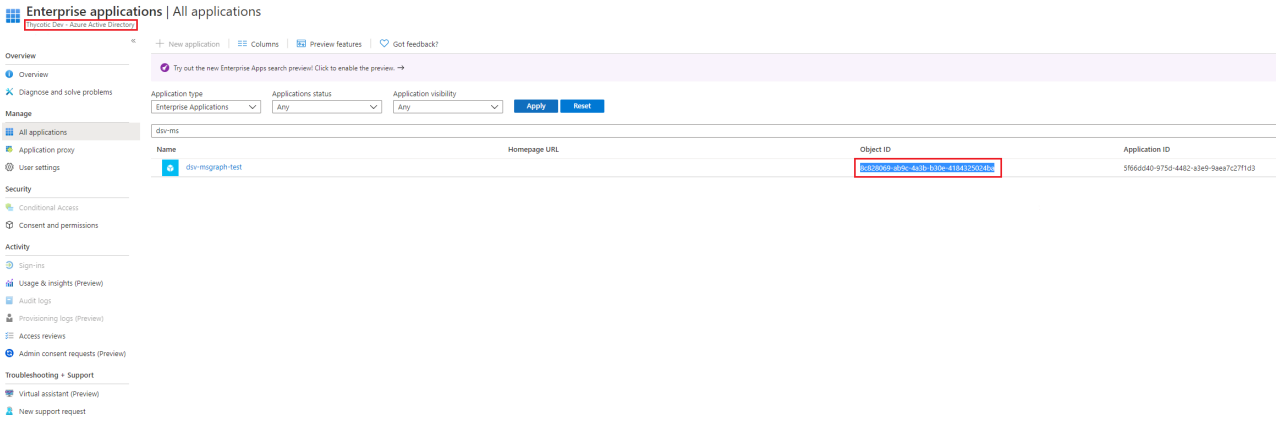
In the Azure Dynamic Secrets section, we discuss DSV using an "existing service principal" vs DSV creating a "temporary service principal." This is guidance on creating an **existing service principal** in the Azure portal. In the case of the **temporary service principal**, no guidance in Azure is necessary because DSV will create them.

Note: Any existing or new application can be used in place of the base service principal.

1. Go to the Microsoft Azure portal and login.
2. Go to **Azure Active Directory**.
3. Click **App registrations**.
4. Click on the **new application** that you created in the Service Principal Guide, or on a previously existing application.
5. Click **Create App Role** to create a new one, or select an existing appRole.
6. Take note of the **ID**. That is the DSV Dynamic Secret `appRoleId` parameter.



7. Navigate to **Active Directory > Enterprise applications**.
8. Select the **application name** that you configured in the above steps.
9. Take note of the **Object ID**. This is the DSV Dynamic Secret `resourceId` parameter.



GCP Dynamic Secrets

There are two ways to generate dynamic GCP secrets:

- Token Generation
- Service Account Key

Token generation creates an access token that can be used as the bearer token in the GCP API. Service account key generation creates a new key on a service account in GCP and then deletes the key after the specified time to live is up.

Setup

Create a GCP Service Account

For setting up GCP token or key based dynamic secrets, you will first need a service account in GCP.

- Go to **Service Accounts** under **IAM & Admin** in the GCP console.
- Click **Create Service Account** and grant it access to a project.

- Generate a key for the service account and save it.
- Under **IAM** Assign the `Service Account Key Admin` and `Service Account Token Creator` roles to the new service account. Also give it `Storage Admin` which will be used for testing the dynamic secrets.

Create the Base Secret

Next create a Secret in DSV with the AWS IAM user access key, secret key, and region.

Create a file named `secret_root.json` substituting your values from the service key file.

```
{
  "projectId": "test-project-1234",
  "type": "service_account",
  "privateKeyId": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "privateKey": "-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----\n",
  "clientEmail": "dsv-test@test-project-1234.iam.gserviceaccount.com",
  "clientId": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "tokenUri": "https://oauth2.googleapis.com/token"
}
```

Create the Secret via the CLI at a path of your choosing.

```
dsv secret create --path gcp/base/svc-account --data @secret_root.json --attributes '
{"type": "gcp"}'
```

OAuth Access Token

Attribute	Description
scopes	Array of GCP OAuth 2.0 scopes for the dynamic token
providerType	`token`

Now you need to create a dynamic secret, which points to the base Secret via its attributes. The dynamic secret doesn't have any data stored in it because data is only populated when you read the secret.

Create an attributes JSON file named `secret_attributes.json` substituting your values.

```
{
    "linkConfig": {
        "linkType": "dynamic",
        "linkedSecret": "gcp:base:svc-account"
    },
    "providerType": "token",
    "scopes": [
            "https://www.googleapis.com/auth/devstorage.full_control"
        ]
```

```
}
```

Create a new dynamic secret.

```
dsv secret create --path dynamic/gcp/token --attributes @secret_attributes.json
```

Now, anytime you read the dynamic secret, the data is populated with the temporary access token that is valid for one hour.

```
dsv secret read --path dynamic/gcp/token
```

This returns a result like:

```
{
    "id": "ba2f1fc7-c16f-4062-a216-3116d1a42545",
    "path": "dynamic:gcp:token",
    "attributes": {
        "linkConfig": {
            "linkType": "dynamic",
            "linkedSecret": "gcp:base:svc-account"
        },
        "providerType": "token",
        "scopes": [
            "https://www.googleapis.com/auth/devstorage.full_control"
        ]
    },
    "description": "gcp dynamic token secret",
    "data": {
        "access_token": "youraccesstoken",
        "expiry": "2020-04-26T22:04:32.3897188Z",
        "ttl": 3600
    }
}
```

You can validate the credentials are able to read storage buckets by making an API request with the access token in the `Authorization` header to the storage API for your project, substituing your values.

```
curl -H 'Authorization: Bearer {access token}'
https://storage.googleapis.com/storage/v1/b?project={project id}
```

Service Account Key

In this example, rather than generating an OAuth token we will generate a new key in JSON format for the service account. This creates a new key in GCP that can be used to authenitcate with the gcloud CLI or other SDKs. Once

the ttl for the dynamic secret expires, the key will be removed.

Service accounts in GCP are limited to 10 keys per account. If you exceed this, you will get a 400 error reading the dynamic secret with a message of `unable to create new service account key googleapi: Error 429: Maximum number of keys on account reached., rateLimitExceeded`.

To help avoid this, ensure that you keep ttls relatively low for service account keys to ensure they get cleaned up. You can also create multiple service accounts with the same permissions in GCP and then create a base secret for each one to help spread the number of keys across service accounts.

Create the Base Secret

For this example, we will reuse the base secret from above. If you haven't done this already, then follow those directions to create the base secret now.

Create the Dynamic Secret

Attribute	Description
providerType	`serviceKey`
ttl	required time to live in seconds of the generated token.

Create or update the attributes JSON file named `secret_attributes.json` changing the provider type to `serviceKey` and replacing the

```
{
    "linkConfig": {
        "linkType": "dynamic",
            "linkedSecret": "gcp:base:svc-account"
    },
    "providerType": "serviceKey",
    "ttl": 3600
}
```

Now create the dynamic secret in the CLI using the JSON above.

```
dsv secret create --path dynamic/gcp/secret-svc-key --attributes @secret_attributes.json
```

Now, anytime you read the dynamic secret, the data is populated with the GCP service key.

```
dsv secret read --path dynamic/gcp/secret-svc-key
```

This returns a result like:

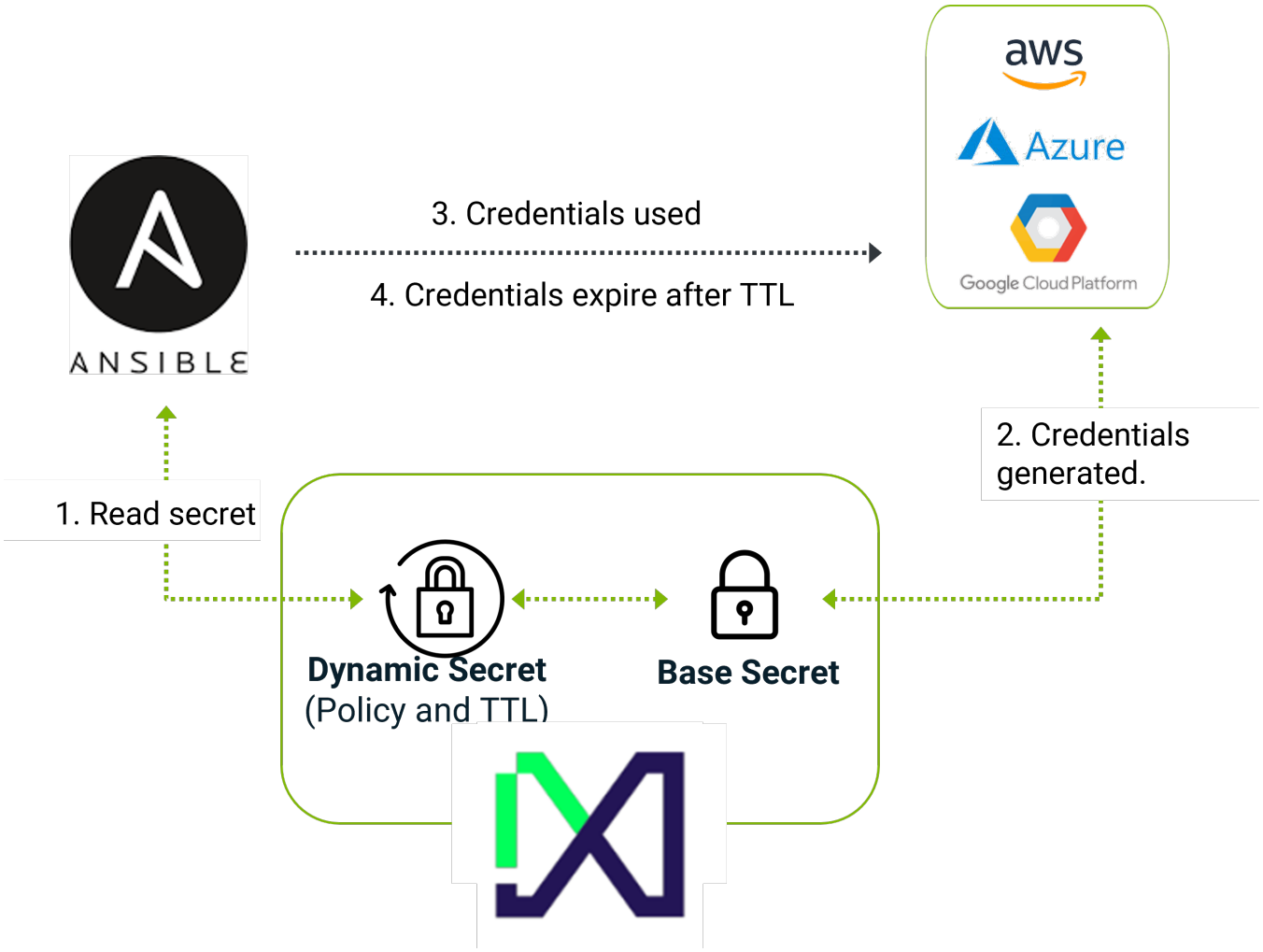
```
{
    "linkConfig": {
        "linkType": "dynamic",
        "linkedSecret": "gcp:base:svc-account"
    },
    "providerType": "serviceKey",
    "ttl": 3600
  },
  "data": {
    "keyAlgorithm": "KEY_ALG_RSA_2048",
    "keyOrigin": "GOOGLE_PROVIDED",
    "name": "projects/test-proj-1234/serviceAccounts/dsv-test@test-prog-
1234.iam.gserviceaccount.com/keys/0e4c690b713bfe0ed517ed56cba4814afd35a8ad",
    "privateKeyData":
    {
      "client_id": "xxxxxxxxxxxxxxxxx",
      "auth_uri": "https://accounts.google.com/o/oauth2/auth",
      "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/dsv-
test%40test-proj-1234.iam.gserviceaccount.com",
      "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
      "client_email": "dsv-test@test-project-1234.iam.gserviceaccount.com",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvQIBADAN...iV7quFF35ILBG+w=\n-----
END PRIVATE KEY-----\n",
      "private_key_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
      "token_uri": "https://oauth2.googleapis.com/token",
      "type": "service_account",
      "project_id": "test-proj-1234"
    },
    "ttl": 3600
  },
  "description": "",
  "id": "34fb64d7-18da-453d-9487-3d1c082ba372",
  "version": "0"
}
```


Copy the inner JSON of `privateKeyData` into a file and name it *svc-account.json*. Then, using the gcloud CLI, run `gcloud auth activate-service-account --key-file svc-account.json` to test if the generated key is valid. If so, you will get a reply similar to **Activated service account credentials for: [service account email]**.

After the ttl expires, you can check the keys on the service account and they will be removed. Note that there may be some delay between when the ttl expires and when the key is removed from the service account.

Database Dynamic Secrets

Both Database Dynamic Secrets and IaaS Dynamic Secrets provide temporary credentials for very specific uses. The possible damage done by leaked credentials is severely limited to due to granular policies and short time-to-live. However, IaaS platforms provide mechanisms for temporary credentials with fine-grained policies, and most databases do not. Therefore, DSV provides a way to provide temporary credentials by creating and deleting users in a just-in-time manner.



DSV Engine Required

Database Dynamic Secrets require the deployment of the DSV Engine. See the instructions at DSV Engine.

Microsoft SQL Dynamic Secrets

Once you have installed the DSV Engine, you can use DSV to create Dynamic Secrets. DSV currently supports **contained** MSSQL databases. DSV does not currently support **traditional** MSSQL databases.

Dynamic Secret Setup

1. **Create a Base Secret**

   In the CLI, create a base secret containing the credentials of the MSSQL account that will be responsible for creating new accounts on a given server. You must mark the secret as a MSSQL root secret by including `type` with a value of `mssql`. All fields in the `data` object are required.

   Note: Port is an integer and does not require quotations.

Example Base Secret:

```
{
    "attributes": {    "type": "mssql"  },  "data": {    "database":
"TestContainedDB",    "password": "yourpassword",    "port": 1433,    "server":
"localhost",    "username": "yourusername"  }}
```

2. **Create a new dynamic secret.** The dynamic secret will be linked to the root secret. Use the following format:

Dynamic Secret Example

```
{
 "attributes": {
    "grantPermissions": {
      "what": "SELECT",
      "where": "exampletable"
    },
    "linkConfig": {
      "linkType": "dynamic",
      "linkedSecret": "mssql:base2"
    },
    "pool": "pool1",
    "ttl": 900,
    "userPrefix": "test"
  }
}
```

Dynamic Secret Guide

1. **`grantPermissions`**: Specifies the permissions assigned by MSSQL to the new user account.
   - `what`: Defines the database access permissions the user will have in MSSQL. Permissions may include `CONNECT`, `CREATE`, `SELECT`, or other SQL statements.
   - `where`: Defines the location within the database for permissions to apply.
2. **`linkType`** is always **`dynamic`** for dynamic secrets.
3. **`linkedSecret`** should be the path of the root secret.
4. **`pool`**: Designates the Engine pool that DSV will use to generate dynamic secrets.
5. **`ttl`**: Specifies the number of seconds for which the new account will exist before the engine automatically deletes it.

   Note: `ttl` must be set at or **above 900**.

6. **`userPrefix`** An optional key whose value is a string prepended to all MSSQL account usernames created from the dynamic secret.
7. **`data`**: This field remains blank for dynamic secrets.

Sending a MSSQL task to an engine

Read the MSSQL dynamic secret. A randomly chosen engine in the engine pool should receive the task and perform it. The engine attempts to create a MSSQL account and reports back success or failure. On success, the user also receives the new working credentials. As long as there is at least one running engine in a given pool, an engine will receive a MSSQL account revocation task and delete the account once its TTL expires.

Third Party Reference

For contained server configuration details, refer to MSSQL Database Documentation

MySQL Dynamic Secrets

Once you have installed the DSV Engine, you can use DSV to create dynamic secrets.

Base Secret

Base secret data defines how to establish a connection with a MySQL server. All values are required and will be used to build a connection string in a URL format. A type must be set in attributes of a base secret. For MySQL, the type field in attributes should always be `mysql`.

Create a file named `mysql_base.json`, substituting your values:

```
{
  "host": "your.host",
  "port": 3306,
  "username": "mysqlusr",
  "password": "myp@ssword"
}
```

Create a secret using the CLI at a path of your choosing:

```
dsv secret create \
    --path db:mysql:root \
    --data @mysql_base.json \
    --attributes '{"type": "mysql"}'
```

Dynamic Secret

A dynamic secret will be linked to the base secret. One base secret can have many dynamic secrets linked to it.

Create a file named `mysql_dynamic1.json`, substituting your values:

```
{
  "linkConfig": {
    "linkType": "dynamic",
    "linkedSecret": "db:mysql:root"
  },
```

```
  "grantPermissions": {
    "what": "SELECT",
    "where": "*.*"
  },
  "pool": "pool1",
  "ttl": 1000,
  "userPrefix": "usr"
}
```

Create a dynamic secret using the CLI at a path of your choosing:

```
dsv secret create --path db:mysql:dynamic1 --attributes @mysql_dynamic1.json
```

Note: when creating a dynamic secret the `data` field should be empty.

Attributes description:

1. **`linkConfig`**: denotes that it is a dynamic secret with a link to a base secret:
   - **`linkType`**: should always be **`dynamic`**
   - **`linkedSecret`**: sets a path to base secret
2. **`grantPermissions`**: defines access privileges
   - **`what`**: a specific privilege type, e.g. `ALL`, `INSERT`, `UPDATE`, `DELETE`
   - **`where`**: a privilege level, e.g. `*.*`, `mydb.*`, `mydb.mytbl`
3. **`pool`**: a pool name to use
4. **`ttl`**: a number of seconds before the engine automatically deletes new credentials, **must be set at or above 900**
5. **`userPrefix`**: *an optional field* that defines a prefix for a new username

To create a new user, the CREATE USER command is used.

To assign privileges, the GRANT command is used.

```
GRANT <"what"> ON <"where"> TO <"username">;
```

Sending a MySQL Task to an Engine

Read the MySQL dynamic secret. A randomly chosen engine in the engine pool should receive the task and perform it. The engine attempts to create a MySQL account and reports back success or failure. On success, the user also receives the new working credentials. As long as there is at least one running engine in a given pool, an engine will receive a MySQL account revocation task and delete the account once its TTL expires.

List MySQL Base Secrets

To find all base secrets that are related to MySQL run:

```
dsv secret search --query "mysql" --search-field "attributes.type"
```

List Dynamic Secrets

To find all dynamic secrets that are linked to a specific base secret run:

```
dsv secret search --query "db:mysql:root" --search-links
```

Read Dynamic Secret Attributes

Using the `secret read` CLI command to read a dynamic secret will initiate a creation of a new credentials. To read a dynamic secret use the `secret describe` CLI command instead.

Example:

```
dsv secret describe db:postgresql:dynamic1
```

The `secret describe` does not return the secret data field, but for dynamic secrets it is always empty.

Third Party Reference

For server configuration details, refer to MySQL Database Documentation

Oracle Dynamic Secrets

Oracle Engine Requirements:

The Oracle database must have **Oracle Instant Client** installed before running the dsv-engine. DSV only supports the **linux-x64** binary distribution. For other platforms, use docker distribution.

Running the Oracle Engine

To run the DSV Engine with Oracle Instant Client

1. Install oracle client (https://www.oracle.com/database/technologies/instant-client/downloads.html)
2. Register the engine:

   ```
   dsv-engine-linux-x64-oracle register --engineName engine01 --secretsvaultcloud.com --
   tenant acme --userToken <your jwt>
   ```
3. Run the Engine:

   ```
   Engine run dsv-engine run
   ```

Docker Setup - PULL from ECR

To run the DSV Engine using Docker

1. Login to AWS ECR: **`aws ecr get-login --region us-east-1`**
2. Login to Docker.
3. Pull the Engine: **`docker pull 661058921700.dkr.ecr.us-east-1.amazonaws.com/dsv-engine-oracle:latest`**
4. Run the Engine:

```
run --env ENGINE_NAME=myengine --env DSV_POOL=pool1 --env DSV_TENANT=mal --env DSV_
URL=devbambe.com --env DSV_TOKEN=<jwt> 661058921700.dkr.ecr.us-east-1.amazonaws.com/dsv-
engine-oracle-dev:latest
```

Oracle Dynamic Secret Setup

To create a dynamic secret in Oracle, first create a base secret.

Create a Base Secret

In the CLI, create a base secret containing the credentials of the account that will be responsible for creating new accounts on a given server. You must mark the secret as an Oracle root secret by including **`type`** with a value of **`oracle`**. All fields in the **`data`** object are required.

Note: Port is an integer and does not require quotations.

Example Base Secret:

```
    {
        "data": {
            "password": "your password",
                "username": "your username",
                "host":     "host",
                "servicename": "servicename",
                "port":   1521},
        "description": "oracle root credential",
        "attributes": {
         "type": "oracle"
    }
}
```

Create a new dynamic secret.

The dynamic secret will be linked to the root secret. The **grantPermissions** field will change depending on the privileges the secret is granting.

Dynamic Secret Examples

System Privilege Dynamic Secret Example

```
{
    "description": "oracle system dynamic credential",
    "attributes": {
        "grantPermissions": {
        "what": "CONNECT",
        "where": "none",
        "type": "system"
        },
        "linkConfig": {
            "linkType": "dynamic",
            "linkedSecret": "oracle:base:awsroot"
        },
        "pool": "pool1",
        "ttl": 900
    },
    "data": {},
}
```

System Privilege Dynamic Secret Guide

1. **`grantPermissions`**: Specifies the permissions assigned by Oracle to the new user account.
   - `what`: Defines the database access permissions the user will have in Oracle. Permissions may include `CONNECT`, `CREATE`, `SELECT`, or other SQL statements.
   - `where`: Defines the location within the database for `object` permissions to apply. For `system` and `role` secrets, the field should be "none".
   - `type`: Defines the object permissions within Oracle. Use `system` to grant system privileges.
2. **`linkType`** is always **`dynamic`** for dynamic secrets.
3. **`linkedSecret`** should be the path of the root secret.
4. **`pool`**: Designates the Engine pool that DSV will use to generate dynamic secrets.
5. **`ttl`**: Specifies the number of seconds for which the new account will exist before the engine automatically deletes it.

   Note: `ttl` must be set at or **above 900**.

6. **`userPrefix`** An optional key whose value is a string prepended to all Oracle account usernames created from the dynamic secret.
7. **`data`**: This field remains blank for dynamic secrets.

Role Privilege Dynamic Secret Example

```
{
```

```
    "description": "oracle role dynamic credential",
    "attributes": {
        "grantPermissions":{
        "what" : "oraclerole",
        "where": "none",
        "type": "role"
       },
        "linkConfig": {
          "linkType": "dynamic",
          "linkedSecret": "oracle:base:awsroot"
       },
   "pool": "pool1",

   "ttl": 900
    }
}
```

Role Privilege Dynamic Secret Guide

1. **`grantPermissions`**: Specifies the permissions assigned by Oracle to the new user account.
   - `what`: Defines the Role access the user will have in Oracle. Set this as the predefined `role name`.
   - `where`: Defines the location within the database for `object` permissions to apply. For `system` and `role` secrets, the field should be "none".
   - `type`: Defines the object permissions within Oracle. Use `role` to grant role privileges.
2. **`linkType`** is always **`dynamic`** for dynamic secrets.
3. **`linkedSecret`** should be the path of the root secret.
4. **`pool`**: Designates the Engine pool that DSV will use to generate dynamic secrets.
5. **`ttl`**: Specifies the number of seconds for which the new account will exist before the engine automatically deletes it.

   Note: `ttl` must be set at or **above 900**.

6. **`userPrefix`** An optional key whose value is a string prepended to all Oracle account usernames created from the dynamic secret.
7. **`data`**: This field remains blank for dynamic secrets.

Object Privilege Dynamic Secret Example

```
{
    "description": "oracle object dynamic credential",
    "attributes": {
        "grantPermissions":{
        "what" : "SELECT",
        "where": "ADMIN.EMPLOYEE",
        "type": "object"
```

```
        },
         "linkConfig": {
           "linkType": "dynamic",
           "linkedSecret": "oracle:base:awsroot"
        },
    "pool": "pool1",
    "ttl": 900
     }
 }
```

Object Privilege Dynamic Secret Guide

1. **`grantPermissions`**: Specifies the permissions assigned by Oracle to the new user account.
   - `what`: Defines the database access permissions the user will have in Oracle. Permissions may include `CONNECT`, `CREATE`, `SELECT`, or other SQL statements.
   - `where`: Defines the object within Oracle for which the user will have privileges. The example secret will allow the user to select the "ADMIN.EMPLOYEE" object within Oracle.
   - `type`: Defines the object permissions within Oracle. Use `object` to grant object privileges.
2. **`linkType`** is always **`dynamic`** for dynamic secrets.
3. **`linkedSecret`** should be the path of the root secret.
4. **`pool`**: Designates the Engine pool that DSV will use to generate dynamic secrets.
5. **`ttl`**: Specifies the number of seconds for which the new account will exist before the engine automatically deletes it.

   Note: `ttl` must be set at or **above 900**.

6. **`userPrefix`** An optional key whose value is a string prepended to all Oracle account usernames created from the dynamic secret.
7. **`data`**: This field remains blank for dynamic secrets.

Sending an Oracle Task to Engine

Read the Oracle dynamic secret. A randomly chosen engine in the engine pool should receive the task and perform it. The engine attempts to create a Oracle account and reports back success or failure. On success, the user also receives the new working credentials. As long as there is at least one running engine in a given pool, an engine will receive a Oracle account revocation task and delete the account once its TTL expires.

Third Party Reference

For server configuration details, refer to Oracle Database Documentation

PostgreSQL Dynamic Secrets

Once you have installed the DSV Engine, you can use DSV to create dynamic secrets.

Base Secret

Base secret data defines how to establish a connection with a PostgreSQL server. All values are required and will be used to build a connection string in a URL format. A type must be set in attributes of a base secret. For PostgreSQL, the type field in attributes should always be `postgres`.

Create a file named `postgres_base.json`, substituting your values:

```
{
  "host": "your.host",
  "port": 5432,
  "database": "postgres",
  "username": "postgres",
  "password": "myp@ssword"
}
```

Create a secret using the CLI at a path of your choosing:

```
dsv secret create \
    --path db:postgresql:root \
    --data @postgres_base.json \
    --attributes '{"type": "postgres"}'
```

Dynamic Secret

A dynamic secret will be linked to the base secret. One base secret can have many dynamic secrets linked to it.

Create a file named `postgres_dynamic1.json`, substituting your values:

```
{
  "linkConfig": {
    "linkType": "dynamic",
    "linkedSecret": "db:postgresql:root"
  },
  "grantPermissions": {
    "what": "ALL PRIVILEGES",
    "where": "postgres"
  },
  "pool": "pool1",
  "ttl": 1000,
  "userPrefix": "usr"
}
```

Create a dynamic secret using the CLI at a path of your choosing:

```
dsv secret create --path db:postgresql:dynamic1 --attributes @postgres_dynamic1.json
```

Note: when creating a dynamic secret the `data` field should be empty.

Attributes description:

1. **`linkConfig`**: denotes that it is a dynamic secret with a link to a base secret:
   - **`linkType`**: should always be **`dynamic`**
   - **`linkedSecret`**: sets a path to base secret
2. **`grantPermissions`**: defines access privileges
   - **`what`**: a specific privilege, e.g. `SELECT`, `INSERT`, `UPDATE`, `DELETE`
   - **`where`**: a database object, e.g. a table name, a view name, a database name
3. **`pool`**: a pool name to use
4. **`ttl`**: a number of seconds before the engine automatically deletes new credentials, **must be set at or above 900**
5. **`userPrefix`**: *an optional field* that defines a prefix for a new username

To create a new user, the CREATE USER command is used.

To assign privileges, the GRANT command is used.

```
GRANT <"what"> ON <"where"> TO <"username">;
```

Sending a PostgreSQL Task to Engine

Read the PostgreSQL dynamic secret. A randomly chosen engine in the engine pool should receive the task and perform it. The engine attempts to create a PostgreSQL account and reports back success or failure. On success, the user also receives the new working credentials. As long as there is at least one running engine in a given pool, an engine will receive a PostgreSQL account revocation task and delete the account once its TTL expires.

List PostgreSQL Base Secrets

To find all base secrets that are related to PostgreSQL run:

```
dsv secret search --query "postgres" --search-field "attributes.type"
```

List Dynamic Secrets

To find all dynamic secrets that are linked to a specific base secret run:

```
dsv secret search --query "db:postgresql:root" --search-links
```

Read Dynamic Secret Attributes

Using the `secret read` CLI command to read a dynamic secret will initiate a creation of a new credentials. To read a dynamic secret use the `secret describe` CLI command instead.

Example:

```
dsv secret describe db:postgresql:dynamic1
```

The `secret describe` does not return the secret data field, but for dynamic secrets it is always empty.

Third Party Reference

For server configuration details, refer to Postgresql documentation.

MongoDB Dynamic Secrets

Once you have installed the DSV Engine, you can use DSV to create dynamic secrets.

Base Secret

Base secret data defines how to establish a connection with a MongoDB server. All values are required and will be used to build a connection string in a URL format. A type must be set in attributes of a base secret. For MongoDB, the type field in attributes should always be `mongo`.

Create a file named `mongodb_base.json`, substituting your values:

```
{
  "host": "your.host",
  "port": 8081,
  "username": "mongodb",
  "password": "myp@ssword"
}
```

Create a secret using the CLI at a path of your choosing:

```
dsv secret create \
    --path db:mongodb:root \
    --data @mongodb_base.json \
    --attributes '{"type": "mongo"}'
```

Dynamic Secret

A dynamic secret will be linked to the base secret. One base secret can have many dynamic secrets linked to it.

Create a file named `mongodb_dynamic1.json`, substituting your values:

```
{
  "linkConfig": {
    "linkType": "dynamic",
    "linkedSecret": "db:mongodb:root"
  },
  "grantPermissions": {
    "what": "readWrite",
    "where": "mydb"
  },
  "pool": "pool1",
  "ttl": 1000,
  "userPrefix": "usr"
}
```

Create a dynamic secret using the CLI at a path of your choosing:

```
dsv secret create --path db:mongodb:dynamic1 --attributes @mongodb_dynamic1.json
```

Note: when creating a dynamic secret the `data` field should be empty.

Attributes description:

1. **`linkConfig`**: denotes that it is a dynamic secret with a link to a base secret:
   - **`linkType`**: should always be **`dynamic`**
   - **`linkedSecret`**: sets a path to base secret
2. **`grantPermissions`**: defines access privileges
   - **`what`**: a specific MongoDB role name, e.g. `read`, `readWrite`
   - **`where`**: a database name
3. **`pool`**: a pool name to use
4. **`ttl`**: a number of seconds before the engine automatically deletes new credentials, **must be set at or above 900**
5. **`userPrefix`**: *an optional field* that defines a prefix for a new username

To create a new user and assign privileges, the db.createUser() method is used.

Sending a MongoDB task to an engine

Read the MongoDB dynamic secret. A randomly chosen engine in the engine pool should receive the task and perform it. The engine attempts to create a MongoDB account and reports back success or failure. On success, the user also receives the new working credentials. As long as there is at least one running engine in a given pool, an engine will receive a MongoDB account revocation task and delete the account once its TTL expires.

List MongoDB Base Secrets

To find all base secrets that are related to MongoDB run:

```
dsv secret search --query "mongo" --search-field "attributes.type"
```

List Dynamic Secrets

To find all dynamic secrets that are linked to a specific base secret run:

```
dsv secret search --query "db:mongodb:root" --search-links
```

Read Dynamic Secret Attributes

Using the `secret read` CLI command to read a dynamic secret will initiate a creation of a new credentials. To read a dynamic secret use the `secret describe` CLI command instead.

Example:

```
dsv secret describe db:mongodb:dynamic1
```

The `secret describe` does not return the secret data field, but for dynamic secrets it is always empty.

Third Party Reference

For server configuration details, refer to MongoDB Database Documentation.

DSV Engine

Starting an Engine

There are three methods for creating and starting an engine. They are:

- Using the **DSV-Engine Program** and the **wizard**. This option simplifies engine creation into workflows.
- Using the **DSV-Engine Program** and **flags**. This option allows manual input of flags to register and run an engine.
- Using the CLI and DSV-Engine Program **separately**. This option allows creation of an engine and engine pool in the CLI before running the engine using the dsv-engine program.

NOTES:

1. The first time an engine is created, a matching configuration file called **.dsv-engine-config.yml** will also be created in your home directory. The `dsv-engine run` command will automatically use the values in this file unless another configuration is specified. You can create multiple configuration files and use them by specifying the path along with the run command (i.e., `dsv-engine run --config dsv-engine-config2.yml`).
2. Setting up the Engine with **Oracle** Databases has separate requirements. See the Oracle page for instructions.

Engine Wizard

Guide	CLI
1. Download the **dsv-engine program** for your operating system. The example uses **dsv-engine** as the program name.	https://dsv.secretsvaultcloud.com/downloads
2. Begin the **registration wizard** and follow the prompts in the CLI to register the engine. Note that the user-token is your authorization token found using the `dsv auth` command in the DSV CLI.	`dsv-engine register --wizard`
3. Start the engine using `dsv-engine run` or the **run wizard**.	`dsv-engine run`**OR**`dsv-engine run --wizard`

Engine Flags

Guide	CLI
1. Download the **dsv-engine program** for your operating system. The example uses **dsv-engine** as the program name.	https://dsv.secretsvaultcloud.com/downloads
2. Use the **register** command followed by the required flags to register the Engine. **Flags:** **--engine-name**: The name of the new engine. **--pool-name**: The name of the new pool. If you omit pool-name, the engine will generate a random name for the engine pool. **--endpoint:** The location of the engine. Use your tenant name followed by the domain you wish to use. **--user-token:** The authorization token from the **dsv CLI**. Use the command `dsv auth` to retrieve the token.	`dsv-engine register --endpoint <tenantname>.secretsvaultcloud.com--engine-name <exampleengine> --user-token <exampletoken>`
3. Use the **run** command to start the engine.	`dsv-engine run`

CLI & Engine Program

To start a DSV Engine, perform the following actions. The example uses the placeholders `examplepool` and `exampleengine`, replace these with the correct engine and pool names for your organization.

Guide	CLI
1. Create an **Engine pool**.	`dsv pool create --name examplepool`
2. Create an **Engine** and assign it to the pool. **Notes:** The create command will return a private key and endpoint. **Make sure to save the private key for Engine registration. It cannot be retrieved later.** An Engine can only be assigned to one pool.	`dsv engine create --name exampleengine --pool-name examplepool`
3. Install the **dsv-engine program**. The example uses **dsv-engine** as the program name. *If you use the same name, make sure to include the dash when performing registration in step 4.*	https://dsv.secretsvaultcloud.com/downloads
4. **Run** the Engine.	`dsv-engine run --endpoint <tenantname>.<secretsvaultcloud.com>--engine-name exampleengine --private-key exampleprivatekey`
5. (Optional) **Ping** the Engine to ensure connectivity.	`dsv engine ping --name exampleengine`
6. (Optional) For support using the Engine Binary, use the built-in CLI help commands.	`dsv-engine register -h` and `dsv-engine run -h`

Starting an Engine in a Container

To start an engine in a container, pull the appropriate image and run a container from it. The result will depend on the environment variables you provide to the new container. If you had created a pool, but not engine, you can register a new engine and start it in one step:

Note: DSV_TOKEN is used to authenticate into the API. It can be generated in the CLI with `dsv auth`.

```
docker run -e DSV_ENGINE=exampleengine -e DSV_POOL=examplepool -e DSV_
ENDPOINT=<tenant.secretsvaultcloud.com> -e DSV_TOKEN=<tokentext> dsv-engine
```

You should see the private key and other information about the new engine displayed once it has been registered, and the container has been started. Store the private key and other information securely.

If you already have a registered engine and want to run it in the container, then provide a different set of environment variables:

```
docker run --name eng --rm -e DSV_ENGINE=exampleengine -e DSV_
ENDPOINT=<tenantname>.secretsvaultcloud.com -e DSV_PRIVATE_KEY=<privatekey> dsv-engine
```

On a successful engine start, you should receive a response saying that the engine is ready and waiting for messages.

List of environment variables for engine Docker container

- ENGINE_NAME
- DSV_POOL
- DSV_TOKEN
- DSV_PRIVATE_KEY
- DSV_ENDPOINT
- DSV_VERBOSITY (warn,debug,error,info)

Installing the Engine as a Service/Daemon

Supported Service Frameworks/Process Managers

The DSV Engine can be installed as a service/daemon using:

Windows

- Windows Services Manager

Linux

- SystemD
- SysV
- Upstart

macOS/OSX

- LaunchD

Installation Commands

Commands/Subcommands	Usage
`dsv-engine service install`	Install the engine as a service with one of the supported service frameworks / process managers.
`dsv-engine service uninstall`	Uninstall the engine service.
`dsv-engine service start`	Start the engine service.
`dsv-engine service stop`	Stop the engine service.
`dsv-engine service restart`	Restart the engine service.
`dsv-engine service status`	Get the current status of the service.

Installation Steps

1. Register the engine using the normal workflow (e.g. `dsv-engine register`)
2. Using an account with the appropriate permissions, run: `dsv-engine service install`.
3. Run: `dsv-engine service start`, or restart the machine.

Encryption as a Service

DSV offers both a fully managed and a user managed Encryption as a Service (EaaS). DSV is able to encrypt/decrypt strings and files under 2MB via the fully-managed encryption API, the manual encryption API or in the CLI using the `crypto` command. The key used for the encryption and decryption is stored as a secret-like object within DSV's architecture. The operations of encrypting and decrypting data are done on-the-fly. Those results are returned to the caller immediately and are not saved within DSV.

Management Subcommands

Management subcommands distinguish whether the encryption key is generated automatically or provided manually.

Subcommand	Function
`auto`	DSV *automatically* generates the encryption key. DSV will default to `auto` if manual is not specified.
`manual`	Users *manually* provide the encryption key. Manual must be specified for each input when encrypting with user supplied keys.

Operation Subcommands

Operation subcommands act on files and strings.

Subcommand	Function	Example
`decrypt`	Function Decrypts a file or string.	`dsv crypto decrypt --path mykeys/key1 --data @file.txt.enc`
`encrypt`	Encrypts a file or string.	`dsv crypto encrypt --path mykeys/key1 --data @file.txt`
`rotate`	Rotates both data and encryption keys to new versions. **For use with `auto` EaaS only.**	`dsv crypto rotate --path mykeys/key1 --data 'ciphertextstring' --version-start 0`

Key Management Subcommands

Key Management subcommands act on encryption keys.

Subcommands	Function	Example
`key-create`	Generates a new encryption key. *Used only with managed (`auto`) encryption. Use `key-upload` to supply your own key.*	`dsv crypto key-create --path mykeys/key1`
`key-delete`	Mark an encryption key for deletion. The key **and all of its versions** will be removed in about 72 hours. A key that is marked for deletion but not yet removed can be restored using `key-restore`.	`dsv crypto key-delete --path mykeys/key1`
`key-read`	Displays the readable data of the encryption key. Reading a `manual` key will show the key and metadata. Reading an `auto` key will display only metadata.	`dsv crypto key-read --path mykeys/key1`
`key-restore`	Restores a key that is marked for deletion. Fully removed keys cannot be restored.	`dsv crypto key-restore --path mykeys/key1`
`key-update`	Creates a new version of a user supplied encryption key. The `--private-key` flag is required. *For use with `manual` encryption only.*	`dsv crypto manual key-update --path mykeys/key1 --private-key MnI1dTh4LOE/RCHKO...QiY=`
`key-upload`	Uploads a new, user supplied, AES256 (symmetric) encryption key to DSV. The `--scheme` and `--private-key` flags are required. **The encryption key must be AES-256, symmetric, base 64 encoded**.	`dsv crypto manual key-upload --path mykeys/key1 --scheme symmetric --private-key MnI1dTh4LOE/RchHkOtiUGVTaFZt...QiY= --nonce S1Nze...1Bz`

Flags

Flags accompany subcommands to set preferences.

Flag	Function	Example
`--data`	Selects the file or string to be encrypted or decrypted	`--data 'secret string'`
`--nonce`	Sets the nonce value for `manual` encryption. If omitted, DSV will generate a nonce value.	`--nonce S1Nze...1Bz`
`--out`	Sets the output name of a decrypted file.	`--out secret.txt`
`--path`	Points to the location of the encryption key.	`--path mykeys/key1`

Flag	Function	Example
`--scheme`	Sets the scheme for `manual` keys.	`--scheme symmetric`
`--version`	Sets the version of the key to use when decrypting data.	`--version 0`
`--version-end`	Sets the target key version when reencrypting data.	`--version-end 4`
`--version-start`	Sets the current key version to begin rotation.	`--version-start 0`

Encrypting Data

Encrypting data requires three steps:

- Create or Upload an encryption key.
- Encrypt the file or string.
- Decrypt the file or string.

Note: Fully-managed encryption uses the `auto` subcommand. When using fully-managed encryption, you do not need to specify `auto` because it is the default for the `crypto` command. When providing your own keys, be sure to use the `manual` subcommand for each input.

Automatic Key Creation

To create a fully-managed, automatically generated encryption-key:

1. In DSV, create an encryption key using the subcommand and flags: `dsv crypto key-create --path mykeys/key1`. Substitute your own path and key name for `mykeys` and `key1`.
2. The CLI returns a confirmation of key creation. This metadata can also be read using the `dsv crypto key-read --path mykeys/key1` command:

```
{
"created": "2021-03-01T19:12:58z", "createdBy": "users:thy-
one:your.username@organization.com", "id": "identificationstring", "lastModified":
"2021-03-01T19:12:58z", "lastModifiedBy": "users:thy-
one:your.username@organization.com", "path": "mykeys:key1", "version": "0"}
```

Manual Key Creation

To upload your own encryption key:

1. In DSV, upload an encryption key using the subcommand and flags: `dsv crypto manual key-upload --path mykeys/key1 --scheme symmetric --private-key MnI1dTh4L0E/RchHk0tiUGVTaFZt...QiY= --nonce S1Nze...1Bz`. **The `private-key` that you supply must be AES 256, symmetric, 64 bit encoded. The `scheme` value must be "symmetric".** If the `nonce` value is omitted, DSV will generate it for you.

2. The CLI returns a confirmation of key upload. This data can also be read using the `dsv crypto manual key-read --path mykeys/key1` command:

```
{
"created": "2021-03-01T19:12:58z", "createdBy": "users:thy-
one:your.username@organization.com", "data": {    "metadata": null,    "nonce":
"S1Nze...1Bz",    "privateKey": "MnI1dTh4LOE/RchHkOtiUGVTaFZt...QiY=",    "scheme":
"symmetric"}, "description": "", "id": "identificationstring", "lastModified": "2021-03-
01T19:12:58z", "lastModifiedBy": "users:thy-one:your.username@organization.com", "path":
"mykeys:key1", "version": "0"}
```

String Encryption

After creating or uploading an encryption key, follow these steps to encrypt a string. If you are using a manually supplied key, be sure to include the `manual` subcommand after the `crypto` command in the examples.

1. Encrypt the string using the `dsv crypto encrypt` subcommand along with the encryption key `--path` and the string `--data.`

```
dsv crypto encrypt --path mykeys/key1 --data 'Example String'
```

2. The CLI returns a confirmation of encryption.

```
{
"ciphertext": "zIPFkidTB51...cz2CEZ4+n","path": "mykeys/key1","version": "0"}
```

3. Make sure you save the ciphertext string and version. You will need that information when attempting to decrypt in the future.
4. Decrypt the string using the `dsv crypto decrypt` subcommand along with the same encryption key `--path` and the ciphertext as the `--data` value.

```
dsv crypto decrypt --path mykeys/key1 --data 'zIPFkidTB51...cz2CEZ4+n'
```

5. The CLI returns the value of the decrypted string.

```
{
"data": "Example String","path": "mykeys/key1","version": "0"}
```

File Encryption

After creating or uploading an encryption key, follow these steps to encrypt a file. If you are using a manually supplied key, be sure to include the `manual` subcommand after the `crypto` command in the examples.

Note: The maximum file size is 2MB including overhead associated with DSV encoding and transporting.

Usage

1. Encrypt the file using the `dsv crypto encrypt` subcommand along with the encryption key `--path` and the `--data` flag pointing to the file location. (*Optional*) Give the encrypted file a new name using the `--out` flag. If no new filename is specified, DSV will append **.enc** to the original filename.

   ```
   dsv crypto encrypt --path mykeys/key1 --data @file.txt
   ```

2. DSV saves the encrypted file. The CLI returns a confirmation of encryption.

   ```
   Ciphertext with metadata successfully saved in file.txt.enc
   ```

3. Decrypt the file using the `dsv crypto decrypt` subcommand along with the same encryption key `--path` and the new .enc file as the `--data` value. (*Optional*) Give the decrypted file a new name using the `--out` flag. If no new filename is specified, DSV will append **.txt** to the file name.

   ```
   dsv crypto decrypt --path mykeys/key1 --data @file.txt.enc --out decryptedfile.decrypted
   ```

4. DSV decrypts and saves the file. The CLI returns confirmation of decryption.

   ```
   Decrypted data with metadata successfully saved in decryptedfile.decrypted
   ```

The decrypted file will contain the metadata associated with the original encrypted file (i.e., version, path, and data). The data value remains base64 encoded. If you want to obtain the original file, you will need to base64 decode the data value.

```
# Linux -- prerequisites: jq, base64, md5sum
> jq -r '.data' decryptedfile.decrypted | base64 -d > decryptedfile.original
> md5sum file.txt decryptedfile.original
adbb83c57dc433b3a1d0e887ea3c029f  file.txt
adbb83c57dc433b3a1d0e887ea3c029f  decryptedfile.original
```

Key Rotation and Versioning

For fully-managed (`auto`) encryption, both keys and data can be rotated.

Note: A new version of a key can only be created by rotating data.

When data is rotated, it is decrypted using the original encryption key, and reencrypted with the new version.

Note: The original version of a key is designated as **Version 0**.

Creating a New Key Version

A new key version is created automatically when encrypted data is rotated *using the most recent version of the key.* To rotate an encryption key and data to a new version:

1. Use the rotate subcommand along with the following.
   - the `--path` to the key to be rotated
   - the already encrypted data (ciphertext or file) from the previous version as the value for `--data`
   - the current **version number** of the data as the value for `--version-start`
   - (Optional) For files, the `--out` flag can be used to specify the name of the reencrypted file.

   ```
   dsv crypto rotate --path mykeys/key1 --data 'zIPFkidTB51...cz2CEZ4+n' --version-start 0
   ```

2. The data is now re-encrypted as version 1, and key version 1 has been created.

   Note: If version 0 of the ciphertext/file is saved, it can still be decrypted using version 0 of the key. The newly returned version 1 ciphertext/file can only be decrypted using version 1 of the key.

3. The CLI returns a confirmation of rotation.

   ```
   {
   "ciphertext": "pcrvO6gXy0a...k9RKKHV9n","path": "mykeys/key1","version": "1"}
   ```

4. The string or file can now be decrypted by passing the new `--data` value along with the `--version` number. If no `--version` is set, DSV will default to the most recent version of the key.

   ```
   dsv crypto decrypt --path mykeys/key1 --data 'pcrvO6gXy0a...k9RKKHV9n' --version 1
   ```

Rotating to an Existing Key Version

To rotate data to an existing version of a key:

1. Use the rotate subcommand along with the following.
   - the `--path` to the key.
   - the already encrypted data (ciphertext or file) from the previous version as the value for `--data`.
   - the **version number** of the key with which the data was previously encrypted as the value for `--version-start`.
   - the new version of the key to use for encryption as the value for `--version-end`.
2. This example input will rotate the file from version 3 of the key to version 6.

```
dsv crypto rotate --path mykeys/key1 --data @passwordv3.enc --version-start 3 --version-end 6 --out @passwordv6.enc
```

3. The CLI returns a confirmation of data rotation.

```
{
"file": "@passwordv6.enc","path": "mykeys/key1","version": "6"}
```

4. The new file can now be decrypted using version 6 of the key.

Manual Key Updating

For user supplied (`manual`) encryption, key values can be updated. Note that the original version of a key is designated as version 0.

To update a key:

1. Use the `key-update` subcommand along with the following.
   - the `--path` to the existing key
   - the new key as the value for `--private-key`
   - (optional) a new `--nonce` string
   - **Example**: `dsv crypto manual key-update --path mykeys/key1 --private key MnI1dTh4L0E/RchHk0tiUGVTaFZt...QiY=`
2. The CLI returns a confirmation of the key update. Note that the newly updated key is now designated as version 1.

```
{
 "attributes": null, "created": "2021-03-01T19:12:58z", "createdBy": "users:thy-one:your.username@organization.com", "data": {    "metadata": null,    "nonce": "S1Nze...1Bz",    "privateKey": "MnI1dTh4L0E/RchHk0tiUGVTaFZt...QiY=",    "scheme": "symmetric"}, "description": "", "id": "identificationstring", "lastModified": "2021-03-01T19:12:58z", "lastModifiedBy": "users:thy-one:your.username@organization.com", "path": "mykeys:key1", "version": "1"}
```

3. All encrypted files or strings must be decrypted with the key **`--version`** that was used for encryption. DSV defaults to using the most recent version unless a version is specified.

Certificate Issuance

DevOps Secrets Vault provides the following functionality:

- The ability to generate and sign leaf (end-entity) certificates or to create and sign a certificate from a certificate signing request (CSR).
- The ability to generate and issue leaf (end-entity) certificates for to issue a certificate from a certificate signing request (CSR) defined by RFC-7512.

All certificates assume **RSA 2048** key-pairs and **SHA-256 Hashing**.

A signing certificate is required and it may be generated in DSV or imported from an outside Certificate Authority (CA). This documentation will often refer to the signing certificate as the "root" certificate. However, in the case of a signing certificate being imported from an outside CA, best practices would be to use an intermediate certificate as the DSV signing certificate.

All the `dsv pki <action>` commands start a workflow if no flags are added. However, `--help (or -h)` can be used for help. In these examples we provide the direct commands.

Generate a Signing Certificate

The command to generate a self-signed root certificate and private key is `dsv pki generate-root`.

Flag	Description
common-name	Required - The domain name of the root CA.
rootcapath	Required - Path and name of a secret that will contain the signing certificate.
domains	Required - List of domains that this signing certificate is allowed to sign leaf certificates.
maxttl	Required - Maximum time to live in hours for a leaf cert signed with this signing certificate. This also sets the expiration date (time) of this root certificate.
crl	Optional - URL where customer-supported certificate revocation list (CRL) resides.
country	Optional.
state	Optional.
locality	Optional.
email	Optional.
organization	Optional.

This command generates a root certificate named *foobar.org* and corresponding private key for signing leaf certificates with the common name *foo.org* and/or *bar.org*. They are saved in the secret path, `ca/myroot`, that is referenced when a leaf certificate is generated and/or signed.

```
dsv pki generate-root --rootcapath ca/myroot --domains foo.org,bar.org --common-name
foobar.org --organization FooBar,Inc --country US --state IA --locality Boone --maxttl
1000
```

The output from the above command only shows the certificate and is base64 encoded.

To retrive the root certificate and private key, run `dsv secret read --path ca/myroot`.

Usage


```
{
  "attributes": {
    "type": "root-cert"
  },
  "created": "2020-04-09T20:29:41Z",
  "createdBy": "users:thy-one:dsvtest9519@mailinator.com",
  "data": {
    "cert":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURnakNDQW1xZ0F3SUJBZ0lFTVp4NWJqQU5CZ2txaGtpRzl3M
EJBUXNGQURCaE1Rc3dDUVlEVlFRR0V3SlYKVXpFTE1Ba0dBMVVFQ0JNQ1NVRXhEakFNQmdOVkJBY1RCVUp2YjI1bE1
STXdFUVlEVlFRS0V3cEdiMjlDWVhJcwpTVzVqTVFrd0J3WURWUVFMRXdBeEZUQVRCZO5WQkFNVERIUm9lV052ZEdsa
kxtTnZiVEFlRncweU1EQTBNRGt5Ck1ESTVOREZhRncweU1EQTFNakV4TWpJNU5ERmFNR0V4Q3pBSkJnTlZCQVlUQWx
WVE1Rc3dDUVlEVlFRSUV3SkoKUVRFT01Bd0dBMVVFQnhNRlFtOXZibVV4RXpBUkJnTlZCQW9UQ2tadmIwSmhjaXhKY
m1NeENUQUhCZO5WQkFzVApBREVWTUJNR0ExVUVBeE1NZEdoNVkyOTBhV011WTI5dE1JSUJJakFOQmdrcWhraUc5dzB
CQVFFRkFBT0NBUThBCk1JSUJDZ0tDQVFFQXRVUjFKaDZ4UkdRYVZ0OWhvaUdvWjdiN3JTVzk3YVFhRnprK2VESUNhZ
ThFSjFpYkdSQlAKVFJJMUZHLzlnMUtNTFhPUjArcDRWSHlvYjhzVVhSb0tYeHZZa2t4eXM4RjBoVVdEblUxZHJFVXh
rZGk0R3BhdQpObEJJaWhmblpRdmtnY0txMzFoYktpSlIwaTU0b0NnNjhyNVY2VUY4bVpNQWloa2cya012emFJMFE0T
GE2d3FaCjlSRlFSUlJLRkIzNEx6SUdnaFpDSldTUkY2UDZnSWJpM2VOck1KRWdsaUdqb1FYWjJlanJ1RURWaHhqQ29
5WjYKdmdUdDIza2dxWnNOQUxxUE9CazJGeGZZQ3FuS2d3TTdRYTNRdmdNeVE0eG5KSTBqTUJaVWpFU0IvSmRiRVo5e
QplckhsZGpSYnFSUjhrR0RsYksweDBkUW1jNHpUQitOc0JRSURBUUFCbzBJd1FEQU9CZO5WSFE4QkFmOEVCQU1DCkF
vUXdIUVlEVlIwbEJCWXdGQVljS3dZQkJRVUhBd0lHQ0NzR0FRVUZCd01CTUE4R0ExVWRFd0VCL3dRRk1BTUIKQWY4d
0RRWUpLb1pJaHZjTkFRRUxCUUFEZ2dFQkFBcEZNYWhFM1FINHQ3U0gzczNNK1ZUSGJpSWhrUnVxazVVZQozK1M2Ykp
iL3ROckRVTE5lSFkyaDBPRGpmcWI3QWk5RElSMjc3dW8vVkhOQW1zWno1bEJ5TjJLZSs3YUxXY2FTClVoek1FVUt6c
m4vMW9OT2Q5S2RuVWJ1cS8xNEVCVmUyb0t4Y1k1cHdJZTZnMkpVMW5oSGM2SENENmJVNVRnVmgKbzNWclJ0NVA5VUs
4aWsraUlDbktObVRJUWhsRDVhZ2VJeVpOUmYyQ01xdzR0TldMRzU4b011UTQrcjVwY2VqegpFSGI1UHpiR29wMGI3N
UdyQVFZbWhFU2d4SnVUZWI3WnZiTUIxbG5QdnFyWWNCN09MR2VyaDY4bHZ4K1NadVk2CmE2NldORmNobjFlR3cOWlQ
xdzl4Vk5VOVhqRndvbjRqaG9VdlRxR0k0L2c0NlJVY1NoZzOKLS0tLS1FTkQgQOVSVElGSUNBVEUtLS0tLQo=",
    "domains": ["foo.org", "bar.org"],
    "maxTTL": 1000,
    "privateKey":
"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBdFVSMUpoNnhSR1FhVnQ5aG9pR
29aN2I3clNXOTdhUWFGemsrZURJQ2FlOEVKMWliCkdSQlBUUkkxRkcvOWcxS01MWE9SMCtwNFZIeW9iOHNVWFJvS1h
4dllra3h5czhGMGhVV0RuVTFkckVVeGtkaTQKR3BhdU5sQklpaGZuWlF2a2djS3EzMWhiS2lKUjBpNTRvQ2c2OHI1V
jZVRjhtWk1BaWhrZzJrTXZ6YUkwUTRMYQo2d3FaOVJGUVJSUktGQjMOTHpJR2doWkNKV1NSRjZQNmdJYmkzZU5yTUp
FZ2xpR2pvUVhaMmVqcnVFRFZoeGpDCm95WjZ2Z1R0MjNrZ3FacO5BTHFQT0JrMkZ4ZllDcW5LZ3dNN1FhM1F2ZO15U
TR4bkpJMGpNQlpVakVTQi9KZGIKRVo5eWVySGxkalJicVJSOGtHRGxiSzB4MGRRbWMOelRCKO5zQlFJREFRQUJBb0l
CQUJYYklUenRhblpTazVKeAo4TFc1MVRKYOw5QmF3cUhLclpLclJrcjd6S3ExTlEwQmRBSDdvM1FwZzlqby8rbzdvO
GMvTGhBZEwxRVFqc2FiCjkrS1o1ekk4aTBwb2lWUC9PV3R3VEVSRk5jdzFzNXBnUlNKL2xKWGI3RU1xU3EOMlZ1RUd
kYy9rT1duRkpaUncKSWY4OW1vMzJRU21VeWM5Q21FZO9hNVdsaORmODZLYjJMS2ZscXE1QWkybCs2VVRQTGovejlpT
GhDcTdqTFRtVwpaSzVhcVdaUnpNQ24rVEhnNEdUY2dBeWlOVzJnbUo2RFBSWldzaHJSUUJ2VVloY1JjSnBKN3FQb3h
EOGpMNXIyCmVXVOUzZGs1bzJSdG5aZFRZUO95NOo4ZFM5c2FOSk1UQmdxNOZNbkFRR2EOS2piYStkQ3RuVjRPaGhiV
24OdGIKR2NtUjJvRUNnWUVBeFdlQnpvR3p2RnJqSGl3ZmYraVlYUnFvcEJrROVBdOgvUC9SUzFQMGNnbjNuYkFKdzZ
OegpEbW1SSHlDNHhFQXhOZzVqZ25mdkMvYS9UcnZXMy9JY3doZzdMMUtIajh6d2NrOGFvWDdOZFNjWVFCZ2wObU1CC
nNDaVpicmdwblVBbHUxZFRLZ3BULzVYZzBERXlHUE15VkpIZmF3cGprVOp1QTdSejRtYjczdkVDZ1lFQTZ4SzYKWHZ
UVWFzcFk1OWV5emhFU1RhNEdzVTFMRzMwRTJCbOowS1h6dEQ4TkVtMkZTMlRJQ2Jsbk9Rakxod2RpU3E5MgptNnZXe
jVpVG1teGwvMHI2cEhZL3Q1RmJOOEV5eHVzZGdCbDBNTkR1THM2bTRubU5uWXpVSTlqOUF6ajg1alVPCmdaTTJlSOl
zMDNqMGZudG1vejQzYXRnVOM5R3EvOEl2eGOwVXhsVUNnWUFGeWcxU2d4ZEVWTjRJU243NS8xWkkKbExtUlpuSjVFZ
OZCKORhcElPTXdYUDUORDJ1WjR6ZENtdkgObWJzUmRsaDdIMXpudktDMEZ4NXhMcTBVaOVNcgpwZzVHU3dOU3drM2k
3Rkw1bllCbENTcDY1cnBsczBXZlp2Rm8vOW1vbFBNR1ZYOUkObGlvVERyMW9CdTZBNWZjClJ4TG9UcnV3emRRdOk2Q
3FhUjdGNFFLQmdGNm1oOHc4SUZOdlppVFR3UGNnQUpLdWc1dFlWK21WaVNISO9qRjgKNElldTYOQOVBS3UreEp6RnZ
qNUV3RTU2TnFXRHlPb2RZcnpyM21MTFNyWmtaazlhSFlXNFRWWkJ3RVEvM3Z6NQpRcO4xSExKVUd2WU5vMnZRaklwe
WtFMS8OTFNBbOhxajM4YnE1Y213WmlHWFpsaE1jTnZnYmVBTWFDSGErb21XCjJrcVJBbOdCQUkyQW8zdk1Uei84ckc
3SFd6blVMLOw1OWZwaUMrVXJvUXUwcUxxR1BDTkQ2d2kyUy9lNkFFS1IKMWhQRWJ1b1NvUG4vaExhaDNHL3VsWk9tM
mU3d1Z6dHpoblRIbUkOWGZrbENaUWV4Q3BQOE9wUDlKUDZHZVVVOQpMbHpaSkFjZHVFck5zb2pXcTluYVhCZkdZUFk
ydOkvOXZyQ29HUGhDMXVWMURnVFlQNk9ZCiOtLSOtRU5EIFJTQSBQUklWQVRFIEtFWSOtLSOtCg=="
```

```
  },
  "description": "",
  "id": "90de1c6b-3c85-42cf-9d6a-758b48f1daf5",
  "lastModified": "2020-04-09T20:29:41Z",
  "lastModifiedBy": "users:thy-one:dsvtest9519@mailinator.com",
  "path": "ca:myroot",
  "version": "0"
}
```


Register (Import) a Signing Certificate

The command to register a signing certificate and private key generated outside of DevOps Secrets Vault is `dsv pki register`.

Flag	Description
certpath	Required - Path to a PEM file containing the signing certificate.
privkeypath	Required - Path to a PEM file containing the signing certificate private key.
rootcapath	Required - Path and name of a secret that will contain the signing certificate.
domains	Required - List of domains that this signing certificate is allowed to sign leaf certificates.
maxttl	Required - Maximum time to live in hours for a leaf cert signed with this signing certificate. If this is set further out in time than the expiration date of the certificate that is being registered, then there will be an error. For example, if this signing certifcate has an expiration date next week, the maxTTL maximium number is 189 hours.
crl	Optional - URL where customer-supported certificate revocation list (CRL) resides.

As an example, create a file with this certificate and name it `cert.pem`.


```
-----BEGIN CERTIFICATE-----
MIIDnjCCAoagAwIBAgIJAMOhi74h4lRqMA0GCSqGSIb3DQEBCwUAMGQxCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJJTDEQMA4GA1UEBwwHQ2hpY2FnbzEhMB8GA1UECgwY
SW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMRMwEQYDVQQDDApmb29iYXIub3JnMB4X
DTIwMDQxMDAxMjMyOFoXDTI1MDQwOTAxMjMyOFowZDELMAkGA1UEBhMCVVMxCzAJ
BgNVBAgMAklMMRAwDgYDVQQHDAdDaGljYWdvMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQxEzARBgNVBAMMCmZvb2Jhci5vcmcwggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQCxDninSZ/wDyXCcRCAgHdGxP8/YW4sX1OcStjl
qOjVVCGEr0wrLG0rDFb/KxVJ3WVM4lh381ZUT/N6qcRrl2ZPupPh9P9jjU5NkJIS
x2wIsuptRFzuw4nSBoIdDdMun0CDbscEuWUIjEdsC5kj7DPLaN16u6icOxxAH9RW
YzQoV92hsjmIZvHtzpCoVMsUMF7ONbzh54wZgajzMPV0jaGKrqLMnuhLs5O1O+AY
4k03NlfsTSNsOA8a+jjXXG331jmuQPh4UphcmUfMjpEfWw6x/qwSrxKz07k6dDWK
KcmJzqAj/MXA7coOvwj7L39uv/cMVzk/MTeLYW2Jbz7h07CBAgMBAAGjUzBRMB0G
A1UdDgQWBBTRG8SieQc672Onj/fPAQss3eA1pjAfBgNVHSMEGDAWgBTRG8SieQc6
72Onj/fPAQss3eA1pjAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IB
```

```
AQCuomjUQVYMGcPz1wzc2GJw57dTONnNyLXUdiOpGOrxhep1veFkCQmgrxAMu7Ky
ZNEoINmkHY1fOOp7hAzKIWpFBSpMwDZg/1vamjEOriJ+JxGWo2C34WZqRJHbunK5
cBmZBeER93L76Pc8k6eC/O1cus+hiqs2Mg7UggORsV+fEs6BELOKQQh+VG+rPq6C
WH9GJr9PiLD+gG6rxOZRrXt6gx1XOoK6REj1W5wMaxeS2+SKOHGPhaRE+z1xXC9z
7Y8j7UnAeE9dikJipfgj48zWskUexW6rxYK7hiz5nX3VCP1XpZp5uFhXmegJ1fmD
QxOdZF6QQRIK4MNGZ2mg1y3F
-----END CERTIFICATE-----
```


Create a file with this corresponding private key and name it `key.pem`.


```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAsQ54pOmf8A8lwnEQgIB3RsT/P2FuLF9TnErY5ajo1VQhhK9M
KyxtKwxW/ysVSd1lTOJYd/NWVE/zeqnEa5dmT7qT4fT/Y41OTZCSEsdsCLLqbURc
7sOJOgaCHQ3TLp9Ag27HBLllCIxHbAuZI+wzy2jderuonDscQB/UVmMOKFfdobI5
iGbx7c6QqFTLFDBezjW84eeMGYGo8zD1dI2hiq6izJ7oS7OTtTvgGOJNNzZX7EOj
bDgPGvo411xt99Y5rkD4eFKYXJlHzI6RH1sOsf6sEq8Ss9O5OnQ1iinJic6gI/zF
wO3KDr8I+y9/br/3DFc5PzE3i2FtiW8+4dOwgQIDAQABAoIBAEBCGUXVcadlR/X2
pN+OQDu9+UkeaibOfgDGJUvMbpwlyXhnSoSMvh4Wf2hiUXqaUE6EAOmdVeKJlbsZ
7ACEVQxwkYU7LokJ2rZJ1snb+Hh7vprjabr52oYP+J7kypUsFPTeenpbcrCUgMNU
vkKMUgvrxh3qB3qT9V/MbXrgzCgriHazR27/pPLJALnOAusu7C0XGSa7eJSY6ysO
neKWkWtJiPWa3wTp9LHxeHrkYbEd4cx2G3no1SM4IUDOUjAkHJ2OyShkyn/vXUn9
Aygnlp0s26MIgXgk46AqoROWIwRYu68FqdXdC16GRmcByALKA5XJ4Hqz9Q8ufoJf
/R9PwjECgYEA5cvcHTX+OCbzgUrtODz3ymHK2q2fSoMGGPPiBHQiqIhaVtprCpMp
6hIy4Vk/D2rHbWj+idMufnvAPjr+qJPRzIdOVmRkDyLHGq2WjBv4Owc5u4Pw+sa9
YPhQNDmCu4wABvc4lbKueP7OtAcpO4nLSk3B9ZLBnOjQNMmDVim2db0CgYEAxT8M
XawhG9LpL7tFtIQsvIxTvYlFimC5+CmnFLjcKD/1jqz8rVJSLCEtPZnh2tDcifxh
yo8UA+/nWHyOtF6JIIhfh+DqUWWWCPxJc5djwM8Zs3TrnawIBYWcl3wUM7X6FLSX
v5unb61XjPYWMU6z64cVaCH2OsCUXing9Sh4qBUCgYAOXZUwGkz/M6grYAS+bElN
VJm62/nGTbSW4MAzaRM1l/iVz2e7rIGFSYf2wH6JtzIqa9LlyNbyPOhAW63J2hvW
fm1ObU44CAOMbmen8KO4hY4dY9OvwDbclgllimba1KC3zsKxOQ7JL5y6cmwx9j5I
Md47POZvqbpCYoqcW1U1vQKBgQC6oxnUWNdLOJqlK5KdaKPcFPv30DgY48WUZ/VM
yk6nVz3HLzA34DkYwJvKOh1Xq2HCvyjZPeE2iH5jYDysnvcp7WBXdh7BxIBlKDNo
SMt+2Xf8Mpnvq6Q7dV3iiOmktIBZrzgXefVI2sCJBSGirlHYfw1mZxzh9o9tOjs+
PnlMsQKBgAUCVf5yqUGETwkv17I/2Fn+l7Hw3Yv8Ced1WKB6bwoF5Hdllr01LgpF
q1Obc+NezxCPQd+dBNBgFbcWpWvYPDfte2u6G94G8OqiOXczwu7Z3iI6puukV4Uy
8Nz6NxjrgibNpB/nuiOi36HKAyDWmo57mc7UofPCEieIK/g3DnwG
-----END RSA PRIVATE KEY-----
```


This command saves this signing certificate and key at the secret path `ca/myroot` and enables it to sign leaf certs for *foo.org* and/or *bar.org* domains (common name).

```
dsv pki register --certpath @cert.pem --privkeypath @key.pem --rootcapath ca/myroot --
domains foo.org,bar.org --maxttl 900
```

Generate and Sign a Leaf Certificate

The command to generate a leaf certificate and private key is `dsv pki leaf`.

Usage

Flag	Description
common-name	Required - The domain name that this certificate will use. This must match a domain in the signing certificate's list.
rootcapath	Required - Path and name of a secret that will contain the signing certificate. It does not matter if the signing certificate was generated by DSV or imported.
ttl	Optional - Time to live in hours. If not specified, then the maxttl of the signing certificate will be used.
store-path	Optional - Path and name of a secret that will contain this leaf certificate and private key. If not specified, then DSV will not store the leaf certificate and private key and there will be no way to retrive them after the initial stdout is deleted.
country	Optional.
state	Optional.
locality	Optional.
email	Optional.
organization	Optional.

For this example, we will request a leaf certificate for *bar.org* and use the imported signing certificate above stored at `ca/myroot`.

```
dsv pki leaf --rootcapath ca/myroot --common-name bar.org --organization FooBar,Inc --
country US --state CA --locality 'San Francisco' --ttl 24
```

A signed certificate and private key is returned in base64 encoding.

```
{
```

```
  "certificate":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURaakNDQWs2Z0F3SUJBZOlFR1lXNFRUQU5CZ2txaGtpRzl3M
EJBUXNGQURCaE1Rc3dDUVlEVlFRR0V3SlYKVXpFTE1BaOdBMVVFQ0JNQ1NVRXhEakFNQmdOVkJBY1RCVUp2YjI1bE1
STXdFUVlEVlFRSOV3cEdiMjlDWVhJcwpTVzVqTVFrd0J3WURWUVFMRXdBeEZUQVRCZO5WQkFNVERIUm9lV052ZEdsa
kxtTnZiVEFlRncweU1EQTBNVEF3Ck1qSTVNVGhhRncweU1EQTBNVEV3TWpJNU1UaGFNR0F4Q3pBSkJnTlZCQVlUQWx
WVE1Rc3dDUVlEVlFRSUV3SkQKUVRFV01CUUdBMVVFQnhNTlUyRnVJRVp5WVc1amFYTmpiekVQTUEwR0ExVUVDaE1HU
mO5dlFtRnlNUWt3QndZRApWUVFMRXdBeEVEQU9CZO5WQkFNVEIySmhjaTVqYjIwd2dnRWlNQTBHQ1NxR1NJYjNEUUV
CQVFVQUE0SUJEd0F3CmdnRUtBb0lCQVFDdUdNbmlITjM4TjRGTGdBNHlESEZTVWYrekxjREFGUWI1SGZleTNDME5VL
3RZeHNrTnNRczkKQUJkZGJyUTBMbjNVWkRNL2hVcUZIR2prSGRkUVROSTJMY2IzRGk4QWdLVU85OHVhOHVpWSttTDZ
ZK2llTE9XegozejVNNnRFOGdFbHNlQUJ4VkFwT29hTGlEZVl4MUxWOUdSUlVoZm1hZ1RFNVF4V3pmdTVKU0wyYVd2M
3RreUhMCnpFandiaGFDVHV0d0gxM1NrczN5OUNwZO91MW1qV1N3WmU0cjRGY284KzdMMEUvSDZLcG9zQk1mWTV5N24
wbm0KeU5NL2ZKM2d3eCtpSkJKa1o1RnJqRWxnNVIyZUs0aG5QdU1zeGFvY05FSElROGNXa1NTOG0zWnpNRnVjYVdFM
QpKNlNTSDQrd0ZXazBZdzA1cTRTZnQreEhGK1VocFdmZkFnTUJBQUdqSnpBbE1BNEdBMVVkRHdFQi93UUVBd0lICmd
EQVRCZO5WSFNVRUREQUtCZ2dyQmdFRkJRY0RBakFOQmdrcWhraUc5dzBCQVFzRkFBT0NBUUVBbzdtTjExRFAKb3c5Y
3VtWXJlVzdzUEFSSWxUcHBwMStIY1BNa0JhL0JvZUwrOEdtM3JDZWgyQnM4b09YQXhyVmVwSkZ5K0VNQQpIZjhQSjF
HazlMeHNzSDJQazk0OTNGMzJlVGhxUWo0d0RuQzg0TkpJZzlYMlpNSkpDSFBjc0wvVU9kenZraEhLCnkvSHk0bDl5Y
0dQdGtudmtURkVkTVdKZ2hOcFgvSkxrTFlQZWthNzFORjFPOEFaMFZVbXJXMDR0YVlDYzZ5UVAKMVljbXhSd1FLNVJ
iYWMxSWUxVEI5VWc5Z2dvUnhZOUpFKyt5aFRoMU5SK0tYUTZucWVNbk1SdStxaERONjRxVwpmMzhBU1lOMklqRndnT
VBEK3E5R3JOdWl2REYxc05lcDVDeFEzdi83S2dtNDNHTFFhZ3o2T0piblNLbmYrM2llCit3MTQxUXZJT1pDZDRnPT0
KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=",
  "privateKey":
"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBcmhqSjRoemQvRGVCUzRBT01ne
HhVbEgvc3kzQXdCVUcrUjMzc3R3dERWUDdXTWJKCkRiRUxQUUFYWFc2ME5DNTkxR1F6UDRWS2hSeG81QjNYVUV6U05
pM0c5dzR2QUlDbER2ZkxtdkxvbVBwaSttUG8Kbml6bHM5OCtUT3JSUElCSmJIZ0FjVlFLVHFHaTRnM21NZFMxZlJrV
VZJWDVtb0V4T1VNVnMzN3VTVWk5bWxyOQo3Wk1oeTh4SThHNFdnazdyY0I5ZDBwTE44dlFxWURydFpvMWtzR1h1Syt
CWEtQUHV5OUJQeCtpcWFMQVRIMk9jCnU1OUo1c2pUUDN5ZDRNTWZvaVFTWkdlUmE0eEpZT1Vkbml1SVp6N2pMTVdxS
ERSQnlFUEhGcEVrdkp0MmN6QmIKbkdsaE5TZWtraCtQc0JWcE5HTU5PYXVFbjdmc1J4ZmxJYVZuM3dJREFRQUJBb0l
CQUdMVUdZNXRHcXE1aTRFagpnV3R4MnNhRFcrY0lHdm92TlpVbktOeDAxbkpSY1VaVkdmN1d1TzE0NXNxWU5GM0c0c
EUyREUyTHllREVYdHJZCkFjbEl3ckFVem5TaXJaWFljVnFNMmh6c3RaTloxK1FSNFJRaG9vZTRPL0tIL2gwZEtoRVV
FaFJEUTlLZE9ReWcKSFVPK1h3UlR2MUczK0JoNExFdzRROUp3Uks1K1YwRysyZjlqbjQ0MO5BZGRTVWZ1UFRpVXVqe
lRTaWNGSlBKdwp0a1hYeU01VkpzVjN0VEZ2a3ZkVE43WFVhUDNLQ3ZOdU9XWFUrbG1BS21qc2xXSDBIRUJhS0NvWWV
qMyt4ZURtCnFFR1A5bXc2eFZVY0hTalgzT1BHVFJrbnR3bXNkRkQ4Z2ZJYi9RZXpVRGVnV0VvM21xSTJpQ2RLbDUwW
URLUWkKSUxzNHY1RUNnWUVBeFdxOEdPMGRCRzBkbGJtTWpEUTE5NnQ0ckhGUjhObHNzOXZ6WndOVzZ4ZOc4d0NFWnF
hTwpVNUlVeXd4YWxBL0xQVmJTdVNHQm54Sy9FQTYrZVJ2cTlxOU5UcEw5UDBDc3dpVldiMHpWdUNDQlZYRitaR3diC
kRKcVB0ZHdlb0dxNVZOaUhFUkVEemRuM0RWMVAxZzFyU09wR3BmT0w5OVpYNU9IcGoraEhob2tDZ1lFQTRjSjgKRWh
zdS9jc1ZSTjcOMGxsdzRQTU5HMFUxZO1YaTlJVkZ5dkdtQUIxQ3FGUmpZeUtFTHZqQ1h6UFN2ZTRGczRvZQpRY1Uza
UVnU1djeEFFSmJ6VTB5Sit6ZHdITkpJOFJMMzhxcTB6dVJUSG1pc2Y4cnhGZUt2QU8ONTE1N2R6WmJHClR6MTMwUTR
Nc1RKbUxyR2xSTOMrMHV5UkRqQm92RUl2V2kwV1lTYONnWUEOMWdYWlYwcW5YNUxJN0dhZVpObXkKdUZkQnJrNWMvU
HZpdkV4VE9seUh5cDhWanV5UGNSeEF5eW5aVzNFb2QzT1g4VXN4cVlmYitGV3hsYzBZcVFUNAppSGZGUzJSRnRhVUh
NQOMyWW5TVlVpWnFKd2F3ZXI4KzNiREtOdGxLYmU5MWtmRXc1S2tudHJ6OXlBT1lLTHplCmZUUmh5c0JkVmdSd0RPc
GxXQVpmb1FLQmdCRmEwQXJjU0JwK2VCNFpQZXQ5c0syNlFYR3RPbFd4NEthSGNEd1AKbzRFeXZxTU9DYTNmUTJZUS9
YQXdIYTA0RlB3ZVRBRW1WZ1NGOWRNdFhtZG9FMEIrQzhWaUY1NC9sQmZrSzJkZQpOQlFMZlZCREg2K2JQRGxBZWMrS
2dLdlFyS0JYVE5OZWtFMWoxUm55RStUWEJ5dHFVNEVIYW9jNnRYSnpiQXgwCmx0blZBb0dCQUladjU2cGNrbXRoMkJ
qZzdDdnpja2VxbHhBeUxKWU1aaW5sYjhjTDJ5UmV1NEQ5Wm0yNHdFOGkKV1N6OEwwUmFlK01Idk85bXlrckVubHhDc
Hd5aFUvL05tUDlENmZGYlh1MWpCb1h4ZUlJRWt1Wk9LdzI5Rm1MMgpFSitKV2MrRkY0cGdpZHBUMCtQL25oc2ZTVGt
4TmtZaWdCSzJ1dmVBdTJIU0NtRWNlRjBlCi0tLS0tRU5EIFJTQSBQUklWQVRFIEtFWS0tLS0tCg=="
}
```


Sign a Certificate Given a Certificate Signing Request (CSR)

The command for honoring a certificate signing request is `dsv pki sign`.

Note: The common name for the certificate in the CSR must match a domain in the signing certificate's list.

Usage

Flag	Description
`csrpath`	Required - Path to a PEM file containing the certificate signing request.
`rootpath`	Required - Path and name of a secret that will contain the signing certificate. It does not matter if the signing certificate was generated by DSV or imported.
`subjectaltnames`	Optional - List of alternative domains. They must match a domain in the signing certificate's list.
`ttl`	Optional - Time to live in hours. If not specified, then the maxttl of the signing certificate will be used.

As an example, create a file with this certificate signing request and name it `internalSite.csr`. It is requesting the common name of *foo.org* so we will sign it with the sample root certificate we generated at the top of this page.


```
-----BEGIN CERTIFICATE REQUEST-----
MIIClDCCAXwCAQAwTzELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAklMMSEwHwYDVQQK
DBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQxEDAOBgNVBAMMB2Zvby5vcmcwggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDcmthlMQcfWwZmKZr1G7aYuTLb
j/hCTIlGEhGDcp0elAEnzWGLdFUsbIMdb7Zl0/SEJLb9cVHGgcf9U67s9+1hqUPY
/xwCbHJ7JYfLHZm3XHT5oA2QUmMNqwZlh/YTwUDUr9NYslTZOUm4y6smzfO5TVOC
Z9SFETi3ZfPsknQQ3EEmPso2yJU0yqxHkgozm2bYOItd1ySEOM4R0JLQEBSgLLo4
QLtxJJZiKKVvuhGZ7SZUcXft4RxBq41uv1YyffWeZYa0b/h7hcb7Gj+pnaI/1PWm
vxdkW6cXnpAmL5k0PXlfQARGkBkUFyF3DQGDfT41UfSHE9qWi0gA6wfhXvCFAgMB
AAGgADANBgkqhkiG9w0BAQsFAAOCAQEAmL2JDxGpKmIU6OuMUsQXtylObyyIMW0q
bmmqrfccfxdV/WNLLOrm/8g0Rp/eWwAGkQY8tZJnlN+BPK6yFpx1TYW6z2aPGTUT
TgKnaheDWnpCPLkRJRqEIHYe9B+vFvEJXl1lU7pA4FGIsNV+1R2TTG4nBp8Nx7Ng
LWCFT4m90R39wCxXEJMoUOIii8mfeaFwlZstyb/pAPuQoWYebOMCTHxJsxRsr/w9
PBJsTPM+USH1xTUTtbEgY4SGFG7C+SYluFHj9c5hhH4OTPv0NH9cmMHxSsbNKbou
wmq9DFjzRXDVjAMLb2fsbBBpQ7/aT3OpJWr9jAX0/FH1Ymg2aIK89w==
-----END CERTIFICATE REQUEST-----
```


```
dsv pki sign --rootcapath ca/myroot --csrpath @internalSite.csr --ttl 24
```

The signed certificate comes back in base64 encoding.

```
{
```

```
  "certificate":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURZakNDQWtxZ0F3SUJBZ0lFRm1OYmV6QU5CZ2txaGtpRzl3M
EJBUXNGQURCaE1Rc3dDUVlEVlFRR0V3SlYKVXpFTE1Ba0dBMVVFQ0JNQ1NVRXhEakFNQmdOVkJBY1RCVUp2YjI1bE1
STXdFUVlEVlFRS0V3cEdiMjlDWVhJcwpTVzVqTVFrd0J3WURWUVFMRXdBeEZUQVRCZ05WQkFNVERIUm9lV052ZEdsa
kxtTnZiVEFlRncweU1EQTBNVEF3Ck1qRTROVGxhRncweU1EQTBNVEV3TWpFNE5UbGFNRTh4Q3pBSkJnTlZCQVlUQWx
WVE1Rc3dDUVlEVlFRSUV3SkoKVERFaE1COEdBMVVFQ2hNWVNXNTBaWEp1WlhRZ1YybGtaMmwwY3lCUWRIa2dUSFJrT
VJBd0RnWURWUVFERXdkbQpiMjh1YjNKbk1JSUJJakFOQmdrcWhraUc5dzBCQVFFRkFBT0NBUThBTUlJQkNnS0NBUUV
BM0pyWVpURUhIMXNHClppbWE5UnUybUxreTI0LzRRa3lKUmhJUmczS2RIcFFCSjgxaGkzUlZMR3lESFcrMlpUdjBoQ
1MyL1hGUnhvSEgKL1ZPdTdQZnRZYWxEMlA4Y0FteHlleVdIeXgyWnQxeDArYUFOa0ZKakRhc0daWWYyRThGQTFLL1R
XTEpVMlRsSgp1TXVySnMzenVVMVRnbWZVaFJFNHQyWHo3SkowRU54QkpqN0tOc2lWTk1xc1I1SUtNNXRtMkRpTFhkY
2toRGpPCkVkQ1MwQkFVb0N5Nk9FQzdjU1NXWWlpbGI3b1JtZTBtVkhGMzdlRWNRYXVOYnI5V01uMzFubVdHdEcvNGU
0WEcKK3hvL3FaMmlQOVQxcHI4WFpGdW5GNTZRSmkrWk5EMTVYMEFFUnBBWkZCY2hkdzBCZzMwK05WSDBoeFBhbG90S
QpBT3NINFY3d2hRSURBUUFCb3pRd01qQU9CZO5WSFE4QkFmOEVCQU1DQjRBd0V3WURWUjBsQkF3d0NnWUlLd1lCCkJ
RVUhBd0l3Q3dZRFZSMFJCQVF3QW9JQU1BMEdDU3FHU0liM0RRRUJDd1VBQTRJQkFRQkh1b2FwSk05VTVUa0IKcU5Pb
0hVMnJ3UmxjOUpRRmc5OTd3Y0UxU0dKbUNKTUd0ZkJMajZRRk80RnFJZGU5Qk90N2o0bnZwQUduYXNmaQpzbzBWa09
tK1dyZUpuRXJiL0dMK0RpMExKbGxSZHduYWJtY2NXTFVkNm5EWWxGYjZLdEdmU3dYQWJyTTh5VVZjCmdqdU1odUl5d
1ExOHR1UEFTWGFrWjUwU2VyOFd4Q3dUMlgvRDhVaGhXR1Ercno5aFV0ZHpUdU5COUdVb21PaGUKbOlXZGxHVVlpcm9
sQS9GQk9nWjZCT2gxVnQ4S3lFN0VLRjZJdU1wM3kvc2szcGVMUmpUL0dIK0JxRW5PNmhzZwpia3NOcTNGSWROYmNlT
ExlV3dLWW1ZUEdQYWFuSnZ3NnZWN3MzRlQ0TUhUaUFtVTRkbTRkZVAvNzRpZXVvTXlXCnNpZTdESkoxCi0tLS0tRU5
EIENFUlRJRklDQVRFLS0tLS0K"
}
```


SSH Key Issuance

In addition to allowing users to generate TLS certificates, DSV provides an ability to generate SSH-2 compatible public keys (currently only RSA supported) and SSH-2 certificates.

- Using SSH-2 public keys allows an administrator to place your public key on the server for which you wish to access. This is usually placed in the user's home directory `~/.ssh/authorized_keys` file.
- Using SSH-2 certificates allows DSV's specific root CA to sign the credentials which can then be used to access any SSH Server where DSV's root CA is trusted.

When users create a regular leaf or root certificate with `dsv pki leaf` or `dsv pki generate-root`, respectively, DSV automatically creates and saves an SSH-compatible public key. DSV stores it in secret data for the leaf or root secret.

```
dsv secret read myleaf
```

Among other fields, such as those for TLS private key, certificate, there will be a field for the SSH public key:


```
"sshPublicKey": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC4nmHvYaqodYKU2..."
```


Adding an SSH public key to a server

In order to authenticate to a remote server using SSH, users need to provide a regular RSA private key, such as a TLS private key DSV generates. Before doing that, users must ensure the server knows about the public key associated with the private key.

For example, administrators can edit the `.ssh/authorized_keys` file and add the public key to the list of authorized public keys for the user of that server.

Downloading keys

Below is an example of how to fetch the keys from DSV for use with SSH.

Fetching the SSH private key:

```
dsv secret myleaf -f data.privateKey | base64 -d > leaf.priv
```

Fetching the public key in SSH-2 format:

```
dsv secret myleaf -f data.sshPublicKey > leaf.pub
```

The names of the files are arbitrary.

Note: The private key must first be base64-decoded.

Authenticating

Having added the public key to the list of authorized keys, users can authenticate:

`ssh -i /path/to/leaf.priv [user@host]`

This example uses a leaf key, but the workflow is the same with a root key.

Trusting a group of keys signed by a root key

The previous example works well, but there is a maintenance problem that appears if the number of users who authenticate to one particular host increases. Administrators would then have to update the list of authorized public keys for each new key. Instead, administrators could make the server trust all keys that are signed by a root key, one that is higher in the chain of trust.

Clients can then authenticate using any leaf private key that has been signed by a certain root private key. Setting this up is a two-step process.

Adding a public root key to the server

1. First, the SSH-compatible root public key must be downloaded and saved. `dsv secret myroot -f data.sshPublicKey > root.pub`
2. A file with the key must be uploaded to the server and placed in the `/etc/ssh/` directory.
3. On the server, edit `/etc/ssh/sshd_config`. The following line appended to the file instructs the SSH daemon service to trust all keys signed by a private key associated with a given public key: `TrustedUserCAKeys /etc/ssh/root.pub` e.g., `echo "TrustedUserCAKeys /etc/ssh/root.pub" >> /etc/ssh/sshd_config`.
4. It is often a good idea to restart the SSH daemon service for changes to be applied immediately.

   `sudo /etc/init.d/ssh restart`

Generating an SSH certificate on the client side

To authenticate with a private key, users need to prove that a given leaf key has indeed been signed by a root private key that is connected with the root public key, which the server trusts. To do this, users need to generate an SSH certificate using the root private key and leaf private key. There is a special command for this.

```
dsv pki ssh-cert --rootcapath myroot --leafcapath myleaf --principals root,ubuntu --ttl 1000
```

All of the following arguments are required:

- `rootcapath` is the path to the root CA secret
- `leafcapath` is the path to the leaf CA secret
- `principals` is a list of one or more principals (user or host names) to be included in a certificate when signing a key
- `ttl` is the amount of time (by default, in hours) for which the certificate will be valid

This will return an SSH-2 signed certificate. DSV saves the certificate in the leaf secret data. Users can copy the certificate and save in a file or download it later.

```
dsv secret myleaf -f data.sshCertificate > leaf.priv-cert.pub
```

Now it is possible to try to authenticate. Users use the same `ssh` command and pass the same private key. The SSH certificate is also submitted automatically behind the scenes by `ssh`. The command tries to find the certificate in the same directory where the leaf private key is. For this reason, the certificate file must be named in a certain way: `[private key]-cert.pub`.

If there is a leaf private key file named `leaf.priv`, then the certificate must be named `leaf.priv-cert.pub`.

Then authentication works.

```
ssh -i leaf.priv [user@host]
```

Another client would just need access to the same root secret. With this root secret and a leaf secret, another user can generate an SSH certificate and use it along with the private key to authenticate. The administrators would not have to do any additional setup on the server.

Break Glass

Commands and Flags

Command	Usage	Flags
`breakglass`	Main command to configure and apply the Break Glass feature.	
`apply`	Subcommand to trigger the break glass event and recover Super Admin credentials.	

Command	Usage	Flags
	`--shares`	Flag used to pass in the distributed secret `shares` needed to recover Super Admin credentials. Pass the distributed shares for this flag.
`generate`	Subcommand to enable the break glass feature.	
	`--min-number-of-shares`	Flag used to set the minimum number of distributed `shares` needed to recover Super Admin credentials. Pass in a numerical value for this flag.
	`--new-admins`	Flag used to choose who the new administrators will be after the Break Glass event. Pass in a list of usernames for this flag.
`status`	Subcommand to return current status of Break Glass implementation.	

Break Glass Setup

To set up Break Glass, enter the `dsv breakglass generate` command along with the `--new-admins` and `--min-number-of-shares` flags. The following example will require three shares to trigger a Break Glass event and give the users username1 and username2 administrative rights following the event.

Note: The number of `new-admins` must be greater than or equal to the number of `min-number-of-shares`.

Example:

```
dsv breakglass generate --new-admins 'username1,username2,username3,username4' --min-number-of-shares 3
```

The `share` values are sent to each new admin. The admin can access this value using the command: **`dsv home __breakglass_share`**

Trigger Break Glass

To trigger a break glass event, a user must collect the minimum number of share values from users who are designated as new admins (`new-admins`). The new admins can access the value using the command: **`dsv home __breakglass_share`**. The share values must then be entered using the command:

```
dsv breakglass apply --shares '{share1},{share2},{share3}'
```

The new admins now have Super Administrator access.

Bring Your Own Key (BYOK) Configuration

Use the following steps to change AWS master keys.

1. In your AWS account, add the following permission to KMS key that is intended for use by DSV. This provides access to DSV to encrypt and decrypt using these keys.

```
            "Sid": "Allow use of the key",
            "Effect": "Allow",            "Principal": {                "AWS":
"arn:aws:iam::<delinea dsv aws account>:root"            },            "Action":
[                "kms:Encrypt",                "kms:Decrypt",                "kms:ReEnc
rypt*",                "kms:GenerateDataKey*",                "kms:DescribeKey"
    ],            "Resource": "*"        }
```

2. In the API/CLI, update the master key `arn` in DSV.

- Using the API:

  PUT `v1/config/keys`

```
 {
 "keyprovider" : "AWS",  "primaryKey" : "arn:aws:kms:us-east-1:<your aws
account>:key/<keyid>",  "secondaryKey": "arn:aws:kms:us-west-1:<your aws
account>:key/<keyid>"}
```

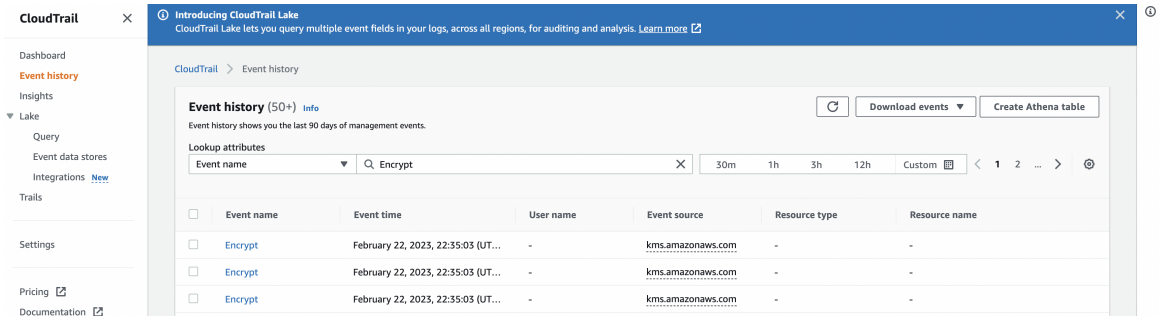
- Using the CLI:

```
dsv byok update --primary-key arn:aws:kms:us-east-1:<your aws account>:key/<keyid> --
secondary-key arn:aws:kms:us-west-1:<your aws account>:key/<keyid>
```

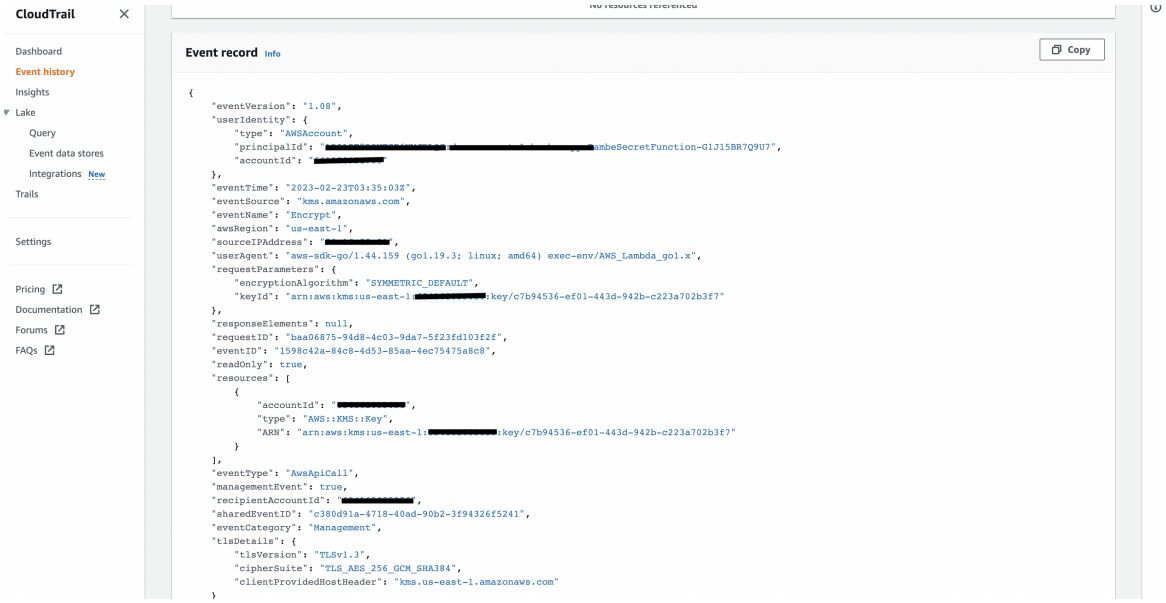
> **Note:** To update these keys, the user needs to have proper authorization to access the `v1/config/keys` API.

Verify Key Changes in Your AWS Account: Assuming CloudTrail is Enabled

1. In your AWS account, go to CloudTrail.
2. In CloudTrail, click **Event history**.
3. In Lookup attributes, choose **EventName=Encrypt**.
4. You should see DSV making an API call to your KMS keys.



Usage



After these steps, everything should work as before, but now the encryption in the backend will use the new keys.

SIEM Audits

DSV can send audit records to four different types of listeners: **syslog**, **cef**, **splunk** and **json**. Each type have a corresponding format for log records and supports one or more transport protocols.

To configure a new SIEM endpoint, use the command:

```
dsv siem create
```

To show all available management commands for SIEM endpoints in DSV, run:

```
dsv siem --help
```

> **Note:** For every audit action, DSV will try twice to reach the endpoint. If the endpoint is unresponsive after **ten actions and retries**, DSV will deregister the endpoint and mark it as `failed`. When viewed with `dsv siem read <name>`, deregistered endpoints will have the field and value: `"failed": true`. The endpoint must be recreated or updated to be used again.

Syslog

- Message format: syslog
- Transport protocols: UDP, TCP, TLS (the minimum TLS 1.2 is used)

Syslog messages must be in RFC 5424-compliant form. DSV will truncate messages over 64KB in length.

Usage

Syslog	Audit
Timestamp	RFC3339 format
Priority	191
Version	1
Hostname	DSV URL (e.g., example.secretsvaultcloud.com)
MsgID	id
Appname	DSV
Message	usertoken + audit message
StructuredData	all other audit fields

Note: A user-specific token, generated by user, is inserted into each message to identify the user.

Sample syslog output

```
<191>1 2020-06-02T14:53:48Z example.secretsvaultcloud.com DSV - - [1 action=POST
created=2020-06-02T14:51:36.519620577Z ipaddress=10.10.10.10 path=token
principal=users:username principalItemId=00000000-51ea-4bfa-b272-80b12e43b676
tenant=tenant tenantName=tenantName] abcdef "
```

To start a SIEM configuration workflow, use the command:

```
dsv siem create
```

Option	Description
Name	required, from 3 to 50 characters long configuration name
Type	required, select 'syslog'
Protocol	required, select one of 'tcp', 'udp' or 'tls'
Host	required, domain name or an IP address
Port	required, port number in range [0..65535]
Authentication method	required, select 'token'

Usage

Option	Description
Authentication	required, type a token that will be added to the begining of a syslog message
Logging format	required, select 'rfc5424'
Route through DSV engine	required, specify if SIEM messages should be sent through an engine pool to deliver to a service behind a firewall
Engine pool	string, specify which pool to use if previous question was answered affirmatively

Sample Values

```
{
  "siemType": "syslog",
  "name": "syslogtest",
  "host": "54.210.93.200",
  "port": 8000,
  "protocol": "tls",
  "authMethod": "token",
  "auth": "abcdef",
  "loggingFormat": "rfc5424"
}
```

Common Event Format (CEF)

- Message format: CEF
- Transport protocols: UDP, TCP, TLS (the minimum TLS 1.2 is used)

CEF	DSV Audit	description
Version	0	constant
Device Vendor	delinea	constant
Device Product	dsv	constant
Device Version	-	unused by dsv
Signature ID	id	audit field
Name	action	audit field

Usage

CEF	DSV Audit	description
Severity	status	200 -> 0 400 -> 1 401 -> 7 403 -> 7 404 -> 0 500 -> 0 anything else -> _
Extension		all other audit fields

Sample CEF output

```
CEF:0|delinea|dsv|-|b40e07d3-6fb9-41e8-9816-356de992b8fa|POST|0|action=POST created:2020-06-02T17:52:30.841020649Z id=b40e07d3-6fb9-41e8-9816-356de992b8fa ipaddress=10.10.10.10 message=login succeeded path=token principal=users:username principalItemId=f18b5bda-51ea-4bfa-b272-80b12e43b676 status=200 tenant=tenatID tenantName=tenantName
```

To start a SIEM configuration workflow, use the command:

```
dsv siem create
```

Option	Description
Name	required, from 3 to 50 characters long configuration name
Type	required, select 'cef'
Protocol	required, select one of 'tcp', 'udp' or 'tls'
Host	required, domain name or an IP address
Port	required, port number in range [0..65535]
Authentication method	required, select 'token'
Authentication	required, but not used for 'cef' type
Logging format	required, select 'cef'
Route through DSV engine	required, specify if SIEM messages should be sent through an engine pool to deliver to a service behind a firewall
Engine pool	string, specify which pool to use if previous question was answered affirmatively

Usage

Sample Values

```
{
  "siemType": "cef",
  "name": "syslogtest",
  "host": "34.210.93.200",
  "port": 8678,
  "protocol": "udp",
  "authMethod": "token",
  "auth": "abcdef",
  "loggingFormat": "cef"
}
```

JSON

- Message format: JSON
- Transport protocols: UDP, TCP, HTTP, HTTPS

DSV will send raw JSON audit via configure transport.

Sample JSON output

```
{"action":"POST","created":"2020-06-02T17:52:30.841020649Z","id":"b40e07d3-6fb9-41e8-9816-
356de992b8fa","ipaddress":"10.10.10.10","message":"login
succeeded","path":"token","principal":"users:user","principalItemId":"f18b5bda-51ea-4bfa-
b272-80b12e43b676","status":"200","tenant":"tenat","tenantName":"tenantName"}
```

To start a SIEM configuration workflow, use the command:

```
dsv siem create
```

Option	Description
Name	required, from 3 to 50 characters long configuration name
Type	required, select 'json'
Protocol	required, select one of 'tcp', 'udp', 'http' or 'https'
Host	required, domain name or an IP address
Port	required, port number in range [0..65535]
Endpoint	optional, used only for 'http' or 'https' transport to build an URL as `http[s]://<host>:<port>/<endpoint>`

Usage

Option	Description
Authentication method	required, select 'token'
Authentication	required, not used for 'tcp', 'udp' and added as 'Authorization' header for 'http' and 'https'
Logging format	required, select 'json'
Route through DSV engine	required, specify if SIEM messages should be sent through an engine pool to deliver to a service behind a firewall
Engine pool	string, specify which pool to use if previous question was answered affirmatively

Sample Values

```
{
  "siemType": "json",
  "name": "syslogtest",
  "host": "34.210.93.200",
  "port": 443,
  "protocol": "https",
  "authMethod": "token",
  "auth": "abcdef",
  "loggingFormat": "json"
}
```

Splunk

- Message format: JSON in the format `{"event":{ <audit fields> }}`
- Transport protocols: HTTPS

To start a SIEM configuration workflow, use the command:

```
dsv siem create
```

Option	Description
Name	required, from 3 to 50 characters long configuration name
Type	required, select 'splunk'
Protocol	required, select 'https'
Host	required, domain name or an IP address

Option	Description
Port	required, port number in range [0..65535]
Endpoint	optional, used to build an URL as `https://<host>:<port>/<endpoint>`
Authentication method	required, select 'token'
Authentication	required, a token added as 'Authorization' header to each request
Logging format	required, select 'json'
Route through DSV engine	required, specify if SIEM messages should be sent through an engine pool to deliver to a service behind a firewall
Engine pool	string, specify which pool to use if previous question was answered affirmatively

Sample Configuration

```
{
  "siemType": "splunk",
  "name": "SplunkProd",
  "host": "instance.splunkcloud.co",
  "endpoint": "services/collector/event",
  "port": 8088,
  "protocol": "https",
  "authMethod": "token",
  "auth": "Splunk 84ba1399-87f2-000g-9b49-797ae7935244",
  "loggingFormat": "json"
}
```

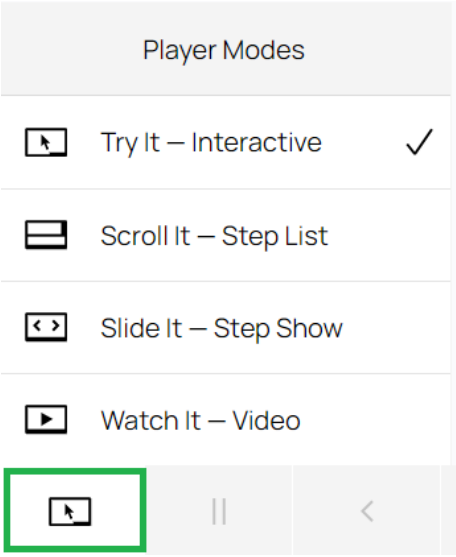
Tutorials

The Tutorials section gives you a variety of DSV use cases and edge cases, as well as deep technical concepts. You can follow each section in any order to complete them successfully.

- Administration and Configuration Video Tutorials
- Policy Tutorial
- Use DSV With Direnv
- Azure Dynamic Secrets

Administration and Configuration Tutorials

Note: You have different options for interacting with each tutorial. You can leave the player in Interactive mode, or switch to a Step List, a Step Show or a video. By default the **Try it - Interactive** mode will be chosen when opening a video. You can access the different modes by selecting the button highlighted in green below:



Policy Tutorial

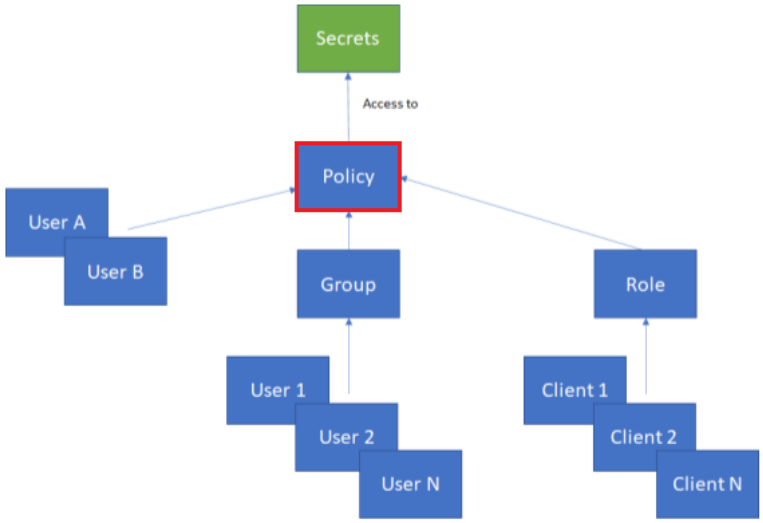
This tutorial addresses a use case in which the initial DSV admin wishes to:

- Delegate resource permissions to one admin team and three separate DevOps teams.
- Give each team of three users access to separate secret paths.
- Assign one person from each team rights to create roles and policies for their teammates.

Policy Structure

Policies are the single source of all permissions in DSV. A policy contains a list of permissions that are then delegated to groups, roles, and/or individual users.

The following image demonstrates the three methods that apply policies to users:

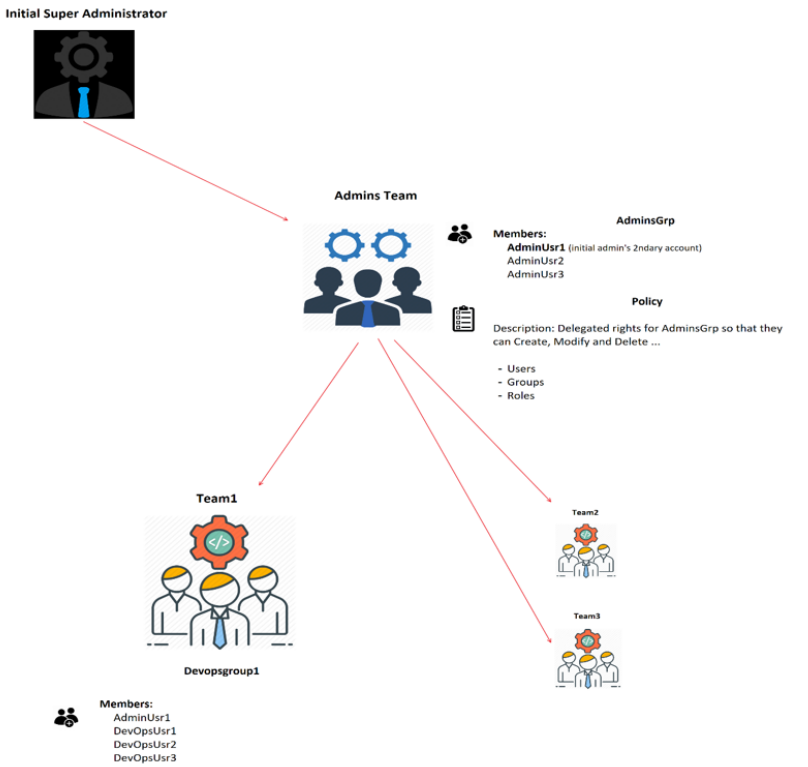


Least Privilege Approach

In this tutorial, we begin as the **Initial Super Administrator** (typically your "thy-one" account). The **Initial Super Administrator** account has full, unrestricted access to all of DSV. It is a best practice to follow the Least-Privilege Administrative Model and only use the Super Administrator account when absolutely necessary.

To avoid using this account, we will:

1. Use the **Super Administrator** account to create new Users.
2. Create a group called "adminsgroup".
3. Assign the new users to the "adminsgroup" group.
4. Create a policy giving the group administrative permissions.
5. Log out of the Super Administrator account.
6. Log in as one of the new users to complete the remaining administrative tasks.



Policy1

Description: Delegated Rights so that DevopsGrp1 can create Secrets on path ***"secrets:servers:us-west:devopsgrp1secrets:<.*>"***.

Policy2

Description: Delegated Rights so that DevopsUsr1 can create Policies on the path ***"secrets:servers:us-west:devopsgrp1secrets_<.*>"***

Policy3

Description: Delegated Rights so that Devopsusr1 can create Roles by the name ***"devopsgrp1-roles<.*>"***

Create Users, Groups and Permissions

First, as the Super Admin, we will create and delegate permissions to the administrators. We will create three administrators and give them permissions to manage Users, Groups, and Roles in DSV.

Note: The following example uses placeholder usernames and passwords. Substitute these values to meet your organization's requirements.

1. Create administrators using the command and flags:

   INPUT:

```
dsv user create --username adminusr1 --password Password1
```

OUTPUT:

```
{
"created": "2021-04-30T14:14:10Z",
"createdBy": "users:thy-one:superadmin@organization.com",
"displayName": "",
"externalId": "",
"id": "dc677f9c-*******-238f6d04e137",
"lastModified": "2021-04-30T14:14:10Z",
"lastModifiedBy": "users:thy-one:superadmin@organization.com",
"provider": "",
"userName": "adminusr1",
"version": "0"
}
```

2. Repeat the command for adminusr2 and adminusr3.

INPUT:

```
dsv user create --username adminusr2 --password Password2
```

OUTPUT:

```
{
"created": "2021-04-30T14:14:10Z",
"createdBy": "users:thy-one:superadmin@organization.com",
"displayName": "",
"externalId": "",
"id": "dc677f9c-*******-238f6d04e137",
"lastModified": "2021-04-30T14:14:10Z",
"lastModifiedBy": "users:thy-one:superadmin@organization.com",
"provider": "",
"userName": "adminusr2",
"version": "0"
}
```

INPUT:

```
dsv user create --username adminusr3 --password Password3
```

OUTPUT:

```
{
"created": "2021-04-30T14:14:10Z",
```

```
"createdBy": "users:thy-one:superadmin@organization.com",
"displayName": "",
"externalId": "",
"id": "dc677f9c-*******-238f6d04e137",
"lastModified": "2021-04-30T14:14:10Z",
"lastModifiedBy": "users:thy-one:superadmin@organization.com",
"provider": "",
"userName": "adminusr3",
"version": "0"
}
```

3. Create the admins group and add the new administrators as members.

   INPUT:

   ```
   dsv group create --group-name adminsgroup --members adminusr1,adminusr2,adminusr
   ```

   OUTPUT:

   ```
   {
   "addedMemberNames": ["adminusr2", "adminusr1", "adminusr3"],
   "errors": {}
   }
   ```

4. Give the adminsgroup permission to create, modify and delete Users, Groups, Roles, Policies, and Secrets on the path `secrets:servers:us-west:<.*>`.

   Note: Permissions *could* be assigned directly to the three users. Assigning permissions to the **group** allows for any additional admin Users to inherit permissions in a single step.

5. Open the configuration text file using the command: `dsv config edit`.
6. Copy and paste the adminsgroup permission data outlined in the red box below. Make sure it is placed after the `groups:<adminsgroup>`value and includes the preceding comma.

```
{
"created": "2020-04-06T12:34:57Z",
"createdBy": "system",
"lastModified": "2021-04-13T19:05:33Z",
"lastModifiedBy": "users:thy-one:superadmin@organization.com",
"permissionDocument": [
  {
    "actions": ["<.*>"],
    "conditions": {},
    "description": "Default Admin Permissions",
    "effect": "allow",
    "id": "bq5i3******po2j2g",
    "meta": null,
    "resources": ["<.*>"],
```

```
    "subjects": [
        "users:<users:thy-one:superadmin@organization.com>"
    ]
  },
  {
    "actions": ["<.*>"],
    "conditions": {},
    "description": "Default Deny Home Permissions",
    "effect": "deny",
    "id": "bskn71nq4h6s72mn0sc0",
    "meta": null,
    "resources": ["home:<.*>"],
    "subjects": [
    "users:<users:thy-one:superadmin@organization.com>"
    ]
  }
```

```
{
"actions": ["<.*>"],
"conditions": {},
"description": "Admin Permissions US-WEST",
"effect": "allow",
"meta": null,
"resources":
["users:<.*>","groups:<.*>","roles:<.*>","clients:<.*>","config:policies:secrets:serve
rs:us-west:<.*>","config:policies:roles:devopsgrp1<.*>","secrets:servers:us-west:<.*>"
  ],
"subjects": ["groups:<adminsgroup>"]
}
```

Initialize the New Admin Account

Once we have created the new admin users, put them into a new admin group, and written a policy giving them permissions, we can log out of the Super Administrator account and use one of the new administrator accounts to continue with the setup. Switching to an account with fewer permissions will help us adhere to the Least-Privilege Administrative Model.

1. Begin initialization with the command: `dsv init`.
2. Select `[o] overwrite the config`. This will replace the current default profile, the superadmin, with the adminusr1 account.
3. Enter your tenant name.
4. Choose your domain.
5. Select a store type.
6. Select a cache strategy for secrets.
7. For the auth type, choose `(1) Password (local user)`.
8. Once initialization is complete, confirm that you are logged in as adminusr1 with the command: `dsv whoami`.
9. The CLI should return: `users:adminusr1`

Delegate Secret Management Rights to DevOps Team1

Now that the administrators have been created and delegated permissions, we can start creating the users for the DevOps teams.

1. Create the three DevOps users.

   INPUT:

   ```
   dsv user create --username devopsusr1 --password Password1*
   ```

   OUTPUT:

   ```
   {
   "created": "2021-04-30T15:30:45Z",
   "createdBy": "users:adminusr1",
   "displayName": "",
   "externalId": "",
   "id": "44f238b5-b657-*****-4defb1d9b4cd",
   "lastModified": "2021-04-30T15:30:45Z",
   "lastModifiedBy": "users:adminusr1",
   "provider": "",
   "userName": "devopsusr1",
   "version": "0"
   }
   ```

   INPUT:

   ```
   dsv user create --username devopsusr2 --password Password1*
   ```

   OUTPUT:

   ```
   {
   "created": "2021-04-30T15:30:45Z",
   "createdBy": "users:adminusr1",
   "displayName": "",
   "externalId": "",
   "id": "44f238b5-b657-*****-4defb1d9b4cd",
   "lastModified": "2021-04-30T15:30:45Z",
   "lastModifiedBy": "users:adminusr1",
   "provider": "",
   "userName": "devopsusr2",
   "version": "0"
   }
   ```

   INPUT:

   ```
   dsv user create --username devopsusr3 --password Password1*
   ```

OUTPUT:

```
{
"created": "2021-04-30T15:30:45Z",
"createdBy": "users:adminusr1",
"displayName": "",
"externalId": "",
"id": "44f238b5-b657-*****-4defb1d9b4cd",
"lastModified": "2021-04-30T15:30:45Z",
"lastModifiedBy": "users:adminusr1",
"provider": "",
"userName": "devopsusr3",
"version": "0"
}
```

2. Create the DevOps group. In the same input, we will also add the three DevOps users and the admin user to the group.

   INPUT:

```
dsv group create --group-name devopsgroup1 --members
devopsusr1,devopsusr2,devopsusr3,adminusr1
```

   OUTPUT:

```
{
"addedMemberNames": ["devopsusr1", "devopsusr3", "devopsusr2","adminusr1"],
"errors": {}
}
```

3. Give the new group (devopsgroup1) access to the path `servers:us-west:devopsgroup1secrets:<*>`. This gives all members of DevOps Team 1 full rights to manage secrets on the path.

   INPUT:

```
dsv policy create --path secrets:servers:us-west:devopsgroup1secrets --subjects
groups:devopsgroup1 --actions create,read,update,delete --resources "secrets:servers:us-
west:devopsgroup1secrets:<.*>" --desc "Devopsgroup1 Secret Management Permissions"
```

   OUTPUT:

```
{
"created": "2021-04-30T15:36:08Z",
"createdBy": "users:adminusr1",
"id": "e5c9f3de-f74b-4d1f-a984-90e31cb2e131",
"lastModified": "2021-04-30T15:36:08Z",
"lastModifiedBy": "users:adminusr1",
"path": "secrets:servers:us-west:devopsgrp1secrets",
```

```
"permissionDocument": [
    {
        "actions": ["create", "read", "update", "delete"],
        "conditions": {},
        "description": "Devopsgrp1 Secret Management Permissions",
        "effect": "allow",
        "id": "c2627q72inos72lhq18g",
        "meta": null,
        "resources": ["secrets:servers:us-west:devopsgrp1secrets:<.*>"],
        "subjects": ["groups:devopsgroup1"]
    }
    ],
"version": "0"
}
```

4. Deny devopsusr3 the rights to create, delete, and update secrets on the path `servers:us-west:devopsgroup1secrets:<*>`. Open the policy file with the command:

```
dsv policy edit --path secrets:servers:us-west:devopsgroup1secrets
```

5. Add the data outlined in the red box below:

```
{
"created": "2021-04-13T13:34:43Z",
"createdBy": "users:thy-one:superadmin@organization.com",
"id": "2d56bf8a-99a7-4a3e-9a30-db5596208480",
"lastModified": "2021-04-13T13:34:43Z",
"lastModifiedBy": "users:thy-one:superadmin@organization.com",
"path": "secrets:servers:us-west:devopsgrp1secrets",
"permissionDocument": [
  {
    "actions": ["create", "read", "update", "delete"],
    "conditions": {},
    "description": "Devopsgrp1 Secret Management Permissions",
    "effect": "allow",
    "id": "c1qprsq5fkhs72p14v7g",
    "meta": null,
    "resources": ["secrets:servers:us-west:devopsgrp1secrets:<.*>"],
    "subjects": ["groups:devopsgroup1"]
  }
```

```
{
  "actions": ["create", "update", "delete"],
  "conditions": {},
  "description": "Devopsusr3 Secret Management Permissions",
  "effect": "deny",
  "meta": null,
  "resources": ["secrets:servers:us-west:devopsgrp1secrets:<.*>"],
  "subjects": ["users:devopsusr3"]
```

```
  }
  ],
"version": "0"
}
```

Test the DevOps Team Permissions to Read Secrets

1. Create a secret on the path `secrets:servers:us-west:devopsgroup1secrets`.

   INPUT:

   ```
   dsv secret create secrets:servers:us-west:devopsgroup1secrets:test  --data "
   {\"username\":\"secretuser\",\"password\":\"passwordtext123\"}"
   ```

   OUTPUT:

   ```
   {
   "attributes": {},
   "created": "2021-04-30T15:40:14Z",
   "createdBy": "users:adminusr1",
   "data": {
   "password": "passwordtext123",
   "username": "secretuser"
   },
   "description": "",
   "id": "76b872be-fb5a-4849-b8c7-f8bea3b01896",
   "lastModified": "2021-04-30T15:40:14Z",
   "lastModifiedBy": "users:adminusr1",
   "path": "servers:us-west:devopsgrp1secrets:test",
   "version": "0"
   }
   ```

2. Create another secret on the path `secrets:servers:us-west:devopsgroup1secrets:test`.

   INPUT:

   ```
   dsv secret create secrets:servers:us-west:devopsgrp1secrets  --data "
   {\"username\":\"secretuser\",\"password\":\"passwordtext123\"}"
   ```

   OUTPUT:

   ```
   {
   "attributes": {},
   "created": "2021-04-30T15:40:14Z",
   "createdBy": "users:adminusr1",
   "data": {
           "password": "passwordtext123",
           "username": "secretuser"
           },
   ```

```
"description": "",
"id": "76b872be-fb5a-4849-b8c7-f8bea3b01896",
"lastModified": "2021-04-30T15:40:14Z",
"lastModifiedBy": "users:adminusr1",
"path": "servers:us-west:devopsgrp1secrets:test",
"version": "0"
}
```

3. Initialize with the devopsusr1 account. In order to test the permissions granted to devopsusr1 we need to initialize the account. This will create a new profile for devopsusr1 in your config file. Be sure to choose auth type "1" as this is a local user.
4. Enter the command `dsv init`.
5. Choose `[a] add a new profile to the config`.
6. Enter the profile name: `devopsusr1`.
7. Initialize `devopsusr2` and `devopsusr3` using the same sequence. Once the profiles are created, we will be able to run single commands as `devopsusr1` and `devopsusr3` in the next step.
8. Read the secrets with the profile of `devopsusr1`. This profile should have the permissions to read the secret under `test` while not having the permissions to read the secret under `devopsgrp1secrets`.

   INPUT:

   ```
   dsv secret read secrets:servers:us-west:devopsgrp1secrets:test --profile devopsusr1
   ```

   OUTPUT:

   ```
   {
   "attributes": {},
   "created": "2021-04-30T15:40:14Z",
   "createdBy": "users:adminusr1",
   "data": {
             "password": "passwordtext123",
             "username": "secretuser"
           },
   "description": "",
   "id": "76b872be-fb5a-4849-b8c7-f8bea3b01896",
   "lastModified": "2021-04-30T15:40:14Z",
   "lastModifiedBy": "users:adminusr1",
   "path": "us-west:devopsgrp1secrets:test",
   "version": "0"
   }
   ```

   INPUT:

   ```
   dsv secret read secrets:servers:us-west:devopsgrp1secrets --profile devopsusr1
   ```

OUTPUT:

```
{
"message": "Invalid permissions"
}
```

9. Attempt to create a secret with the profile of `devopsusr3`. This profile should not have the rights to create a secret on that path.

   INPUT:

```
dsv secret create secrets:servers:us-west:devopsgrp1secrets:test2  --data "
{\"username\":\"secretuser2\",\"password\":\"passwordtext123\"}" --profile devopsusr3
```

   OUTPUT:

```
{
"message": "Invalid permissions"
}
```

Grant Groups the Ability to Search Entities via List Privileges

In the previous section, we verified that the members of devopsgroup1 can only manage the secrets on the path `servers:us-west:devopsgrp1secrets:<.*>` and restricted a single member of that group, devopsusr3, to only be able to read secrets on that path.

Now let's say the members needed to see the non-sensitive information (for example, description, path, create) of secrets on a different path. We can do this by creating permissions on the root policy to grant List Privileges for all secrets in DSV to devopsgroup1. List Privileges can also be granted for users, groups and roles.

In this example, we will only grant the permission for secrets by:

- Editing the config using the command `dsv config edit --profile thyone`
- Adding the section outlined in red below to the set of permissions that currently exist on the config policy:

```
{
  "created": "2020-04-06T12:34:57Z",
  "createdBy": "system",
  "lastModified": "2021-04-30T14:34:09Z",
  "lastModifiedBy": "users:thy-one:superadmin@organization.com",
  "permissionDocument": [
    {
      "actions": ["<.*>"],
      "conditions": {},
      "description": "Default Admin Permissions",
      "effect": "allow",
      "id": "bq5i3seothfc72po2j2g",
      "meta": null,
      "resources": ["<.*>"],
```

```
      "subjects": [
        "users:<users:thy-one:superadmin@organization.com>"
      ]
    },
    {
      "actions": ["<.*>"],
      "conditions": {},
      "description": "Default Deny Home Permissions",
      "effect": "deny",
      "id": "bskn71nq4h6s72mn0sc0",
      "meta": null,
      "resources": ["home:<.*>"],
      "subjects": [
        "users:<thy-one:superadmin@organization.com>"
      ]
    },
    {
      "actions": ["<.*>"],
      "conditions": {},
      "description": "Admin Permissions US-WEST",
      "effect": "allow",
      "id": "c261aofnu9hs72pma9t0",
      "meta": null,
      "resources": [
        "users:<.*>",
        "groups:<.*>",
        "roles:<.*>",
        "clients:<.*>",
        "config:policies:secrets:servers:us-west:<.*>",
        "config:policies:roles:devopsgrp1<.*>",
        "secrets:servers:us-west:<.*>"
      ],
      "subjects": ["groups:<adminsgroup>"]
    }

  {
    "actions": ["<list>"],
    "conditions": {},
    "description": "Global List Permissions - Secrets",
    "effect": "allow",
    "meta": null,
    "resources": ["secrets"],
    "subjects": ["groups:<devopsgroup1>"]
  }

  ],
  "tenantName": "dsvtestlab",
  "version": "2"
  }
```

Test the DevOps Team Permissions to Search Resources

Using the profile of devopsusr1, search for the secrets located on the path `servers:us-west:devopsgrp1secrets`. While the devopsusr1 profile was not able to read secrets on this path before, the list permissions allows the user to search for that secret and view its non-sensitive properties.

INPUT:

```
dsv secret search devopsgroup1secrets --profile devopsusr1
```

OUTPUT:


```
{
"cursor": "",
"data": [
  {
    "attributes": {},
    "created": "2021-04-30T15:40:14Z",
    "createdBy": "users:adminusr1",
    "description": "",
    "id": "76b872be-fb5a-4849-b8c7-f8bea3b01896",
    "lastModified": "2021-04-30T15:40:14Z",
    "lastModifiedBy": "users:adminusr1",
    "path": "servers:us-west:devopsgrp1secrets:test",
    "version": "0"
  },
  {
    "attributes": {},
    "created": "2021-04-30T17:46:23Z",
    "createdBy": "users:adminusr1",
    "description": "",
    "id": "90c728d1-7584-49d4-86a9-89fa4ca8daa0",
    "lastModified": "2021-04-30T17:46:23Z",
    "lastModifiedBy": "users:adminusr1",
    "path": "servers:us-west:devopsgrp1secrets",
    "version": "0"
  }
],
"length": 2,
"limit": 25,
"sort": ""
}
```


Delegate Rights to Manage Policies to a DevOps Team Member

Give devopsusr1 the rights to create, read, update, and delete polices on the path `secrets:servers:us-west:devopsgroup1secrets_<.*>`. The permissions will be applied **directly** to the user as opposed to a group. We will also give devopsgroup1 read access to any policies created by devopsusr1. Edit the policy again by adding the red-boxed json snippet below.

1. Open the policy using the command:

```
dsv policy edit --path secrets:servers:us-west:devopsgroup1secrets
```

2. Add the red-boxed JSON data to the policy:

```
{
  "created": "2021-04-13T13:34:43Z",
  "createdBy": "users:thy-one:superadmin@organization.com",
  "id": "2d56bf8a-99a7-4a3e-9a30-db5596208480",
  "lastModified": "2021-04-13T13:34:43Z",
  "lastModifiedBy": "users:thy-one:superadmin@organization.com",
  "path": "secrets:servers:us-west:devopsgrp1secrets",
  "permissionDocument": [
{
      "actions": ["create", "read", "update", "delete"],
      "conditions": {},
      "description": "Devopsgrp1 Secret Management Permissions",
      "effect": "allow",
      "id": "c2627q72inos72lhq18g",
      "meta": null,
      "resources": ["secrets:servers:us-west:devopsgrp1secrets:<.*>"],
      "subjects": ["groups:devopsgroup1"]
},
{
      "actions": [ "read"],
      "conditions": {},
      "description": "Devopsusr3 Secret Management Permissions",
      "effect": "deny",
      "id": "c2629jn2inos72lhq190",
      "meta": null,
      "resources": ["secrets:servers:us-west:devopsgrp1secrets:<.*>"],
      "subjects": ["users:devopsusr3"]
    },
```

```
{
      "actions": ["create", "read", "update", "delete"],
      "conditions": {},
      "description": "Devops Team1 Policy Management Permissions",
      "effect": "allow",
      "meta": null,
      "resources": ["config:policies:secrets:servers:us-
west:devopsgrp1secrets:devopsgrp1policy_<.*>"],
      "subjects": ["users:devopsusr1"]
    },
    {
      "actions": ["read"],
      "conditions": {},
      "description": "Devops Team1 Policy Read Permissions",
      "effect": "allow",
      "meta": null,
      "resources": ["config:policies:secrets:servers:us-
```

```
west:devopsgrp1secrets:devopsgrp1policy_<.*>"],
      "subjects": ["groups:devopsgroup1"]
    }
```

```
  ],
    "version": "2"
  }
```

Test DevOpsUsr1's Permission to Create Policies

Create a policy using the profile devopsusr1, then read the policy using the profile devopsusr2. The first attempt to create a policy should fail because devopsusr1 is not permitted to create on the path `testfailure`. The 2nd attempt will succeed. This policy grants devopsgroup1 full privileges to manage secrets beyond the path `servers:us-west:devopsgrp1secrets:devopsgrp1policy_1`.

INPUT:

```
dsv policy create --path secrets:servers:us-west:devopsgrp1secrets:testfailure --subjects
groups:devopsgroup1 --actions create,read,update,delete --profile devopsusr1
```

OUTPUT:

```
{
  "message": "Invalid permissions"
}
```

INPUT:

```
dsv policy create --path secrets:servers:us-west:devopsgrp1secrets:devopsgrp1policy_1 --
subjects groups:devopsgroup1 --actions create,read,update,delete --desc "Devopsgroup1
User-Created Policy1" --profile devopsusr1
```

OUTPUT:

```
{
  "created": "2021-04-30T18:06:17Z",
  "createdBy": "users:devopsusr1",
  "id": "bc3c38d6-c7cc-49b4-817a-f98b6c409974",
  "lastModified": "2021-04-30T18:06:17Z",
  "lastModifiedBy": "users:devopsusr1",
  "path": "secrets:servers:us-west:devopsgrp1secrets:devopsgrp1policy_1",
  "permissionDocument": [
    {
      "actions": ["create", "read", "update", "delete"],
      "conditions": {},
      "description": "Devopsgroup1 User-Created Policy1",
      "effect": "allow",
```

```
      "id": "c264e69ehf7c72g0ddg0",
      "meta": null,
      "resources": [
        "secrets:servers:us-west:devopsgrp1secrets:devopsgrp1policy_1:<.*>"
      ],
      "subjects": ["groups:devopsgroup1"]
    }
  ],
  "version": "0"
}
```

Delegate Rights to "Create Roles" to a DevOps Team Member

Give devopsusr1 the rights to create, read, and assign roles by the name `devopsgrp1-roles<.*>`. This user will be the only member of the group that can create roles. Note that the resource must be named appropriately otherwise the attempt to create will fail. This step will make it easier to audit the creation of policies and provide user accountability.

INPUT:

```
dsv policy create --path roles:devopsgrp1_role --subjects users:devopsusr1 --desc
"Devopsgrp1 Role Assignment Permissions" --resources "roles:devopsgrp1_role<.*>" --actions
create,assign,read
```

OUTPUT:

```
{
  "created": "2021-04-30T18:09:42Z",
  "createdBy": "users:adminusr1",
  "id": "9f46574a-41cd-4d1b-a03b-d91740aa0321",
  "lastModified": "2021-04-30T18:09:42Z",
  "lastModifiedBy": "users:adminusr1",
  "path": "roles:devopsgrp1_role",
  "permissionDocument": [
    {
      "actions": ["create", "assign", "read"],
      "conditions": {},
      "description": "Devopsgrp1 Role Assignment Permissions",
      "effect": "allow",
      "id": "c264fphehf7c72g0ddgg",
      "meta": null,
      "resources": ["roles:devopsgrp1_role<.*>"],
      "subjects": ["users:devopsusr1"]
    }
  ],
  "version": "0"
}
```

We will also give `devopsgroup1` read permissions for any role created by `devopsusr1`:

1. Open the policy using the command: `dsv policy edit --path roles:devopsgrp1_role`
2. Edit the policy we have just created by adding the red-boxed json snippet below:

```
{
  "created": "2021-04-22T15:18:02Z",
  "createdBy": "users:adminusr1",
  "id": "5c8b225f-89d6-4f4e-9c67-03b333a9ff4d",
  "lastModified": "2021-04-22T15:18:02Z",
  "lastModifiedBy": "users:adminusr1",
  "path": "roles:devopsgrp1_role",
  "permissionDocument": [
    {
      "actions": ["create", "assign", "read"],
      "conditions": {},
      "description": "Devopsgrp1 Role Assignment Permissions",
      "effect": "allow",
      "id": "c20p7alfo4sc72ggua4g",
      "meta": null,
      "resources": ["roles:devopsgrp1_role<.*>"],
      "subjects": ["users:devopsusr1"]
    }
```

```
{
      "actions": ["read"],
      "conditions": {},
      "description": "Devopsgrp1 Role Read Permissions",
      "effect": "allow",
      "meta": null,
      "resources": ["roles:devopsgrp1_role<.*>"],
      "subjects": ["groups:devopsgroup1"]
    }
```

```
],
  "version": "0"
}
```

3. Test `devopsusr1`'s permission to create roles:
   a. Attempt to create a role using a name outside of what devopsusr1 has the permissions to create:

      INPUT:

      ```
      dsv role create --name devopsgrp1-roletestfailure --profile devopsusr1
      ```

      OUTPUT:

      ```
      {"message": "Invalid permissions"}
      ```

   b. Now perform a test within the user's permissions:

INPUT:

```
dsv role create --name devopsgrp1_role1 --profile devopsusr1
```

OUTPUT:


```
{
"created": "2021-04-30T18:18:03Z",
"createdBy": "users:devopsusr1",
"description": "",
"externalId": "",
"groups": null,
"id": "73b0073c-b695-43fe-885c-932c8b9a5d8f",
"lastModified": "2021-04-30T18:18:03Z",
"lastModifiedBy": "users:devopsusr1",
"name": "devopsgrp1_role1",
"provider": "",
"version": "0"
}
```


Create DevOpsTeam1's Client Credentials for an Application

Using the role that we just created with the devopsusr1 `devopsgrp1-role1`, we will create client credentials. The credentials will be associated with the role and inherit the permissions that the role has been delegated.

1. Add the role to the `devopsgrp1policy_1` Policy. We will use the `update` flag to add the role as an additional subject of the policy:

   INPUT:

   ```
   dsv policy update --path secrets:servers:us-west:devopsgrp1secrets:devopsgrp1policy_1 --
   subjects groups:devopsgroup1,roles:devopsgrp1_role1 --actions create,read,update,delete
   --desc "Devopsgrp1 User-Created Polciy1"
   ```

   OUTPUT:


```
{
"created": "2021-04-30T18:06:17Z",
"createdBy": "users:devopsusr1",
"id": "bc3c38d6-c7cc-49b4-817a-f98b6c409974",
"lastModified": "2021-04-30T18:20:24Z",
"lastModifiedBy": "users:devopsusr1", "path": "secrets:servers:us-
west:devopsgrp1secrets:devopsgrp1policy_1", "permissionDocument": [
  {
     "actions": ["create", "read", "update", "delete"],
     "conditions": {},
     "description": "Devopsgrp1 User-Created Polciy1",
     "effect": "allow",
     "id": "c264kq1ehf7c72g0ddhg",
```

```
        "meta": null,
        "resources": [ "secrets:servers:us-west:devopsgrp1secrets:devopsgrp1policy_1:<.*>"
        ],
        "subjects": ["groups:devopsgroup1", "roles:devopsgrp1_role1"]
      }
    ],
    "version": "1"
    }
```

2. Create the DevOps Team1 client. A Client ID and Client Secret will be provided for the next step:

   INPUT:

```
dsv client create --role devopsgrp1_role
```

   OUTPUT:

```
{
"clientId": "33c2b014-27af-49fa-b4b3-44e8c1cad2b9",
"clientSecret": "1E_uAzxTWbwMjJcfEIP1294pAhp-pkOX5ECqDNZOk8s",
"created": "2021-04-30T18:21:38Z",
"createdBy": "users:adminusr1",
"id": "f131e1fb-bc04-4015-ac8b-0e7ba5c2e20f",
"role": "devopsgrp1_role1",
"url": false
}
```

Test the "Read Secret" Permissions of the DevOpsTeam1's Client Credential

1. Initialize with the client using `dsv init`.
2. Select `[a] add a new profile to the config`, and name your profile `clienttest`.
3. Choose `(2) Client Credential` for the Auth Type.
4. When prompted, provide the Client ID and Client Secret below:

```
Found an existing cli-config located at 'C:\Users\superadmin\.dsv.yml'
Select an option:
   [o] overwrite the config
   [a] add a new profile to the config
   [n] do nothing (default:n) a
Please enter profile name: clienttest
Please enter tenant name: dsvtestlab
Please choose domain:
   (1) secretsvaultcloud.com (default)
   (2) secretsvaultcloud.eu
   (3) secretsvaultcloud.com.au
   (4) secretsvaultcloud.ca
Selection:
Please enter store type:
   (1) File store (default)
   (2) None (no caching)
   (3) Pass (Linux only)
```

```
   (4) Windows Credential Manager (Windows only)
Selection: Please enter directory for file store (default:C:\Users\superadmin\.thy):
Please enter cache strategy for secrets:
   (1) Never (default)
   (2) Server then cache
   (3) Cache then server
   (4) Cache then server, but allow expired cache if server unreachable
Selection:
Please enter auth type:
   (1) Password (local user) (default)
   (2) Client Credential
   (3) Thycotic One (federated)
   (4) AWS IAM (federated)
   (5) Azure (federated)
   (6) GCP (federated)
   (7) OIDC (federated)
Selection: 2
Please enter client id for client auth: 33c2b014-27af-49fa-b4b3-44e8c1cad2b9 Please enter
client secret for client auth: ********************************************
```

5. Create a secret on the path `secrets:servers:us-west:devopsgrp1secrets:devopsgrp1polcicy_1:`.

   INPUT:

   ```
   dsv secret create secrets:servers:us-west:devopsgrp1secrets:devopsgrp1policy_1:test --
   data "{\"username\":\"secretuser\",\"password\":\"passwordtext123\"
   ```

   OUTPUT:

   ```
   {
   "attributes": {},
   "created": "2021-04-30T18:27:49Z",
   "createdBy": "users:adminusr1",
   "data": {
     "password": "passwordtext123",
     "username": "secretuser"
     },
   "description": "",
   "id": "04e203f9-b275-4140-bce5-218b80815c23",
   "lastModified": "2021-04-30T18:27:49Z",
   "lastModifiedBy": "users:adminusr1",
   "path": "servers:us-west:devopsgrp1secrets:devopsgrp1policy_1:test",
   "version": "0"
   }
   ```

6. Read the secret with the profile of the Client Credentials `Clienttest`:

   ```
   dsv secret read secrets:servers:us-west:devopsgrp1secrets:devopsgrp1policy_1:test --
   profile Clienttest
   ```

7. You have successfully delegated permissions to DevOps Team1. Repeat the procedure above for Team 2 and Team 3.

Use DSV With Direnv

`direnv` is a commonly used tool to load environment variables for projects in the Linux/Mac communities.

`direnv` is an extension for your shell. It augments existing shells with a new feature that can load and unload environment variables depending on the current directory.

In this workflow, it's common to load environment variables from your `$HOME/.envrc` (optionally in your `.profile`).

Challenges

Keeping credentials in plain text in your .envrc might be a quick solution, but it's not a secure approach. Removing sensitive values from your `.envrc` or `.env` file can be a great step towards improving your security. Use `dsv` to retrieve secrets on demand or load in your session context.

When saving environment variables, it's common to see secrets set via environment variables in `.envrc` or other formats (`.env`).

```
export GH_TOKEN="plaintexttoken"
export GITHUB_TOKEN="plaintexttoken"
```

Instead, leverage `dsv` to populate your credentials on environment load.

```
export GH_TOKEN="$(dsv-cli secret read --path "core-services:tokens:github-pat:github-pat"
--filter '.data.github-token' --plain --profile mycustomprofilename)"
export GITHUB_TOKEN=$GH_TOKEN
```

Quick Start on Creating a Secret Like This

Using the DSV CLI, you can create a secret like this:

```
rolename="core-services-tokens"
secretpath="core-services:tokens:github-pat"
secretpathclient="clients:${secretpath}"

desc="github token for org, repo, and all blanket usage"


secretkey="github-pat"
secretvalue='{"github-token":">>>> SECRET HERE <<<<"}'
dsv secret create \
  --path "secrets:${secretpath}:${secretkey}" \
  --data "${secretvalue}" \
  --desc "${desc}"
dsv secret read --path "core-services:tokens:github-pat:github-pat" --filter
'.data.github-token' --plain
```

Optionally, create a client credential and use this as an alternative profile that has limited access to only a specific path. This is only needed to set up a different profile based on client credentials instead of your normal DSV login.

```
dsv role create --name "${rolename}"
clientcred=$(dsv client create --role "${rolename}" --plain | jq -c)
```

DSV Tweaks

When you configure DSV, you can further enhance this approach by leveraging the caching setup. Use a longer cache lifecycle to reduce API calls, and improve performance.

Limit Scope Of Secret When Possible

The secret is still in memory when loaded as an environment variable, so while it's more secure than plain text, you can take it even further by investigating using the DSV SDK directly, as well as minimizing the lifespan of the secret.

For example, instead of loading an environment variable that is in scope for all tools and commands, call it as part of your script like this:

```
customcli --param foo \
    --param bar \
    --token $(dsv secret read --path "core-services:tokens:github-pat:github-pat" --filter
'.data.github-token' --plain)
```

In PowerShell, look at Microsoft's documentation about Secret Management With PowerShell

Azure Dynamic Secrets

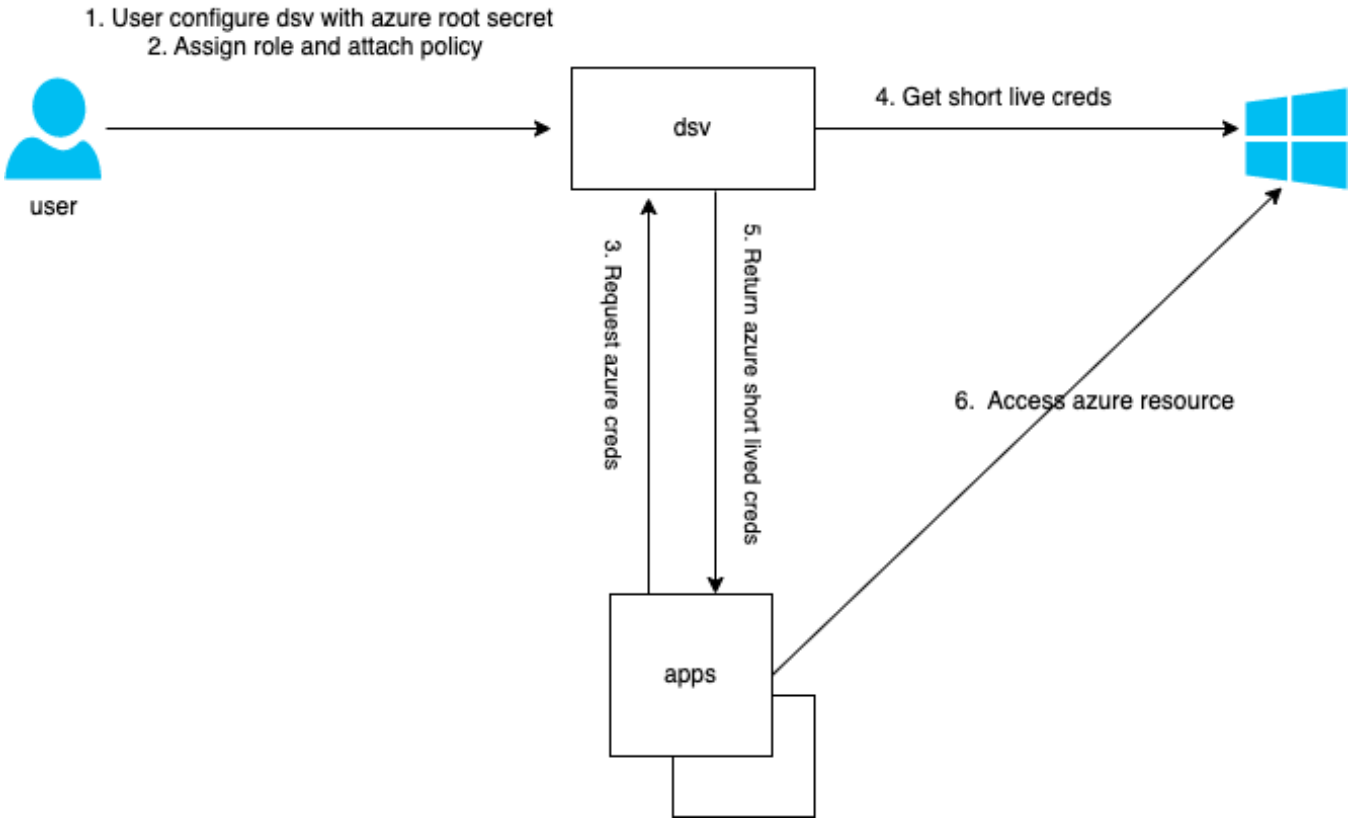
Azure dynamic secrets is revocable, time-limited access and on-demand credentials for azure cloud.

Challenge/Scenarios

To consume Azure services (e.g., Azure Cosmos DB), the application must have valid Azure credentials. Azure uses service principal to authenticate its users. An Azure service principal is a security identity used by user-created apps, services, and automation tools to access specific Azure resources. It only needs to be able to do specific things, unlike a general user identity. It improves security if you only grant it the minimum permissions level needed to perform its management tasks. Any new application that needs to access to these azure resource adds operational overhead as more service principals are required new service principal to access.

Solution

Use DSV dynamic secrets for Azure. This starts with linking a DSV secret to an Azure Service Principal. Then each time you request the secret, it creates a short lived secret to access an Azure hosted service. You can set the TTL for how long those credentials will stay valid.



Benefits

Each app instance can request unique, short-lived credentials. Unique credentials ensures isolated, auditable access and enable revocation of a single client. While short-lived reduces the time frame in which they are valid.

Try for yourself Refer to azure dynamic secret.

DSV Integrations

The following integrations are supported for DevOps Secrets Vault.

- Kubernetes
- Terraform
- Jenkins
- GitHub
- GitLab
- Azure DevOps
- Ansible

- Puppet
- Chef

Kubernetes

Go to GitHub

DSV has two Kubernetes plugins to retrieve secrets.

- Kubernetes sidecar

  The Kubernetes sidecar uses a sidecar for each pod in a cluster, and they all communicate with a single broker pod running in the cluster that caches secrets. Refer to Kubernetes Architecture.

- Kubernetes webhook

  This plugin uses mutating webhooks, and injects the secrets into the cluster's secret data store (etcd) so they can be used globally.

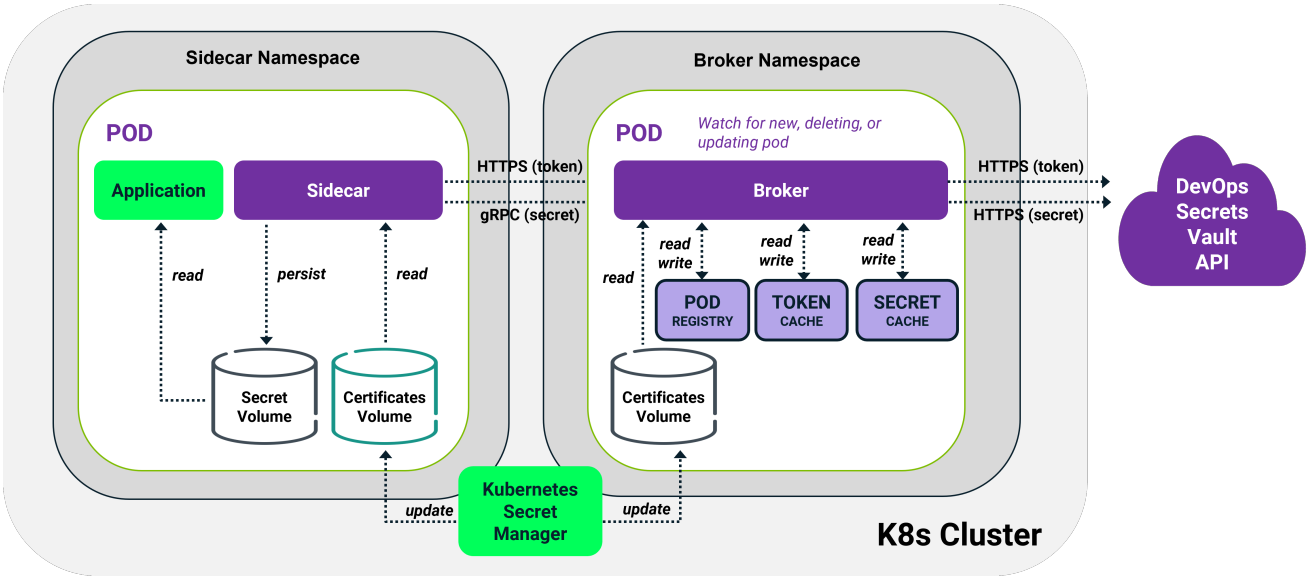
Selecting a Kubernetes plugin

The mutating webhook is the most supported option. It is the most flexible for a large deployment.

If you are already using etcd for secrets, the webhook can be incorporated easily. If however you'd like more granular access, the sidecar keeps secrets inside each pod.

Kubernetes Sidecar Architecture

The illustration shows an example of a Kubernetes architecture implementation.



In studying the diagram, it would be easy to mistakenly conclude that the Kubernetes Secrets Manager is being used to store the pods' secrets, which is not the case. The action of Kubernetes Secrets Manager here is to distribute TLS certificates to secure the connection between the DSV broker and sidecar agent, in cases where this is desirable. In most cases this would be unnecessary since the user cluster will typically be secured already.

If secrets were to be stored in Kubernetes Secrets Manager, they would be universally available in the cluster, contrary to the goal. Instead, with the DSV broker and the volume mount sharing depicted in the diagram, each pod sees only its own secrets, and secrets remain available as long as the pods are healthy.

The sidecar Kubernetes integration to DevOps Secrets Vault consists of several images available from Docker. These illustrate how to build containers incorporating DevOps Secrets Vault functionality. To obtain these images, at your Docker command line, use *docker pull* commands for each image:

```
docker pull thycotic/dsv-k8s-controller:latest
docker pull thycotic/dsv-k8s-client:latest
```

Kubernetes helps coordinate containerized applications across a cluster of machines. DevOps Secrets Vault (DSV) integrates with any existing Kubernetes application deployment. This article, with reference to the example YAML code, explains how you would use the provided client and broker YAML to implement the DSV application with your cluster.

Description of Operations

The example application uses a **broker** and client container deployment with volume mount sharing for pods to access the retrieved secrets. This page includes an example of a *broker.yml* suitable for creation.

Considerations for Sidecars

If the sidecar and broker/controller pods are in the same namespace, no additional actions need to be taken.

If the sidecar and broker/controller pods are in different namespaces, the sidecar needs to know the broker's namespace using the following environment variable in the sidecar configuration.

Broker is in a Different Namespace

```
# Required if broker is in a different namespace
        - name: BROKER_NAMESPACE
          value: brokernamespace
```

Specific Namespace Watched by Broker

By default, the broker/controller watches all pods in the cluster; however, the broker can be configured to only watch pods from a specific namespace using the following environment variable in the broker config.

```
# Optional: Pods Only in this Namespace will be Watched by Broker
        - name: SIDECAR_NAMESPACE
          value: example
```

Introduction to the Client

The client container fetches and periodically updates a configuration file stored at a shared volume. This is defined as a shared volume by the pods within the container (see *example.yml*).

Be sure in your application container to add a volume mount to the shared information, as follows.

```
volumeMounts:
- name: client-volume
    mountPath: /var/secret/
```

For the container running the DSV client, you should define the following as environment variables:

```
env:
- name: REFRESH_TIME
    value: 5s
- name: THY_SECRETS
    value: resources/us-east-1/server1
- name: POD_IP
    valueFrom:
    fieldRef:
        fieldPath: status.podIP
- name: POD_NAME
    valueFrom:
    fieldRef:
        fieldPath: metadata.name
```

`THY_SECRETS` defines the path(s) of the secrets the container uses. This is a list separated by spaces.

Example YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: thycotic-keys
  namespace: default
type: Opaque

---

apiVersion: v1
kind: Deployment
metadata:
  name: secret-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: secret-example
```

```
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: secret-example
      annotations:
        dsv: testtenant
    spec:
      containers:
      - name: bambe-example
        image: <your app image>
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - name: client-volume
          mountPath: /var/secret/
      - name: bambe-client
        image: thycotic/dsv-k8s-client:<tagname>
        imagePullPolicy: IfNotPresent
        env:
        - name: REFRESH_TIME
          value: 5s
        - name: THY_SECRETS
          value: resources/us-east-1/server1
        - name: LOG_LEVEL
          value: error
        - name: POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
          - name: POD_SERVICEACCOUNT
            valueFrom:
              fieldRef:
                fieldPath: spec.serviceAccountName
         volumeMounts:
        - name: client-volume
          mountPath: /var/secret/
          readOnly: false
        - name: secretkey
          mountPath: /tmp/keys
          readOnly: true
      volumes:
```

```
      - name: client-volume
        emptyDir: {}
      - name: secretkey
        secret:
          secretName: thycotic-keys
```

Introduction to the Broker

The Role definition at the beginning of the *broker.yml* file enables the broker pod to execute. The Service descriptions in the *broker.yml* example below are also required as the DSV client uses the name to make internal calls.

In using the *broker.yml* file, be sure to first swap in variable values appropriate to your organization, specifically:

```
spec:
  template:
    spec:
      containers:
        env:
        - name: TENANT
          value: your_tenant_name
        - name: CLIENT_ID
          value: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
        - name: CLIENT_SECRET
          value: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxxxx
```

When the broker is running, it watches for new pods coming online that execute with a specific Annotation, *dsv*. For each such pod, it looks at the value of the **tenant** to be used, and adds the pod to its internal registry.

Kubernetes Plugin Configuration

The Kubernetes plugin provides a means of managing workloads and services for the containers configured for the DevOps Secrets Vault. The following steps are required to configure a Kubernetes plugin.

1. Provide a certificate, an authentication provider, and a corresponding role. Refer to Authentication by Certificate. Save that data. DSV doesn't keep any of these keys. The certificate and key are passed to the broker via the Kubernetes tls secret.
2. Create the Kubernetes `tls` secret.

```
apiVersion: v1
kind: Secretmetadata:  name: dsv-auth-tls-secrets  namespace: sandbox05-pportaltype:
kubernetes.io/tlsdata:  tls.crt: <your client/leaf cert>  tls.key: <your client/leaf
private key>
```

3. Add the volume in broker/controller yaml and mount at /etc/dsv/certs. We chose mounting secrets for automatic updates (https://kubernetes.io/docs/concepts/configuration/secret/#mounted-secrets-are-updated-

automatically).

```
      volumes:
          - name: dsv-tls-secrets          secret:            secretName: dsv-auth-tls-
  secrets


      volumeMounts:
         - name: dsv-tls-secrets         readOnly: true         mountPath: /etc/dsv/certs
```

4. Add the new `ENV` variable to the broker. This tells the broker what type of authentication it uses. name: `AUTH_TYPE` value: `certificate`

The Broker YAML File

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: dsv-service-pod-reader-binding
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: dsv-service-pod-reader-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dsv-service-pod-reader-binding
subjects:
  - kind: ServiceAccount
    name: default
    namespace: default
---
apiVersion: v1
kind: Secret
metadata:
  name: thycotic-keys
  namespace: default
type: Opaque

---

apiVersion: v1
kind: Deployment
metadata:
  name: dsv-broker
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      app: dsv-broker
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: dsv-broker
    spec:
      containers:
      - name: dsv-broker
        image: thycotic/dsv-k8s-controller:<tagname>
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - name: secretkey
            mountPath: /tmp/keys
            readOnly: true
        env:
        - name: REFRESH_TIME
          value: 5m
        - name: THY_API_URL
          value: https://%s.devbambe.com/v1
        - name: TENANT
          value: testtenant
        - name: CLIENT_ID
          value: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
        - name: CLIENT_SECRET
          value: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxxxx
        - name: LOG_LEVEL
          value: debug
      volumes:
        - name: secretkey
          secret:
            secretName: thycotic-keys

---
kind: Service
apiVersion: v1
metadata:
  name: dsv-broker
spec:
  selector:
    app: dsv-broker
  ports:
  - protocol: TCP
    port: 80
    targetPort: 3000
---
```

```
kind: Service
apiVersion: v1
metadata:
  name: dsv-auth
spec:
  selector:
    app: dsv-broker
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 8080
  - name: https
    protocol: TCP
    port: 443
    targetPort: 443
```

This file can also be used locally, for example:

```
kubectl create -f broker.yml
```

Kubernetes Mutating Webhook

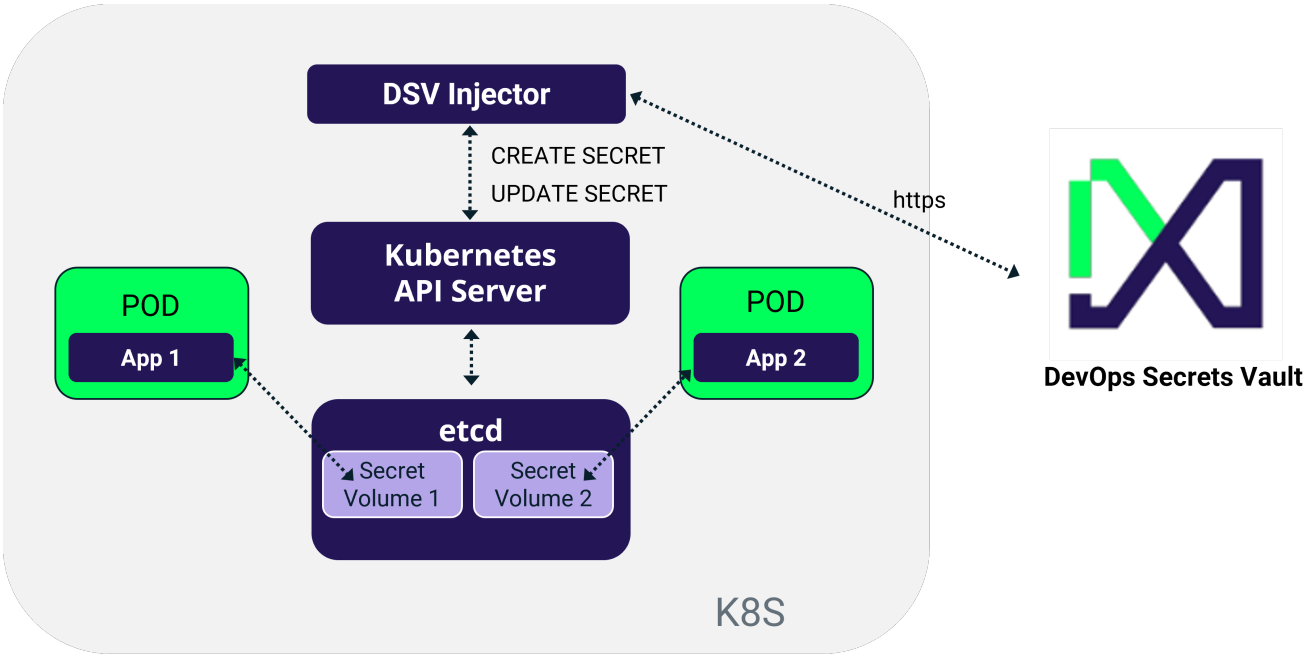
The Kubernetes Mutating Webhook has two parts, the Injector and the Syncer.

The Injector uses a YAML definition that maps secrets in a DSV tenant to variables in the Kubernetes secrets area. It runs when the cluster starts, sets these variables, and populates them with the secrets data from DSV.

Then the Syncer runs as a `cron` task, generally every minute, that updates the Kubernetes environment with updates that happen in DSV.

Architecture

The illustration shows an example of a Kubernetes Mutating Webhook architecture implementation.



Implementing the Kubernetes Mutating Webhook

Tools for implementing the Kubernetes Mutating Webhook are found on the GitHub page for the Kubernetes Secrets Injector and Syncer.

Terraform

Go to GitHub

The DevOps Secrets Vault (DSV) Terraform Provider makes Secrets data available and provisions client secrets for existing roles.

Jenkins

The DSV Jenkins Plugin supports Scripted and Declarative Pipeline syntax.

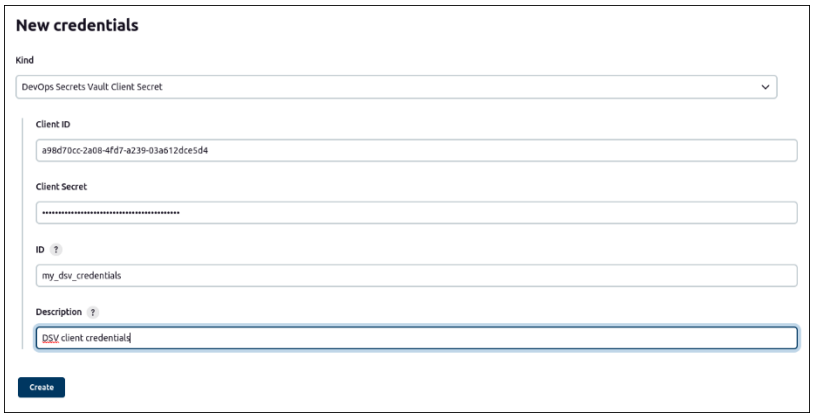
- Jenkins DSV Plugin: The DevOps Secrets Vault (DSV) Jenkins Plugin allows you to access and reference your Secrets data available for use in Jenkins builds.

  Go to GitHub
- Jenkins Declarative Pipeline Syntax: Jenkins Declarative Pipeline.

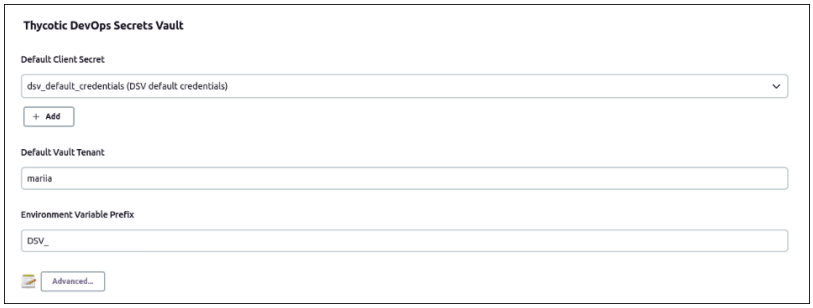
Usage

The current plugin can be installed from the list of plugins in Plugin Manager. While it is installed and Jenkins is restarted, the plugin can be used in Freestyle Projects or Multi-Configuration Projects and configured in the UI.

After installing the Thycotic DevOps Secrets Vault plugin (Manage Jenkins Credentials tab), create the DSV client credentials.

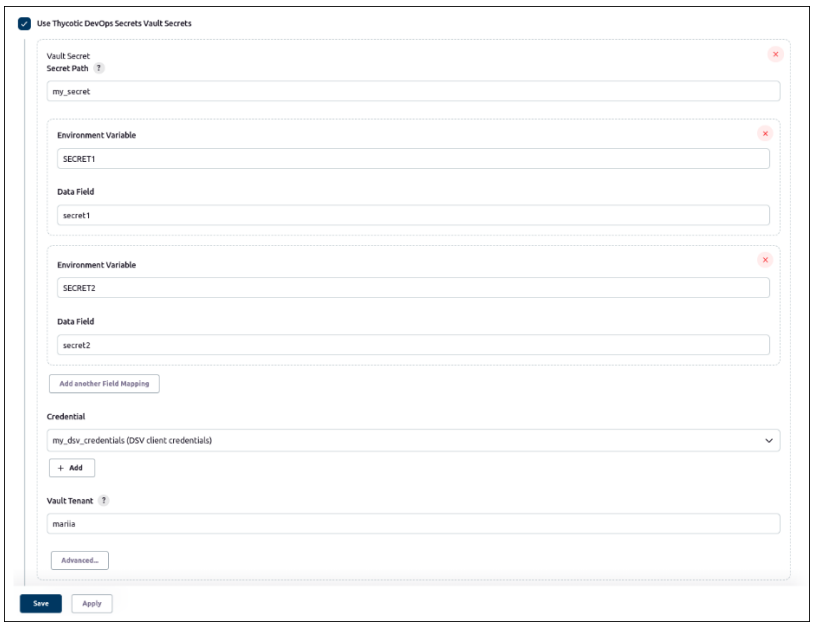


The default DSV configuration can be updated from the Manage Jenkins Configure Secrets tab.



Here, `my_secret` and its data fields `secret1` and `secret2` will be set as env variables `DSV_SECRET1` and `DSV_SECRET2`.

Note: By default, `DSV_` is applied as a prefix for every env variable name and can be configured in Jenkins configurations.



We can use this simple build script to check "is secret read."

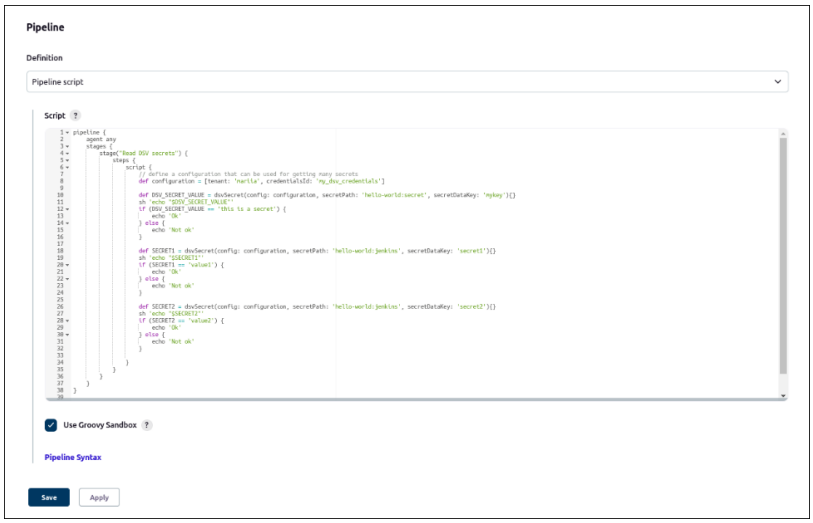
Note: For versions 1.1.1 and higher, output values are hidden from the console.

Jenkins Declarative Pipeline

The DevOps Secrets Vault (DSV) Jenkins Plugin allows secrets to be used in a Jenkins build using Declarative Support.

Pipeline Script

In version 1.1.1, `dsvSecret` can be used in a Pipeline script.



Example

```
pipeline {
    agent any
    stages {
        stage("Read DSV secrets") {
            steps {
                script {
                    // define a configuration that can be used for getting many secrets
```

```
                    def configuration = [tenant: 'mariia', credentialsId: 'my_dsv_
credentials']

                    def DSV_SECRET_VALUE = dsvSecret(config: configuration, secretPath:
'hello-world:secret', secretDataKey: 'mykey'){}
                    sh 'echo "$DSV_SECRET_VALUE"'
                    if (DSV_SECRET_VALUE == 'this is a secret') {
                        echo 'Ok'
                    } else {
                        echo 'Not ok'
                    }

                    def SECRET1 = dsvSecret(config: configuration, secretPath: 'hello-
world:jenkins', secretDataKey: 'secret1'){}
                    sh 'echo "$SECRET1"'
                    if (SECRET1 == 'value1') {
                        echo 'Ok'
                    } else {
                        echo 'Not ok'
                    }

                    def SECRET2 = dsvSecret(config: configuration, secretPath: 'hello-
world:jenkins', secretDataKey: 'secret2'){}
                    sh 'echo "$SECRET2"'
                    if (SECRET2 == 'value2') {
                        echo 'Ok'
                    } else {
                        echo 'Not ok'
                    }

                }
            }
        }
    }
}
```

Checking for hidden values can be seen in the console output.

```
Started by user Mariia
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Declarative pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Read DSV secrets)
[Pipeline] script
[Pipeline] {
[Pipeline] dsvSecret
[Pipeline] // dsvSecret
[Pipeline] sh
+ echo

[Pipeline] echo
Ok
[Pipeline] dsvSecret
[Pipeline] // dsvSecret
[Pipeline] sh
+ echo

[Pipeline] echo
Ok
[Pipeline] dsvSecret
[Pipeline] // dsvSecret
[Pipeline] sh
+ echo

[Pipeline] echo
Ok
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

GitHub

Go to GitHub

Use Delinea DevOps Secrets Vault for retrieval of your secrets.

Instead of storing all your secrets directly in your GitHub repo configuration, store client credentials to connect and retrieve the desired secret or multiple secrets from your secure vault.

GitLab

Go to GitHub

Delinea DevOps Secrets Vault (DSV) CI plugin allows you to access and reference your Secrets data available for use in GitLab jobs.

Azure DevOps

Go to GitHub

The Azure DevOps pipeline task is used to read secrets from the Delinea DevOps Secrets Vault.

DSV lookup plugin for Ansible

DSV has lookup plugin for Ansible in the `delinea.core` collection. The collection is certified and available in Ansible Galaxy.

To install `delinea.core` collection run:

```
ansible-galaxy collection install delinea.core
```

Use `ansible-galaxy collection list` to verify the installation. Example:

```
$ ansible-galaxy collection list delinea.core

# /root/.ansible/collections/ansible_collections
Collection   Version
------------ -------
delinea.core 1.0.0
$
```

Source code of the `delinea.core` collection is available on GitHub: DelineaXPM/ansible-core-collection.

Requirements

The DSV lookup plugin depends on version **0.0.1** of Python DSV SDK. To install it run:

```
pip install python-dsv-sdk==0.0.1
```

Authentication

Only available option for authentication is via client credentials, i.e. client id and client secret. Read more about client credentials here.

Usage

The DSV lookup plugin can be used to access data from DSV and then store it in variables within your playbook.

Use "ansible-doc" to display all available configuration options:

```
ansible-doc --type lookup delinea.core.dsv
```

Recommended way to use the plugin is to configure it with environment variables and then set only path to a secret in the playbook file:

```
vars:
    my_secret: "{{ lookup('delinea.core.dsv', '<path to secret>') }}"
```

Also you can set only client id and client secert as env vars and provide tenant name as a named argument:

```
vars:
    my_secret: "{{ lookup('delinea.core.dsv', '<path to secret>', tenant='<tenant name>')
}}"
```

Another option available from Ansible 1.5 is the Ansible Vault. Using it you can store client credentials for DSV in encrypted files.

Permissions

Note: We strongly recommend using policies to control access to secrets needed by the plugin. The role tied to the client should only have read access to applicable secrets.

For example if name of the role used to generate client credentials is "ansible-role" and in your playbook you have:

```
vars:
    dsv_secret_one: "{{ lookup('delinea.core.dsv', 'playbooks:example:one') }}"
    dsv_secret_two: "{{ lookup('delinea.core.dsv', 'playbooks:example:two') }}"
```

Then for this role create a policy with only read action allowed:

```
dsv policy create \
    --path 'secrets:playbooks:example' \
    --subjects 'roles:ansible-role' \
    --actions 'read' \
    --resources 'secrets:playbooks:example:one,secrets:playbooks:example:two'
```

Example

The example shows how you can use DSV lookup plugin to read secret from DSV and store in a playbook variable.

To prepare DSV for this example, you need to:

- create a secret

```
dsv secret create --path 'mysecret' --data '{"key": "1"}'
```

- create a role

```
dsv role create --name 'ansible-example'
```

- generate client credentials

```
dsv client create --role 'ansible-example'
```

- create a policy with permission that will allow role "ansible-example" to read "mysecret" secret

```
dsv policy create \
    --path 'secrets:mysecret' \    --resources 'secrets:mysecret' \    --subjects
'roles:ansible-example' \    --actions 'read'
```

This example requires Python and Ansible to be installed. To install Ansible follow official installation guide from Ansible docs.

Python 3.10 and the `ansible-core` version 2.13.5 are used.

```
$ ansible --version
ansible [core 2.13.5]
  < skipped for brevity >
  python version = 3.10.8 (main, Oct 13 2022, 22:36:54) [GCC 10.2.1 20210110]
  jinja version = 3.1.2
  libyaml = True
```

Install the `delinea.core` collection which includes DSV lookup plugin:

```
ansible-galaxy collection install delinea.core
```

Next, install Python DSV SDK:

```
pip install python-dsv-sdk==0.0.1
```

For security reasons we do not recommend passing client id and client secret directly as named arguments directly to lookup plugin from Ansible playbook. In this example set "DSV_CLIENT_ID" and "DSV_CLIENT_SECRET" env variables to your values of client id and client secret respectively.

Also set "DSV_TENANT" to your tenant name (e.g. "demo" for "https://demo.secretsvaultcloud.com")

Now create a simple playbook which reads secret "mysecret" from DSV, stores it in the "my_secret" variable and then prints "key" value from the secret's data.

```
cat <<EOF > dsv_playbook.yml
- hosts: localhost
  vars:
      my_secret: "{{ lookup( 'delinea.core.dsv', 'mysecret' ) }}"
  tasks:
    - debug: msg="Key retrieved from DSV = {{ my_secret["data"]["key"] }}"
EOF
```

Example of running the `dsv_playbook.yml` using `ansible-playbook`:

```
$ ansible-playbook dsvlookup.yml
< skipped for brevity >

TASK [debug]
*******************************************************************************************
**************************
ok: [localhost] => {
    "msg": "Key retrieved from DSV = 1"
}

PLAY RECAP
*******************************************************************************************
**************************
localhost                  :
ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

$
```

Please note that DSV lookup plugin is similar to reading a secret using DSV CLI. If you create a playbook like this:

```
cat <<EOF > dsv_playbook2.yml
- hosts: localhost
  tasks:
    - debug: msg="{{ lookup( 'delinea.core.dsv', 'mysecret' ) }}"
EOF
```

and run it, then the output will be similar to reading the secrets using CLI:

```
dsv secret read --path mysecret
```

Puppet

Go to GitHub

The Puppet module facilitates the consumption of secrets from DevOps Secrets Vault (DSV).

Note: Although Puppet is no longer actively supported, implementation details can be addressed by the Integrations Support team at integrations@delinea.com.

Chef

Go to GitHub

The DSV Chef Cookbook provides a new resource, dsv_secret, as well as a sample cookbook. This resource allows integration into DSV.

Note: Although Chef is no longer actively supported, implementation details can be addressed by the Integrations Support team at integrations@delinea.com.

Release Notes

DevOps Secrets Vault is regularly updated to provide improvements and introduce features.

As a Cloud application, DSV lacks version numbers; the current version serves all users because it is always the only version available.

The Command Line Interface (CLI) is locally installed using OS-specific executables. These bear version numbers to reflect updates.

- The version number will always be the same across the OS-specific editions of the CLI executable.
- You obtain these updated versions of the CLI executables by downloading them from DevOps Secrets Vault Downloads.
- The CLI itself will notify you when a new version is available for download.
- Generally, older versions of CLI executables will continue to work, but you will want to have the latest executables to benefit from fixes and obtain new features.

DSV Cloud Service: Change Log

Update	Notes
August 2023	**CLI Release Notes**
	new feature: Added a Policy Editor to the UI. This separates the permission documents from the policy, allowing for more granular administration.
	fixed: Fixed an error when retrieving some dynamic secrets using an engine.

Update	Notes
June 2023	CLI Release Notes
	fixed: Reset the number of failed attempts for SIEM configuration if audit exporting was successful.
	fixed: Improved performance for tenants with no SIEM configurations by saving an empty result for tenant in local cache.
	fixed: Added a task to cleanup old engine messages.
	fixed: Set CORS headers even when returning 500 HTTP code.
	fixed: Generated audit logs on writing response header, instead of on writing response body. This enables audit logs for endpoints that do return an EMPTY response body.
	fixed: Added missing path/query parameters for a service principal search and deleted the API for the swagger specification.
May 2023	CLI Release Notes
	new feature: The dsv-k8s-sidecar repo is now open source and simpler to use. https://github.com/DelineaXPM/dsv-k8s-sidecar.
	fixed: The bulk add of members to a group and a member to many groups is working in the UI.
	fixed: Resolved an issue migrating a CEF extension format from JSON to key-value pairs for new CEF SIEM endpoints. Old CEF endpoints will continue using JSON format.
April 2023	CLI Release Notes
	new feature: Added support for Tilt to the dsv-k8s repo. With a single command, you can stand up an interactive session and see how to use DSV in k8s.
	fixed: Fixed a validation error on UI Break Glass page.
	fixed: The UI secret preview now indicates the correct user permissions.
	fixed: Fixed an issue that prevented policies to give access to a subset of groups, specifically “groups:<prefix.*>”.
March 2023	CLI Release Notes

Update	Notes
	new feature: The UI can show metadata for users without read permissions, but who have list permissions.
	fixed: A UI logout now successfully invalidates the session as well as discarding the token.
February 2023	CLI Release Notes
	new feature: The UI now supports Break Glass configuration.
	new feature: BYOK commands have been added to the CLI with `dsv byok update`.`
	improvement: The CLI supports the following new flags in the Rest API and CLI for searching and sorting. - dsv engine list: new --query, -q, -pool-name, -sort, --sorted-by flags. See dsv engine list --help - dsv pool list: new --query, -q, --sort, --sorted-by flags. See dsv pool list --help * dsv role search: new --sort, --sorted-by flags. See dsv role search --help - dsv client search: new --sort flag allows sorting of client credentials by created time. - dsv user groups: new --query, -q, --limit, --cursor, --sort flags. See dsv user groups --help - dsv groups search: new --sort, --sorted-by flags See dsv groups search --help - dsv user search: new --sort, --sorted-by flags. See dsv user search --help - dsv policy search: new --sort, --sorted-by flags. See dsv policy search --help improvement: Security updates now prevent hijacking of the UI in a frame.
	improvement: Security updates now prevent hijacking of the UI in a frame.
January 2023	CLI Release Notes
	new feature: The following installers have been added for all architectures: Homebrew, Aqua, PowerShell, Curl, Snap, and Scoop.

Update	Notes
	improvement: All new development can now be added directly to GitHub, rather than using bulk changes. As a result, our workflows are now updated with additional internal tools, including GoReleaser.
	fixed: Fixed an issue where authentication providers and SIEM pages were not shown in the UI for some users.
December 2022	CLI Release Notes
	improvement: Search functions have been ported to Rest from GraphQL.
	improvement: The character limit for policies has been increased from 2k to 8k.
	fixed: User members of groups that were delegated rights to create groups and roles were granted the rights in the CLI and API but denied those rights in the UI. Now, the delegated rights are now correctly recognized in the CLI, API, and UI.
	fixed: Fixed a bug that incorrectly required a data field when updating a secret in the UI.
November 2022	**CLI Version**: 1.39.0
	new feature: A new dsv-gitlab plugin is available. The plugin integrates into GitLab CI to retrieve secrets from DSV.
	improvement: An endpoint was added to view expired service principals and allow for manual deletion.
	improvement: Secrets are now masked in the Jenkins plugin logs.
	improvement: Context was added to a dialog that alerts the user when deleting a pool with engines attached.
	improvement: Updates have been made to the caching rules for sensitive UI pages.
	improvement: Users are prohibited from deleting a break glass secret.
	improvement: The API now deletes engines immediately, instead of allowing an optional force flag.

Update	Notes
	improvement: References to the `root-ca-path` and `assumed-role` flags have been removed from the CLI documentation.
	improvement: In the CLI, usage is no longer printed for unknown flags.
October 2022	**CLI Version**: 1.38.0
	new feature: DSV supports Bring Your Own Key (BYOK) encryption key management.
	improvement: GitHub updates include access to the CLI and a GitHub action, dsv-github-action, to use Delinea DevOps Secrets Vault for retrieval of your secrets.
	new feature: The CLI supports creating a new profile using certificate authentication.
September 2022	**CLI Version**: 1.37.0
	new feature: The UI includes an Audit page that presents actions recorded for specific users and the date recorded.
	new feature: The UI includes a dashboard that presents total requests for an adjustable time interval. Total secrets across all vaults is also displayed on the dashboard.
	new feature: Authentication providers can be created and deleted from the UI.
	improvement: The default profile can now be changed in the CLI.
	improvement: Scriptable initialization for the CLI is available only for username/password or client credentials authentication.
	new feature: The Ansible core collection for Delinea DevOps Secrets Vault is now available.
	improvement: Added the AWS authentication method for Terraform.
August 2022	**CLI Version**: 1.36.0
	new feature: The UI now supports SIEM. The user can create and delete SIEM integrations from a selection available in Administration.

Update	Notes
	new feature: At login, a **Remember me on this device** checkbox is added. When enabled, the default behavior for storing user credentials is maintained. Disable the checkbox and user credentials are not stored for subsequent logins.
	new feature: Added wizard support to add multiple Permission documents to a single Policy.
	new feature: Added wizard support for SIEM functions.
	fixed: Corrected an issue where during the init configuration, setting the store type to **none** caused and error.
	fixed: Resolved some inconsistencies with **Role Name** casing when creating or referencing a Role.
July 2022	**CLI Version**: 1.35.0
	new feature: Official binaries are available for Apple M1s for download at: https://dsv.secretsvaultcloud.com/downloads.
	new feature: Splunk is now supported for audit logging in SIEM integrations.
	new feature: Edits made to a secret are stored as versions, which can be rolled back and implemented as the current version of the secret.
	new feature: Added support for using Declarative syntax to call our Jenkins plugin from a pipeline.
	new feature: Cloud Authentication support is added to the Go SDK.
	new feature: Ansible Plugin now supports the EU domain and other top level domains.
June 2022	**CLI Version**: 1.34.0
	new feature: Policies can be viewed, created, and deleted in the UI. Basic policy functionality is supported, with future enhancements to come for full functionality and customization.
	new feature: Kubernetes side car is now supported on Microsoft Windows OS.
	new feature: Kubernetes webhook now supports dynamically updating secrets.
May 2022	**CLI Version**: 1.33.0

Update	Notes
	new feature: The UI now supports the creation of secrets in Shared Vaults, as well as Home Vaults.
	new feature: The clients attached to a Role are viewable in the UI. Clients can also be created and deleted using new features in the UI.
	new feature: Engines and engine pools are now accessible through the UI. Engines and pools can be viewed, created and deleted in the updated UI.
	improvement: CLI timeout is now manually configurable. If a user's CLI is idle for a predefined amount of time, a timeout is initiated. This is controlled by the `refreshTokenTTLHours` value in the config file, and can be set per tenant.
	improvement: The creation of a SIEM endpoint inside the CLI is now supported.
April 2022	**CLI Version**: 1.32.0
	new feature: Added Dynamic Secret Support for MongoDB. When using a MangoDB dynamic secret, you can create and delete local users in a just-in-time manner in your database.
	improvement: The CLI wizard has been updated for improved user interaction.
	improvement: Searching inside the CLI is more consistent and convenient. Resources can be searched without using the search term. This improvement has been made for Secrets (Home Vault and Root), Groups, Users and Roles.
	improvement: Added UI improvements for displaying Home Vault Secrets. Users now can view the personal secrets in the UI that are created in the Home Vault, using the CLI.
	improvement: The UI for Users and Secrets includes an Audit tab for viewing audit activity.
	improvement: The UI for Secrets allows you to delete secrets.
	improvement: The Kubernetes Webhook plug-in now supports custom namespaces.
	improvement: The visual appearance of the UI has been updated to represent our company brand.
	fixed: Fixed an issue with the SIEM integration to allow for endpoint support.

Update	Notes
February 2022	**CLI Version**: 1.31.0
	improvement: Added guided sub-command support on CLI Wizards to create and update secrets.
	improvement: Added support in the Kubernetes Sidecar extension for Authentication by Cert. The Broker can be configured to use the Certificate method of authentication instead of client credentials.
	fixed: An incorrect response that displayed after editing/updating a Thycotic One user has been resolved. Previously, updating a Thycotic One user would add an extra thy-one prefix to the displayed user name.
January 2022	**CLI Version**: 1.30.0
	improvement: When selecting a group in the UI, you will now see a **Members** tab. You can use the **Members** tab to add and remove users to and from the selected group.
	improvement: We have added Role Management to our UI. You can now view, create, edit, and delete roles, as well as view the client credentials that are attached to the selected role.
December 2021	**CLI Version**: 1.29.0
	improvement: Our Kubernetes sidecar extension now supports the use of custom namespaces. Pods can now be restricted to only access secrets located in that namespace, thereby preventing pods from accessing secrets that they do not need access to.
	improvement: When selecting a user in the UI, you will now see a **Membership** tab. Here, you will be able to view the current group membership of the selected user, as well as edit the group membership.
	fixed: When using the CLI wizard to set up a siem connection, a blank input for the default value of Engine routing would lead to an error "Blank input is invalid". The CLI now allows the blank input and assumes the default value.
	fixed: Previously in the CLI, our parsing function would not allow the creation of a secret/role with a '-c' suffix in path. The behavior has been corrected.

Update	Notes
	fixed: Federated User Accounts (such as thy-one users) were not able to see management modules in UI, due to an issue with querying the logged-in user's permissions. The backend query would come up blank because the prefixed user account (thy-one:) isn't recognized.
	fixed: User Accounts that were created with usernames that include uppercase letters were not able to see management modules in the UI, due to an issue with querying the logged-in user's permissions. The backend query would come up blank because the user is not recognized in any policy with an uppercase letter (usernames are automatically forced to lowercase when referenced to policies). User and Role creation now forces the casing of characters in user and role names to lowercase.
	fixed: When editing the currently populated `displayname` field of a user with an empty string value, the cmd would successfully execute, but it would not actually change the field to an empty value. We have added error handling that states "Editing a User's displayname should be 3 to 100 characters."
November 2021	**CLI Version**: 1.28.0
	new feature: Geolocation-Based Routing - Previously, our data flow configuration was an active-passive failover with the East Coast site as our primary for all U.S. customers. To ensure the same performance for our West and East Coast customers, we have changed to an active-active failover configuration. Now, U.S. customers will automatically route to the site closest to their Data Center, further minimizing any latency issues. Geolocation is determined by IP address.
	improvement: Added the delete function for deleting groups in the UI.
October 2021	**CLI Version**: 1.27.0
	improvement: Added the ability to View and Create Groups in the UI.
	improvement: Added the **Last Login** field to the User Preference Page in the UI.
	improvement: Authentication by certificate is now available as an option in the CLI.
	fixed: Runtime error when using a bad flag for the **dsv group create** command.
	fixed: Broken Azure authentication due to a change from Microsoft (letter casing was changed in the Resource Group ID). Reauthorization may be required.

Update	Notes
September 2021	**CLI Version**: 1.26.0
	improvement: Added Support for AWS EC2 instances that use IMDSv2. Support for IMDSv1 will continue.
	improvement: Added the ability to view, create, and edit users in the UI. (Safari not supported)
	fixed: Resolved an issue where Azure authentication fails when attempting to initialize dsv from an azure VM.
August 2021	**CLI Version**: 1.25.0
	Engine Version: 1.9.0
	improvement: Authentication by Certificate now requires a Base64 encoded private key along with the certificate.
	improvement: BreakGlass enhancement - When deleting a user, DSV will now check to see if the user is a member on the new admin list for breakglass. This is to ensure that no user on this list is deleted without providing a replacement for the potential breakglass admin.
	improvement: Added guided sub-command support on CLI Wizards for Clients, BreakGlass, Engine, and Pool.
	improvement: In the DSV UI, added the ability to make changes to editable fields on existing secrets.
July 2021	**CLI Version**: 1.24.0
	improvement: For GCP Dynamic Secrets, added configurable time-to-live for OAuth.
	improvement: Breakglass will now allow duplicate policy entries and cleanup duplicates upon removal.
	improvement: In the DSV UI, improved the view/preview of basic secret attributes.
June 2021	**CLI Version**: 1.23.0
	Engine Version: 1.8.0

Update	Notes
	new feature: DSV now supports authentication by certificate.
	fixed: Fixed a bug preventing the DSV Engine from connecting in .au regions.
May 2021	**CLI Version**: 1.22.0
	Engine Version:1.7.0
	new feature: The emergency Break Glass feature allows DSV users to recover Super Administrator access if those credentials are lost or compromised.
	new feature: Introduced the first version of a web GUI. It includes the ability to list secrets that you have access to.
	new feature: SIEM integration is now available through the DSV engine.
	new feature: The DSV Engine can now be run as a service.
April 2021	**CLI Version**: 1.21.0
	new feature: DSV now supports Encryption as a Service using user-supplied keys.
	new feature: The `report` command generates a list of secrets or groups. Use the `secret` subcommand to see the secrets available to a user, group, or role. Use the `group` subcommand to see the group memberships of a user or role.
	improvement: Users and roles can now be searched by provider or fully-qualified name.
	improvement: Thycotic One user login is now interactive in the CLI. The API login route has been disabled.
March 2021 Release 2	**CLI Version**: 1.2.0
	Engine Version: 1.6
	new feature: DSV now offers a fully managed Encryption as a Service.
	improvement: Users can be given a display name using the `--displayname` flag.
	improvement: Maximum policy limit per tenant has been increased from 500 to 1,000.
	improvement: Secrets can now be accessed by ID using the `--id` flag.

Update	Notes
	improvement: Secret searches can now be sorted using the `--sort` flag.
March 2021	**CLI Version**: 1.19.0
	Engine Version: 1.5
	improvement: Unresponsive SIEM endpoints will be automatically deregistered after ten failed attempts.
	improvement: DSV now supports syslog SIEM integration over TCP.
	improvement: The dsv-engine now prioritizes flags over configuration files.
	improvement: The help menu for the `audit` command has been updated.
	fixed: When creating a new thycotic-one user, passing an external-id flag will no longer prevent account creation.
	fixed: The dsv-engine wizard for Windows PowerShell and macOS bash no longer truncates user-token and private-key.
February 2021	**CLI Version**: 1.18.0
	Engine Version: 1.4
	new feature: Added wizards for run and register in the DSV Engine.
	new feature: Added dynamic secret support for Azure Microsoft Graph API.
	improvement: Added DSV_VERBOSITY flag for use with docker image scripts.
	fixed: Engines registered in containers will now run using the newly created configuration file.
January 2021	**CLI Version**: 1.17.0
	Engine Version: 1.3
	new feature: Added `sendWelcomeEmail` property. When a user is created using Thycotic One for authentication, setting `sendWelcomeEmail` to `true` will send a new login email to the user.

Update	Notes
	new feature: Added ability to modify authentication provider details using the `edit` subcommand.
	improvement: After account lockout from failed authentication, the CLI now displays the time remaining until reauthentication is available.
	improvement: Updated dsv-engine validation and API error messages for clarity.
	fixed: When querying log data, the correct dates will display.
	fixed: Client credential `URL` value no longer switches with search.
	fixed: External ID is no longer required for Thycotic One users.
December 2020	**CLI Version**: 1.16.0
	Engine Version: 1.2
	new feature: Added dynamic secret support for contained MSSQL databases.
	new feature: Added ephemeral client credentials. Credentials can be limited using the `--uses` and `--ttl` flags.
	improvement: Passwords can no longer be reused, increasing security.
	improvement: DSV APIs now limit the number of invalid login attempts, increasing security.
	improvement: Azure dynamic secrets now use consistent naming conventions between the base and dynamic secret.
	improvement: strictTransportHeader is present in requests.
November 2020	**CLI Version**: 1.15.0
	Engine Version: 1.1
	new feature: Added dynamic secret support for PostgreSQL and Oracle databases.
	improvement: Engines and pools can now be manipulated via the `engine` and `pool` commands.

Update	Notes
October 2020	**CLI Version**: 1.14
	Engine Version: 1.0
	new feature: Added the DSV Engine. This agent is installed on the customer network for access while limiting the need to open the firewall. Initially for database dynamic secrets, but in the future will be used for password rotation, authentication, or other needs.
	new feature: Bootstrapped client credentials. When creating client credentials, a one-time use URL can be created so that the new machine or application can retrieve the Client Secret.
September 2020	**CLI Version**: 1.13
	improvement: CLI name changed from "thy" to "dsv" in downloads and documentation for all commands
	new feature: Home Vault GA. Completed Roles, GetByVersion, Rollback, Restore, Policy for giving others access.
	improvement: Wizards for Groups will not allow invalid Users
	improvement: Wizards for Users look for Auth provider and act accordingly rather than ask for a password first
	improvement: secrets attributes can be updated without affecting other fields
	improvement: Thycotic One users not sent sign-up emails by default. Can change this setting
	improvement: whoami command provides more information for cloud auth providers
	improvement: Group names can't have spaces
	improvement: Roles with Auth providers must include an external ID
August 2020 (Update 1)	**CLI Version**: 1.12.1
	fixed: CLI update check

Update	Notes
August 2020	**CLI Version**: 1.12
	new feature: Home Vault Beta. Users get their own secret space without needing a policy.
	improvement: Global flags defined
	improvement: Policy update help information and examples.
	improvement: Improved auth-provider help
	improvement: Pre-validation for SIEM endpoints
	fixed: Added Metadata to Groups
July 2020 (Update 1)	**fixed**: Enforce case insensitivity on subjects returned in JWT record.
July 2020	**CLI Version**: 1.11
	new feature: SSH public key generation and SSH Certificate signing/storage was added.
	new feature: CLI now contains wizards for Users, Groups, and Roles.
	improvement: Policy update help information and examples.
	improvement: Added IDs and status information to audit records.
	improvement: Standardized on the use of colons for policies instead of slashes
	fixed: Enhancements to auth providers.
	fixed: Group memberships are not evaluated for policy updates.
	fixed: Group member sometimes returned code 500 (internal server error) on deletion attempt.
June 2020	**CLI Version**: 1.10
	new feature: SIEM endpoints. Support Syslog, CEF, and JSON log formatting on TLS,TCP, UDP, HTTP, and HTTPS transport protocols.
	new feature: Introduced CLI wizards to PKI, SIEM, Policy, and Auth-provider commands for simplified human navigation.

Update	Notes
	improvement: Additional Secrets search capabilities. Enabled search for Secrets on any attribute, path, or description.
	improvement: Provide the ability to add a CRL URL to a signing certificate.
	fixed: CLI version check fixed regardless of the update cache
	fixed: Group membership evaluated for policy updates.
	update: Deprecated "settings" attribute on the Configuration document will be removed next release. All auth provider management should go through the config/auth endpoint
May 2020	**CLI Version**: 1.9
	new feature: Google Cloud Platform (GCP) Dynamic secrets. DSV can issue ephemeral secrets for GCP service accounts
	new feature: OIDC Support. Thycotic One can connect to any IDP provider that supports OIDC and in-turn those users can authenticate to DSV.
	improvement: If a base secret has a dynamic secret linked to it, it errors on attempt to delete it.
	improvement: New flag for singing a leaf certificate that includes the singing certificate for the trust chain
	fixed: Groups with 3rd party auth fixed
	fixed: Client permission check
	fixed: Restore user with 3rd party auth
April 2020	**CLI Version**: 1.8
	new feature: Google Cloud Platform Authentication using service accounts and GCE metadata
	new feature: X.509 Certificate Issuance. Certificate signing capablilties.
	improvement: Azure dynamic secret role validation
	improvement: Azure dynamic secret temporary service principal cleanup. (deletes expired service principals in Azure MSI)

Update	Notes
	improvement: Dynamic secrets easier to edit
	fixed :CLI encryption key works if store path is in a non-default location.
	fixed :Client tokens used even if already logged in.
March 2020	**new feature**: Azure Dynamic Secrets. DSV can use Azure Service Principals to provide ephermal credentials
	new feature: (API only) Ability to issue X.509 certificates
	improvement: Ability to retrieve auth settings by version
	improvement: Make help commands available even if the CLI config is missing
	improvement: Protect error check. Protect against creating policy errors
	improvement: Ability to search for dynamic secrets given a base secret
	improvement: Improved error reporting for dynamic secrets
	fixed : A malformed policy could prevent reading all policies.
February 2020	**improvement**: protect against user lockout. When editing authentication providers, block any changes that locks the user out of the account.
	improvement: audit search results now inclusive of the dates in a range (previously the first day was omitted).
	improvement: consistent version listing. Removed the “v” in the version number when searching older versions to be consistent with other listings.
	new feature: AWS Dynamic secrets. DevOps Secrets Vault can use AWS Security Token Service (STS) to provide ephemeral AWS credentials.
January 2020	**improvement**: the `rollback` command allows you to roll back Secrets (and Policies and Authentication Providers) to their earlier versions
	improvement: Windows users can now more easily edit Secrets, with Notepad or another designated editor opening right from the command line
	fixed: a defect in the Kubernetes extension caused verbose error reporting on irrelevant conditions

Update	Notes
December 2019	**improvement**: the `thy init` command no longer requires an `--advanced` flag, as it now always steps through key initialization settings
	improvement: the DSV CLI executables will now prompt when a new version is available for download
	fixed: a defect in CLI audit log listing behavior would show listings even when the start date was in the future and would show listings later than the end date
November 2019	**improvement**: after deleting a Secret, Role, User, Group, Policy, or Authentication Provider, the new `restore` command will undelete the item up to 72 hours later
	improvement: architectural changes back uptime of 99.999 percent; continuous backup enables hot backup fail-over in under a minute
October 2019	**improvement**: a Secret's data, attributes, and description can be individually updated via the *update* command's new *--data*, *--attributes* and *--desc* flags, respectively
	improvement: the Secret *update* command's new Boolean *--overwrite* flag controls whether the *--data* flag's content overwrites or merges with extant data object fields
	improvement: improvement: updated server side policy caching to better handle permission updates
	improvement: the CLI now supports finding and examining audit logs, previously possible only via the API
September 2019	**improvement**: better scaling of configuration files achieved by keeping policies and authentication providers in separate files
	improvement: the *permissions* command has been superseded by the *policy* command; named policies no longer require everyone to modify a global document
	improvement: the new Change Password feature enables users to change their passwords

Update	Notes
	improvement: adding Users to a Group achieves permissions delegation
	improvement: deleting a Secret now deletes all past versions, rather than just the latest
	fixed: the API Audit Search function's bug, related to the improperly named *Secret* parameter, is resolved by the properly named *path* parameter
August 2019	**fixed**: issue where the refresh token generated by Thycotic One authentication was not correctly generating the full subject name and could cause access denied errors
	fixed: issue where adding a pre-existing Thycotic One user as a DSV User would not correctly save the Thycotic One user id
	fixed: issue where the config created and updated metadata fields that were not properly shown in responses
	added: version validation to *config update* to help prevent conflicts
July 2019	first General Availability of the service

Support

The Delinea Support Community is available at: *https://support.delinea.com/s/*.

This page provides a high-level summary of support options.

Note: Support options depend on license status, with paid licenses having more support channels. See our Support Services Guide for complete details about our support policy.

Free Licenses

If you have a free license, use this document collection to find information about DSV and how to use it.

Paid Licenses

Paid DSV subscribers have access to support by phone and email. You also can open a case in Delinea's support ticketing system, which promotes follow-through to issue resolution.

- Use the means you prefer, except for Severity 1 issues—for those, always use phone support.

  Severity 1 means a critical problem that has caused *complete loss of service* and work cannot reasonably continue at your worksite.

Obtaining a Support PIN

To obtain support by email or phone, first log in to the Support Portal to obtain a PIN. The PIN validates that your license includes support, and you must provide the PIN in your email or when you call. The PIN also makes it easier for the person helping you to locate your customer records and give you better support.

- Visit the Support Portal Login Page using the credentials you received when your organization subscribed to the DSV service.
- After logging in, you will be on the main page. Click on the large blue bar labeled PIN to obtain a PIN number.

Support by Phone

Delinea delivers support by phone worldwide. Select the applicable number from this list:

Region	Country	Support Number
AMERICAS	all	+1 202 991 0540
EMEA	UK	+44 20 3880 0017
	Germany	+49 69 6677 37597
APAC	Australia	+61 3 8595 5827
	Philippines	+63 2 231 3885
	New Zealand	+64 9-887 4015
	Singapore	+65 3157 0602

Support by Email

Send your email to support@thycotic.com **with the PIN number as part of the subject line** of your email, for example:

- PIN 345 Workflow Stopped Unexpectedly

Include this information:

1. company name
2. contact name
3. contact phone number
4. product name
5. details of the issue

You must send your email using an email address already noted in your account with Delinea.

- Sending a support request from an email address not on file may delay our response.

Support Ticketing

As an alternative to support by email or phone, you can open a support ticket and track your issue to resolution.

- Visit the Support Portal Login Page/login using the credentials you received when your organization subscribed to the DSV service.
- After logging in, you will be on the main page. Click the **Cases** tab, then **Create a Case**.
- Follow the instructions to complete your case.