



IBM Security Verify Privilege

Privilege DevOps Vault User Guide

# Table of Contents

Foreword .....	6
Overview .....	7
API .....	7
Quick Links .....	8
Third-Party Downloads.....	8
Quick Start Guide.....	8
Download the CLI Executable for your Operating System .....	8
Initialize the CLI.....	9
CLI Secrets Examples.....	11
Creating Users .....	14
Local User and Thycotic One User Authentication .....	15
Provide Users Access to Secrets.....	17
CLI Reference .....	19
CLI Command Syntax.....	20
Objects .....	20
Workflows for Creating or Updating Objects.....	20
Parameters .....	21
Output Modifiers.....	22
Encoding and Beautify.....	22
Filter .....	23
Out.....	23
Output Piping.....	24
Secret.....	24
Commands that Act on Secrets.....	24
Examples .....	25
User.....	30
Understanding Qualified Usernames .....	30
Commands that Act on Users.....	30
Examples .....	31
Group .....	34
Commands that Act on Groups .....	34
Examples .....	34

Role.....	37
Commands that Act on Roles .....	37
Examples .....	37
Client.....	38
Commands that Act on Clients.....	39
Examples .....	39
Policy.....	41
Policy Evaluation.....	42
Policy Examples.....	43
Admin Policy and Auth Providers.....	49
Commands that Act on Policies .....	49
Home Vault .....	52
Examples .....	52
GetByVersion.....	55
Authentication .....	57
Password .....	57
Client Credentials .....	57
Third Party Authentication .....	57
Profiles.....	58
Add a Profile to a Config.....	58
See the Config Contents .....	58
Using an Alternate Profile for a Specific CLI Action.....	58
Authentication: AWS.....	58
Authentication: Azure.....	63
Authentication Google Cloud Platform (GCP).....	67
Authentication: OIDC.....	83
OIDC Providers .....	83
Azure AD OIDC Example.....	89
Okta Identity Provider Example .....	90
Dynamic Secrets.....	98
Linking.....	98
Search for linked Secrets .....	99
AWS Dynamic Secrets .....	100

AWS Federate .....	100
AWS Assume Role .....	102
Azure Dynamic Secrets.....	106
Azure Service Principal.....	110
GCP Dynamic Secrets.....	116
Setup.....	117
OAuth Access Token.....	117
Service Account Key .....	119
MySQL Dynamic Secrets.....	121
Privilege DevOps Vault Engine Required .....	121
Dynamic Secret Setup .....	121
Sending a MySQL task to an engine .....	122
Privilege DevOps Vault Engine .....	122
Customer Firewall .....	123
Registering a pool and an engine .....	123
Starting an engine.....	123
Certificate Issuance .....	124
Generate a Signing Certificate.....	124
Register (Import) a Signing Certificate.....	126
Generate and Sign a Leaf Certificate.....	128
Sign a Certificate Given a Certificate Signing Request (CSR) .....	129
SSH Key Issuance.....	131
Adding an SSH public key to a server.....	131
Trusting a group of keys signed by a root key.....	132
Architecture and Security.....	133
Availability.....	134
Business Continuity and Disaster Recovery .....	134
Confidentiality.....	134
Client Authentication.....	134
Integrity Checks.....	135
Personally Identifiable Information (PII) and GDPR.....	135
Audits:.....	137
Logging Format and Transport Protocols supported.....	137

SYSlog.....	137
CEF.....	139
JSON .....	140
Release Notes.....	142

## Foreword

IBM Security Verify Privilege DevOps Vault (Privilege DevOps Vault) uses the term *secret* to address a privileged account. Therefore, the product is also referred to as DevOps Secrets Vault (DSV). Privilege DevOps Vault is powered by the original product manufacturer, Thycotic. The product documentation contains several links that can direct to Thycotic's documentation. Also, since Privilege DevOps Vault interoperates with other cloud providers and development platforms, links to such third parties are also part of the documentation.

# Overview

IBM Security Verify Privilege DevOps 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.

IBM Security Verify Privilege DevOps 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.

- 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)
- SDK support
  - [Java](#)
  - [Go](#)
  - [Python](#)
  - [NET Core](#)
- DevOps Tools Support (Plugins)
  - [Jenkins](#)
  - [Terraform](#)
  - [Kubernetes](#)
  - [Ansible](#)
- Robotic Process Automation
  - [UIPath](#)
  - [Automation Anywhere](#)
  - [Blue Prism](#)
- SOC II Compliance - report available upon request

## API

This documentation is for general IBM Security Verify Privilege DevOps Vault operation and command line interface (CLI). If you prefer the API, here is the [API documentation](#)

## Quick Links

### Third-Party Downloads

[jq Library](#) for filtering JSON results

[Linux pass](#)

[Windows Credential Manager](#)

[AWS CLI](#)

[Azure User Assigned MSI](#)

## Quick Start Guide

### Download the CLI Executable for your Operating System

Download the Command Line Interface executable files to each of the workstations where you operate IBM Security Verify Privilege DevOps Vault.

- IBM Security provides Privilege DevOps Vault CLI executables for multiple platforms [here](#).
- Once installed, these CLI executables periodically check the download site for updates and inform you if an update is available.

### Rename the Executable

The executable file name reflects the OS and 32-bit or 64-bit architecture. Rename the executable to *dsv* or *dsv.exe* to simplify command entry.

### Place the Executable

Place the executable in the file directory location of your choice and note the path.

### Add the Executable Path to the PATH Environment Variable

While not required, adding the location of the executable to your PATH environment variable enables you to invoke *dsv* without specifying its path or having to pre-pend `.\`

- For Windows, press the Windows key and type "edit environment variables". Select the offered item.
  - In the Environment Variables dialog, under the System Variables section, select the Path and click edit.
  - Add the path to the *dsv* executable—for example `*C:\Users<name>\` and save.
- 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`



## 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, it fills out the full command 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 shows the output of any secrets below that path. Such as `resources/us-east-1/server` `resources/us-east-2/server`.

## Initialize the CLI

### Required Information

Privilege DevOps Vault CLI initialization presents you with a series of questions and choices. If you are the **initial administrator**, that is the person who setup the tenant, then you have the required information from signing-up. However, if you are not the initial administrator, you need to collect this information from that person:

- Tenant
- Domain
- local or federated user, and if federated, which authentication provider
- credentials - username or access key, password or secret key as examples

### "dsv init"

Begin setup with the `dsv init` command. This starts a workflow:

```
dsv init Please enter
tenant name: example
```

Specify the tenant name IBM Security provided when setting up your organization's account.

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

```
Please choose domain:
(1) secretsvaultcloud.com (default)
(2) secretsvaultcloud.eu
(3) secretsvaultcloud.com.au
```

Your domain is based on the server location that was chosen during provisioning: United States, European Union, or Australia/Asia, respectively.

NOTE: In all of these selections with numbered choices, the first choice is marked (*default*) because that is the selection if you simply hit "enter" without entering a number.

Next, Privilege DevOps Vault prompts you about **credential storage**.

Please enter store type:

- (1) File store (default)
- (2) None (no caching)
- (3) Pass (linux only)
- (4) Windows Credential Manager (windows only)

Select (1) *File store (default)* to keep the credentials in a configuration file. If you select this, Privilege DevOps Vault prompts for the storage location.

Select (2) *None (no caching)* to avoid storing the credentials. With this option active, Privilege DevOps Vault requires authentication with every command.

Select (3) *Pass (linux only)* to use [Linux pass](#) for encrypted storage.

Select (4) *Windows Credential Manager (windows only)* to use [Windows Credential Manager](#) to store credentials.

Your next selection concerns the **type of authentication**.

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)

Select (1) *Password (local user) (default)* to authenticate by username and password.

Select (2) *Client Credential* to authenticate by Client ID and Client Secret authentication; this supports use of Privilege DevOps Vault commands by applications.

Select (3) *Thycotic One (federated)* to authenticate using IBM Security's access manager.

NOTE: The person who signed up for IBM Security Verify Privilege DevOps Vault is the *initial administrator* and is automatically setup using Thycotic One. 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 (4) *AWS IAM (federated)* to authenticate as a trusted Identity Access Management Role or User.

Select (5) *Azure (federated)* to authenticate as a trusted Azure Managed Service Identity (MSI).

Select (6) *GCP (federated)* to authenticate as a trusted Google Service Account.

Select (7) *OIDC (federated)* to authenticate through Thycotic One to an external IDP using the OIDC protocol.

Next, the initialization process prompts about the **cache strategy for Secrets**. The choice here depends on your specific set of concerns around security, network connectivity, performance, and systems availability.

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

Note that in this context, *server* refers to your Privilege DevOps Vault tenant and *cache* refers to storage on the local machine with the CLI installed.

Select (1) *Never (default)* to never cache Secrets. Every credential request requires an API call.

Select (2) *Server then cache* to make an API call every time, but if not accessible, then the cached Secret is used.

Select (3) *Cache then server* to use the cached Secret unless it has expired, in which case an API call is made.

Select (4) *Cache then server, but...* If the cached Secret has expired, an API call is made for the Secret. If the API call fails, then use the expired cached Secret.

- Finally, you are prompted for your credentials and authentication provider. For the initial administrator, they are the username and password that you setup in Thycotic One during the sign-up, with the username often your email address. The authentication provider is the default, **thy-one**
- Local users does not need to specify an authentication provider.

```
Please enter username for tenant "example":  
admin@example.com Please enter password:  
*****
```

```
Thycotic One authentication provider name (default thy-one): thy-one
```

That completes setup. You can begin using the IBM Security Verify Privilege DevOps Vault Command Line Interface to [create your first secret](#)

## CLI Secrets Examples

### Create a Secret

#### Using a file

Here is an example of JSON that could be made a Secret. The JSON is arbitrary, so you can set any number of fields (key-value pairs).

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

To create a Secret, open a text editor and create and save a file (.json) similar to the example above.

Create the Secret and specify the path to its storage location:

NOTE: Every Secret correlates uniquely with a specific path that describes the location of the Secret. 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.

#### 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@example.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@example.com",
"path": "servers:us-east:server01",
"version": "0"
}
```

Files may also be used to enter attributes `--attributes` or a description `--desc`

### Direct Command

Instead of using a file, the data can be entered as part of the command:

#### Linux:

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

#### Powershell:

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

#### CMD:

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

## Outputs:

```
{
  "attributes": null,
  "created": "2019-01-03T23:11:48Z",
  "createdBy": "users:thy-one:admin@example.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@example.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@example.com",
"data": {
  "host": "server01",
  "password": "secretp@sssword",
  "username": "administrator"
},
"description": "",
"id": "c5239a6c-422e-4f57-b3a6-5167656af852",
"lastModified": "2020-01-17T15:38:49Z",
"lastModifiedBy": "users:thy-one:admin@example.com",
"path": "servers:us-east:server01",
"version":
"0" }
```

## If you 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@example.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@example.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 -bf data.password secretp@ssword
```

## 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@example.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@example.com",
  "path": "servers:us-east:server01",
  "version": "1"
}
```

## Creating Users

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

For this quick-start guide, as the initial admin, we create two users - a local User and a Thycotic One User.

First, a local Privilege DevOps Vault User, designated with their email address `local@example.com` is created. For local users, an email address is not required.

```
thy user create --username local@example.com --password BadP@ssword
```

Second, a Thycotic One User is created in Privilege DevOps Vault. Here a valid email address is required as the username.

```
thy user create --username thycoticone@example.com --provider thy-one
```

The user receives an email with a link to both confirm their email address and setup a password.



Once the Thycotic One User clicks that link and sets a password, is ready to authenticate to Privilege DevOps Vault.

## Local User and Thycotic One User Authentication

The local and Thycotic One users can then, on their own machines, download the CLI and start the `thy init` process. The admin must provide the local user with their password, and both of them with the Privilege DevOps Vault 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) Thycotic One (federated)
- (4) AWS IAM (federated)
- (5) Azure (federated)
- (6) GCP (federated)
- (7) OIDC (federated)

The local user selects (1) and enter their username and password. The Thycotic One user selects (3) and enter their email, Thycotic One password, and for the provider name simply hit enter to default to *thy-one*.

The local user must change their password immediately as a best practice because the admin knows it and had to transfer it to them somehow. The command is: `thy auth changepassword`

At this point, the users are created and able to authenticate to Privilege DevOps Vault (they can confirm with the command `thy auth` and get a token), however, they do not have permission to access anything yet because Privilege DevOps Vault defaults to *deny all*. In the next step, the admin creates policies granting permission to these users.





## Provide Users Access to Secrets

Assuming we have two secrets, each located at:

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

And two users:

`local@example.com` and `thycoticoneuser@example.com`

Our goal is to create policy to allow:

- both users access to `servers:us-east:server01`
- `local@example.com` to have access to `servers:us-east:production:server01`
- `thycoticoneuser@example.com` to be denied access to `servers:us-east:production:server01`

## Create a Group

Optionally, we can put these Users in a Group with two commands. The first command creates the group:

```
dsv group create --groupname firstgroup
```

The second command puts the Users in the Group

```
dsv group add-members --group-name firstgroup --data
```

```
'{"memberNames":["local@example.com","thyone:thycoticoneuser@example.com"]}'
```

## Create Policy for 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
```

Where *path* starts with **secrets:** followed by the secret path.

NOTE: That *resources* are not specified separately, but they 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`, `list`, `assign` is allowed.

*subjects* are the Users that are getting access to the secrets.

Note: The local user does not need a prefix, but any federated users, in this case Thycotic One, refers to the name of the auth provider. The default auth-provider name for Thycotic One in Privilege DevOps Vault is **thy-one**

*effect* is allow

The resulting policy looks like this if you read it using the command `dsv policy read secrets:servers:us-east -e yaml`

```
path: secrets:servers:us-east
permissionDocument:
- actions:
- <.*> conditions: {} description: Allow Policy
  effect: allow
id:
```

```

xxxxxxxxxxxxxxxxxxxxx
xx
  meta: null
resources:  -
secrets:servers:us-
east:<.*>  subjects:
- groups:firstgroup
version: "0"

```

This policy now enables both Users (`local@example.com` and `thycoticoneuser@example.com`) to gain full access to all secrets located at the path `servers:us-east` and below.

## Create Policy for Deny Access

If we decided that the `thycoticoneuser@example.com` must no longer have access to the secrets at `servers:us-east:production` we can write another policy to deny that access. The command looks like this:

```

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

```

The resulting policy looks like this if you read it using the command `dsv policy read secrets:servers:us-east:production -e yaml`

```

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

```

Now `local@example.com` has access to everything at `servers:us-east` and below, including `servers:us-east:production`. However, `thycoticoneuser@example.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/Policy](#) in this documentation.

## CLI Reference

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

CLI commands commonly act on these object types:

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

This Reference complements the separately maintained IBM Security Verify Privilege DevOps Vault [API Reference](#).

## CLI Command Syntax

With few exceptions, CLI commands follow a simple syntax:

```
dsv (object) (command) (flags and parameters)
```

For example, in `dsv role create`, `role` is the object of the command `create`. Some parameters and flags apply only to some commands. Privilege DevOps Vault also includes output modifiers for filtering and formatting responses to commands.

## Objects

Object	Syntax	Definition
auth	auth	authenticate to the vault or display the current access token
cli-config	cli-config	manage the CLI authentication file
client	client (<client-id> * --client-id)	manage client credentials for application vault access
config	config	manage the top level configuration document for the admin policy and authentication providers
eval	eval	check the value of a command line flag or variable
group	group (<group-name> * --groupname)	manage collections of Users uniformly by placing them in a managed Group
init	cli-config init or init	initialize Privilege DevOps Vault on first run
pki	pki	manage certificate issuance
policy	policy (<path> * --path * -r)	manage policies on permissions for Secrets, Roles, Users, and other entities in the vault
role	role (<name> * --name * -n)	manage Roles
secret	secret (<path> * --path * -r)	create, update, and retrieve Secrets from the vault
siem	siem	manage endpoints for pushing audit logs
user	user (<username> * --username)	manage Users
whoami	whoami	display the currently authenticated User

## Workflows for Creating or Updating Objects

For many objects, if the command is `create` or `update`, then adding no flags starts a workflow.

A workflow is a series of questions that guides the user through the creation or update process. Workflow supported objects include:

- `dsv init` (This command is only done with a workflow)

- 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 must 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, whether to beautify the JSON output of a Secret read.

```
dsv secret read --path example/bash-json --beautify
```

## 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

Privilege DevOps Vault offers global flags that combine with most commands to format or redirect output.

- `--encoding, -e` specify the output format as either JSON or YAML
- `--beautify, -b` beautify JSON or YAML output
- `--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

## Encoding and Beautify

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

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 -b -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 Privilege DevOps Vault 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 create an environment variable named `MYSecret` that store the Secret data. To view the data, 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 Privilege DevOps Vault 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 <b>VI</b> , <b>nano</b> , or <b>Notepad</b>
update	modify a Secret, with <code>--data</code> , <code>--attributes</code> and <code>--desc</code> flags to modify selected portions only, and a Boolean <code>--overwrite</code> flag to control whether the <code>--data</code> flag's content overwrites or merges with extant data object fields
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'
```

#### 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 must 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

*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@example.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@example.com",
  "path": "servers:us-east:server01",
  "version":
  "12" }
```

This Bash command only changes 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@example.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@example.com",
  "path": "servers:us-east:server01",
  "version": "13"
}
```

The flag `--overwrite`, if added to the above command wipes-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 or attribute

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 are displayed 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)

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 returns the first 10 plus a cursor. To obtain the next two results, use this command: `dsv secret search -q admin --limit 10 --cursor AFSDFS...DKFJLSDJ=`

Cursors may be lengthy:

```
dsv secret search -q resources --limit 10 --cursor
eyJpZCI6ImEwOTFjOWIzLWE4MmQtNGRiYy1hYThiLTlxMDY0NDZhZjA3MSIsInBhdGgiOiIiLCJ2ZXJzaW9uIjoidi
ljdXJyZW50IiwidHlwZSI6IiIsImxhdGVzdC I6MH0=
```

## Describe

Use *describe* to show only metadata; you do 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:)

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

This example sends 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

```
\$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
```

## 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

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

## Delete

To *delete* a Secret simply specify the path.

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

When you delete a Secret, it is no longer usable. However, with the soft delete capacity of Privilege DevOps Vault, you have 72 hours to use the `restore` command to undelete the Secret. After 72 hours, the Secret is no longer retrievable.

If 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` undeletes 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 rolls 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 do not lose those three versions; they 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 makes with any other Secret.

## User

For Privilege DevOps Vault, 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 is 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 has the username *aws-dev:test-admin* while the local User with username *test-admin* does not have a qualifier, so its username is just *test-admin*.

### Commands that Act on Users

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

## Examples

### Changepassword

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 is 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 *changepassword* command within the CLI fails.

### Create

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

Parameter	Content
<code>--username</code>	local username; required; supports local authentication by username and password; need not match that used by a federated authentication provider (if present)
<code>--password</code>	password for local authentication by username and password
<code>--provider</code>	matches the <i>name</i> attribute of the authentication provider in the <i>settings</i> section of the config
<code>--external-id</code>	identifier recognized by third-party federated authentication providers, such as AWS or ARN

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::000000000000: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.

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

### Output:

```
[
  {
    "externalId": "arn:aws:iam::000000000000: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 get --username test-admin
```

## Delete

The *delete* command removes 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 is no longer usable. However, with the soft delete capacity of Privilege DevOps Vault, you have 72 hours to use the *restore* command to undelete the User. After 72 hours, the User is no longer retrievable.

If 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
update	update a Group
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

This example command creates a Group named **admins** from a file **data.json** containing `{"groupName": "admins"}` (or same with singlequote 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 }
```

This example creates 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:

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

## Find Group Membership

To see what Groups the user Billy belongs to, 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 assigns 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, follow this example:

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

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

If 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 Privilege DevOps Vault, 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
<code>--desc</code>	description of the Role
<code>--name</code>	name of the Role
<code>--provider</code>	matches the <i>name</i> attribute of the authentication provider in the <i>settings</i> section of the config
<code>--external-id</code>	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: `dsv role search devadmin`

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

```
dsv role search --query us-east/server02 --limit 2 --cursor  
eyJpZCI6ImZmZjZjODUxTjJ2ZXJzaW9uIjo50IiwidHiJ9
```

## 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 removes Roles.

Provide a Role name and delete the Role:

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

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

If 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 User in the vault
search	find clients by Role name
read	read a client's details
delete	delete a User 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 includes a *clientId* and *clientSecret* suitable for use during CLI installation, or within REST calls to authenticate as the Role assigned to the *clientId*.

```
{  
  "clientId": "a59d37bf-4028-4eb9-9df4-6f1fea7d9298",  
  "clientSecret": "rV7l8l77DDwTLkdzWkL18UF9blycz3r9yfRhQTYICFc",  
  "role": "app-role"  
}
```

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.

### 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 a59d37bf-4028-4eb9-9df4-6f1fea7d9298
```

### Delete

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

```
dsv client delete --client-id a59d37bf-4028-4eb9-9df4-6f1fea7d9298
```

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

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

## Bootstrapping

There are times when machines or applications require access to Privilege DevOps Vault 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 is created and returned. The URL is no longer valid after the initial use, so if the intended machine or application gets an error "url already used" then there must be an alarm to investigate.

First create the Client ID and URL:

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

Where "role" is a Role created earlier and is attached to a Policy to provide the proper permissions. "--url" is the flag that tells Privilege DevOps Vault 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 timeframe, then it becomes invalid.

The result looks something like this:

```
"clientId": "5f1761dd-95ac-479f-a386-f9c379055b04",
"created": "2020-09-29T13:39:31Z",
"createdBy": "users:admin@example.com",
"id": "2f375a20-a670-4843-8b78-502649bc668e",
"role": "bootstraptest",
"url": true,
"urlPath": "https://company.secrestvaultcloud.com/v1/clients/bootstrap/5f1761dd-95ac-479f-a386-f9c379055b04",
"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/5f1761dd-95ac-479f-a386-f9c379055b04
```

With a result containing the Client Secret:

```
"id": "2f375a20-a670-4843-8b78-502649bc668e",
"clientId": "5f1761dd-95ac-479f-a386-f9c379055b04",
"clientSecret": "r_jqAZz6zs_Toqidv-Paz8wWe9OoP9HyjzRan7t7bc4",
"role": "bootstraptest",
"url": true,
"accessed": "2020-09-29T13:45:21Z",
"created": "2020-09-29T13:39:31Z",
```



```
"createdBy": "users:admin@example.com"
```

If the URL is accessed a second time, then the response contains: `"code":400,"message":"url has already used"`

## Policy

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

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@example.com id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx lastModified: '2019-
09-24T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.com
path: secrets:servers:us-west
permissionDocument:
- actions:
- read conditions: {}
  description: ''
  effect: allow id:
  xxxxxxxxxxxxxxxxxxxxxxxx
  meta: resources:
    - secrets:servers:us-
      west:<.*> subjects:
- groups:west
  adminsversion: '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, that is 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, system:log) to a colon delimited path to the resource.
subjects	the Policy provides authorization to these entiries. 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 is `["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 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** do 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 behaviour.

- 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 starts step-by-step questions to guide you through the process. However, in the following examples, the direct command is 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:<.>*`.

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@example.com id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxxx lastModified: '2020-
07-16T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.com
path: secrets:servers:us-east-1
permissionDocument:
- id: xxxxxxxxxxxxdescription: Developer Policy.
subjects: -
users:<developer1@thycotic.com|developer2
@thycotic.com> actions:
- "<read|delete|create|update|share>"
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@example.com      id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx lastModified: '2020-
07-16T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.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

**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@example.com id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx lastModified: '2020-
07-16T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.com
path: roles:dev-role
permissionDocument: - id:
xxxxxxxxxxxx description:
Limited Role Assignment Policy.
subjects:
- users:developer@thycotic.com
- roles:onboarding-roleactions: - 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 Admin 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 runs to create the policy is:

```
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:`

When User1 is authenticated and needs to access the secret, the command is

```
dsv home read databases/mongo/primary'
```

When User2 is authenticated and needs to access the secret, the command is

```
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@example.com id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx lastModified: '2020-
07-16T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.com
path: secrets
permissionDocument:
- actions:
- list conditions: {}
description: secret
search
  effect:
  allow id:
  xxxxxxxxxxxxxx
meta: null
resources:
- secrets
subjects:
- groups:adminsversion:
"0"
```

Note: Searching secrets only enables the users to see the path, but not the actual data in the secret. That requires 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@example.com id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx lastModified: '2020-
07-16T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.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-roleversion: "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@example.com id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxxx lastModified: '2020-
07-16T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.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@example.com id:
xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxxx lastModified: '2020-
07-16T20:13:53Z' lastModifiedBy:
users:thy-one:admin@example.com
path: audit
permissionDocument:
- actions:
- list
  conditions:
  {}
description:
"" effect:
allow id:
xxxxxxxxxxxxx
meta: null
resources:
- audit
subjects:
- users:developer1@thycotic.comversion: "0"

```

## Read System Logs

**Case:** A user needs to be able to read the application log messages

## **Solution:** Add a policy for the system:log resource path

The command to create this policy is `dsv policy create system:log --actions list --resources system:log --subjects users:developer1@thycotic.com`

```
created: '2020-06-24T18:12:26Z'  
createdBy: users:thy-one:admin@example.com id:  
xxxxxxxx-xxxx-xxxx-xxxx-  
xxxxxxxxxxxx lastModified: '2020-  
07-16T20:13:53Z' lastModifiedBy:  
users:thy-one:admin@example.com  
path: system:log  
permissionDocument:  
- actions:  
- list conditions: {}  
  description: "" effect:  
  allow id: xxxxxxxxxxxx  
  meta: null resources: -  
  system:log subjects:  
- users:developer1@thycotic.co  
  mversion: "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@example.com id:  
xxxxxxxx-xxxx-xxxx-xxxx-  
xxxxxxxxxxxx lastModified: '2020-  
07-16T20:13:53Z' lastModifiedBy:  
users:thy-one:admin@example.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"
```



## Admin Policy and Auth Providers

In this section we

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

### Commands that Act on Policies

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
delete	delete a configuration
restore	restore a deleted configuration (if within 72 hours of deletion and not hard deleted)

### 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 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@example.com",
  "permissionDocument": [
    {
      "actions": ["<.*>"],
      "conditions": {},
      "description": "Default Admin Permissions",
      "effect": "allow",
      "id": "bml7jee33mlc72u313tg",
      "meta": null,
      "resources": ["<.*>"],
      "subjects": ["users:<thy-one:admin@example.com>"]
    },
    {
      "actions": ["<.*>"],
      "conditions": {},
      "description": "Default Deny Home Permissions",
      "effect": "deny",
      "id": "bsd72rfelvk72up3o1g",
      "meta": null,
```

```

    "resources": ["home:<.*>"],
    "subjects": ["users:<thy-one:admin@example.com>"]
  }
],
"tenantName": "company",
"version": "1"
}

```

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

In keeping with best practices, you must set up a less privileged User policy for routine use. The highly privileged initial Admin account must 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 Privilege DevOps Vault.

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 he Home secrets (API only in Beta)

## Edit

NOTE: IBM Security recommends against changing the Default Admin Policy other than to add a User as a back-up admin. Even then, best practices are 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: IBM Security 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 --path us-east/server02 --data '{"something":"value"}'
```

or

```
dsv config update --path us-east/server02 --data @configfilename.json
```

## Grant Admin Access Rights to All Home Vaults

If it is required 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 opens the OS default editor and you can modify the *effect* field.

## Add an Authentication Provider

The general command is:

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

in which:

- name is the friendly name used in Privilege DevOps Vault 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
  - 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-project-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 only has a Thycotic One connection

```
{
  "created": "2019-09-18T18:38:49Z",
  "createdBy": "",
  "id": "bm17jee33mlc72u313u0",
  "lastModified": "2020-05-10T02:25:04Z",
  "lastModifiedBy": "users:admin@example.com",
  "name": "thy-one",
  "properties": {
    "baseUri": "https://thycotic-one-sscdev-dev-eastus-web01.azurewebsites.net",
    "clientId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx",
    "clientSecret":
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  },
  "type": "thycoticone",
```

```
    "version": "1"
  },
```

## 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 Privilege DevOps Vault, they are given access to their own Home vault without an explicit policy granting access.

The Home value lists a path like "users:" Privilege DevOps Vault determines which username based on whomever authenticated. So if joesmith@example.com authenticates, then creates a Home value, that value is 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'
```

### 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 must 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@example.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@example.com",
  "path": "users:user@example.com:secret1",
  "version":
"12" }
```

This Bash command only changes the value for `host` in the data section.

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

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

The flag `--overwrite`, if added to the above command wipes-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:)

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

This example sends 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

```
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` Sets the maximum number of search results that 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 returns the first 10 plus a cursor. To obtain the next two results, use this command:

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

Cursors may be lengthy:

```
dsv secret search -q resources --limit 10 --cursor  
eyJpZCI6ImEwOTFjOWIzLWE4MmQtNGRiYy1hYThiLTYxMDY0NDZhZjA3MSIsInBhdGgiOiIiLCJ2ZXJzaW9uIjoidi  
1jdXJyZW50IiwidHlwZSI6IiIsImxhdGVzdC I6MH0=
```

## 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 is no longer usable. However, with the soft delete capacity of Privilege DevOps Vault, you have 72 hours to use the `restore` command to undelete the Secret. After 72 hours, the Secret is no longer retrievable.

If 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
```



## Authentication

Privilege DevOps Vault supports several authentication methods.

### Password

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

Privilege DevOps Vault 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 Privilege DevOps Vault 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.

Routine activities associated with the client credentials authentication method include:

- creating a new Role
- adding the new Role to the Privilege
- DevOps Vault 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.

### Third Party Authentication

Besides ThycoticOne, IBM Security Verify Privilege DevOps Vault works with third party authentication providers, including:

**AWS IAM:** Privilege DevOps Vault 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:** Privilege DevOps Vault uses the assigned Azure Managed Service Identity (MSI). See **Authentication: Azure**

**GCP Service Accounts:** Privilege DevOps Vault 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: Azure**

**OIDC Provider** Privilege DevOps Vault connects to Thycotic One, which in-turn may connect to any OIDC provider. See **Authentication: OIDC**

## Profiles

On initial configuration, your IBM Security Verify Privilege DevOps Vault config has just one profile with the choices you specified for credentials storage, authentication type, and cache strategy for Secrets.

However, Privilege DevOps Vault 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 Privilege DevOps Vault uses by default.

### Add a Profile to a Config

Privilege DevOps Vault 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 is 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 tries 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 must 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@example.com
```



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

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, 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 is *aws-dev:test-admin*.

```
<snip>
- actions:
- <.*>
- conditions: {}
  description: Default Admin Policy
  effect: allow
  id: xxxxxxxxxxxxxxxxxxxxxxxx
  meta: null
  resources:
  - <.*>
  subjects:
- users:<aws-dev:test-admin|admin@example.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) Thycotic One (federated)
(4) AWS IAM (federated)
(5) Azure (federated)
(6) GCP (federated)
(7) OIDC (federated)
```

Privilege DevOps Vault prompts 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 Privilege DevOps Vault and access data.

```
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 Privilege DevOps Vault 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 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, 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@example.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@example.com
id: xxxxxxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z" lastModifiedBy:
users:thy-one:admin@example.com
name: thy-one properties:
baseUri:
https://login.thycotic.com/
clientId:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxx
  clientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
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 Privilege DevOps Vault to reference this policy
- 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, use:

```
dsv config auth-provider <name> read -e yaml where the example name we use here is azure-prod
```

The readout looks similar to this:

```
created: "2019-11-12T18:34:49Z"
createdBy: users:thy-one:admin@example.com
-id: xxxxxxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-18T03:58:15Z" lastModifiedBy:
users:thy-one:admin@example.com
name: azure-prod properties:
tenantId: xxxxxxxxxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxxx
type: azure
version: "0"
```

## Azure User Assigned MSI Example

First you need to configure the User that corresponds to an [Azure User Assigned MSI](#).

The username is a friendly name within Privilege DevOps Vault. 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-
xxxxxxxxxxxxxxxxxxxxx/resourcegroups/build/providers/Microsoft.ManagedIdentity/us
erAssignedIdentities/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, 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 is *azure-prod:test-api*.

```
<snip>
-
actions
:
- <.*> conditions: {}
  description: Default Admin Policy
  effect: allow
id:
xxxxxxxxxxxxxxxxxxxxx
xx
  meta:
  null
  resource
  s: -
  <.*>
  subjects
  :
  - users:<azure-prod:test-api|admin@example.com>
<snip
```

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.

```
```BASH
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) Thycotic One (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, 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 is *azure-prod:identity-rg*.

```
<snip>
-
actions
:
- <.*> conditions: {}
  description: Default Admin Policy
  effect: allow
id:
bgn8gjei66jc7148d9
i0
  meta:
null
resource
s: -
<.*>
subjects
:
- users:<azure-prod:test-api|admin@example.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.

```
```BASH
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) Thycotic One (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)

IBM Security Verify Privilege DevOps 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.

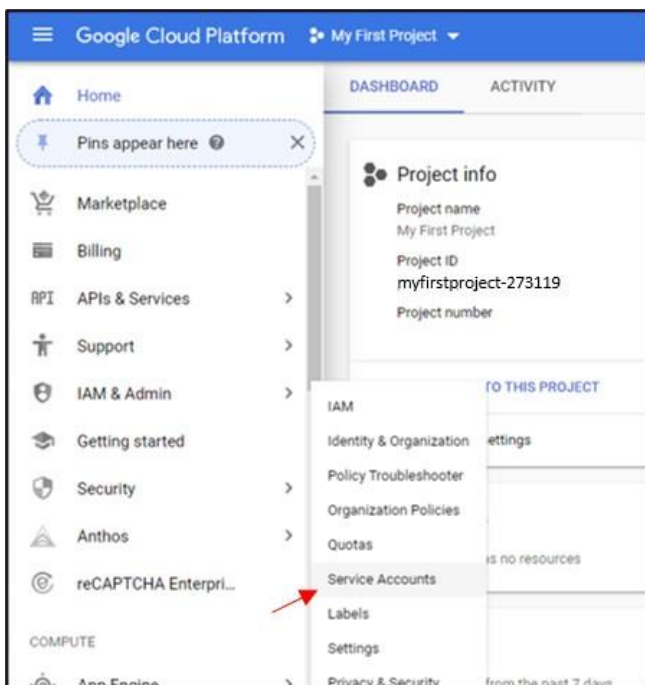
### Google Service Account Authentication

To setup GCP authentication using service accounts in Privilege DevOps Vault, a GCP service account must be provided that Privilege DevOps Vault 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 use the GCP Console.

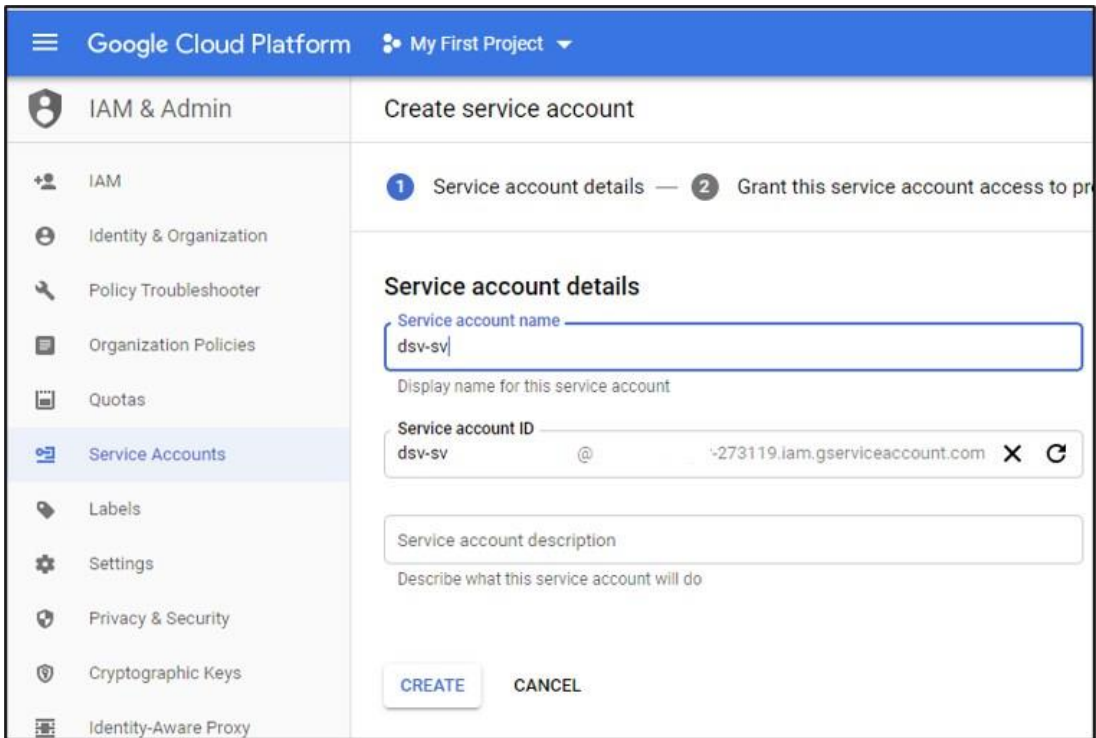
### GCP Service Account Setup

In the GCP Console Home page, go to your project, hover **IAM & Admin**, and then click **Service Accounts**.

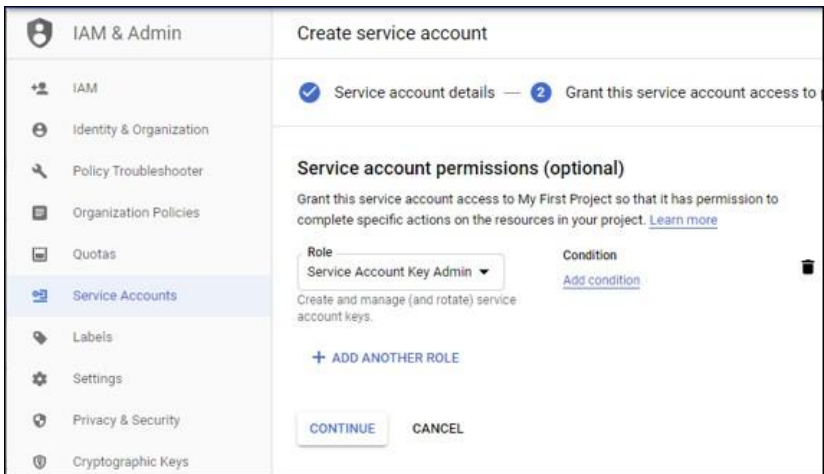


At the top, click **CREATE SERVICE ACCOUNT**.

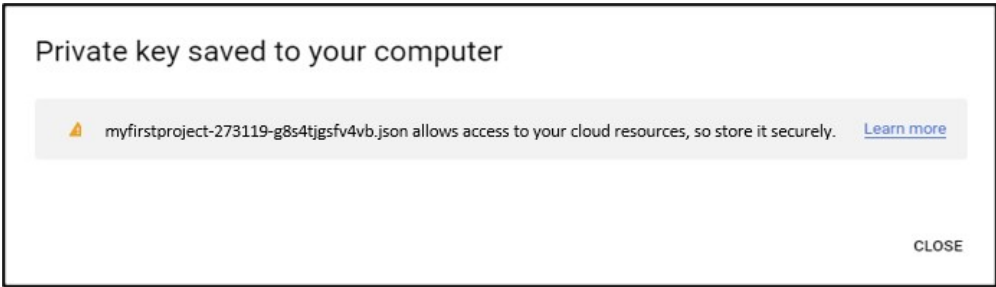
For the first step, enter an account name. We use `dsv-svc` in this example. Click **CREATE**.



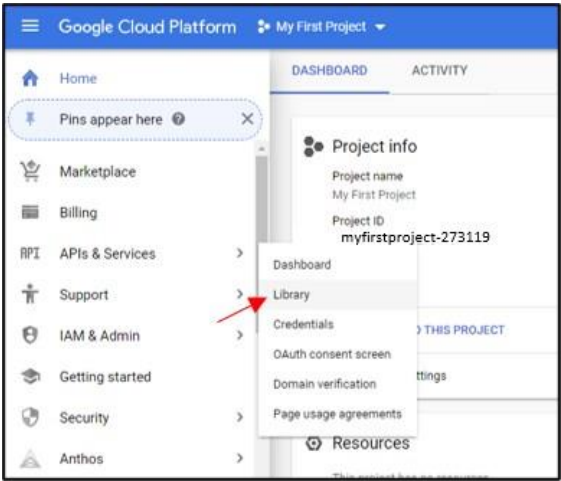
In the second step, click the dropdown arrow in the **Select a role** box, then type `service account key admin` in the filter and select **Service Account Key Admin**. Then click **Continue**.



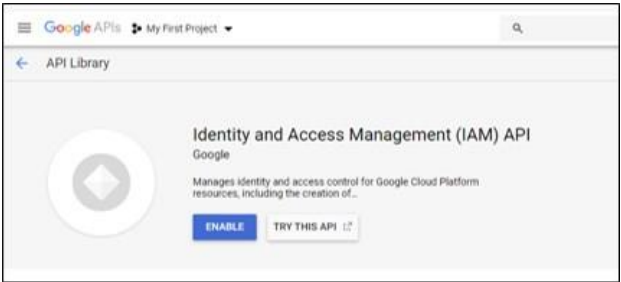
In the third step, click **CREATE KEY** and when the option to generate a file slides in from the right, select **json** and click **CREATE**. A file is downloaded that has all the information needed to setup the Privilege DevOps Vault authentication provider.



The Google API for IAM must be enabled. To do this in the Google Console, go to the relevant project and on the left nav, 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**.



### Privilege DevOps Vault Authentication Provider Setup

Go back to the terminal (IBM Security Verify Privilege DevOps 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:





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, 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 is `gcloud:gcp-test`.

```
<snip>
-
actions
:
- <.*> conditions: {}
  description: Default Admin Policy
  effect: allow
  id: xxxxxxxxxxxxxxxxxxxxxxxx
  meta:
  null
  resource
s: -
<.*>
subjects
:
- users:<gcloud:gcp-test|admin@example.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) Thycotic One (federated)
- (4) AWS IAM (federated)
- (5) Azure (federated)
- (6) GCP (federated)
- (7) OIDC (federated)



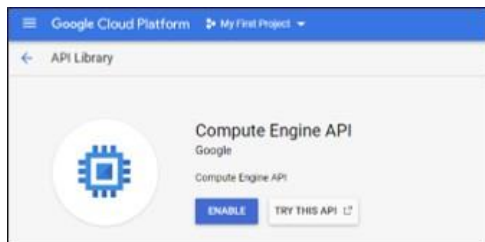
Run `dsv auth` to verify authentication. A token is displayed.

Run `dsv secret read <path to any secret>` to verify secret access.

## Google Compute Engine (GCE) Metadata Authentication

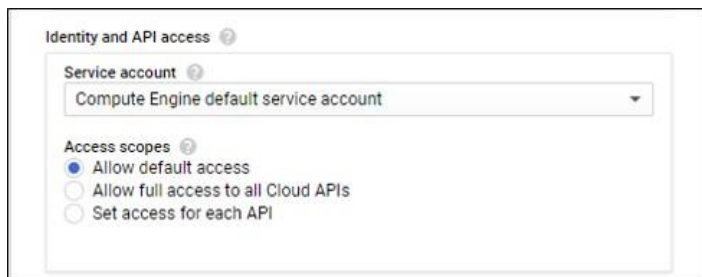
The idea behind GCE Metadata authentication is to enable a GCE instance to gain access to IBM Security Verify Privilege DevOps Vault.

In this example we assume you have created a Linux Google Compute Instance and have the Google Compute Engine API enabled.

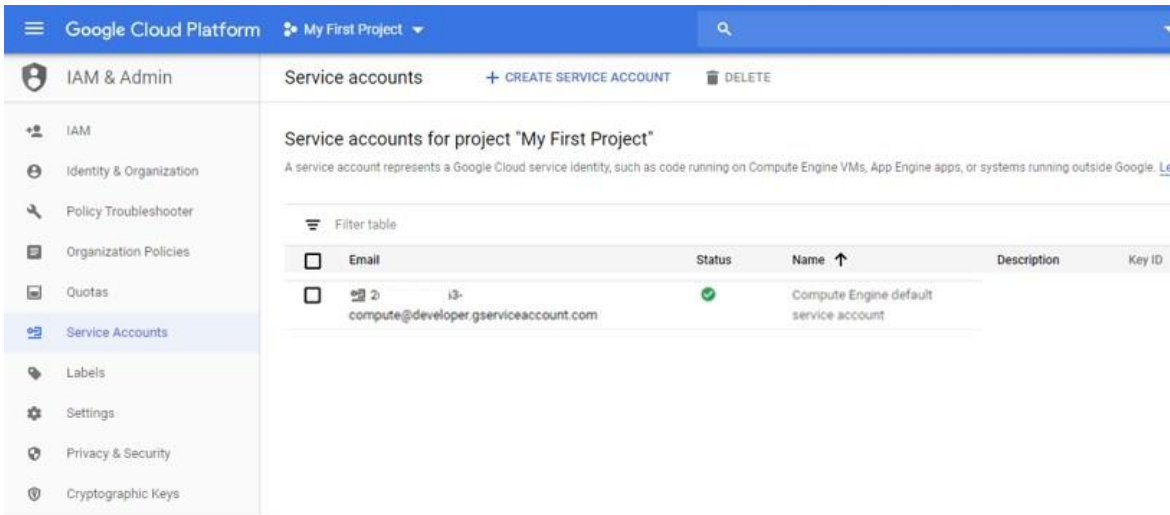


It is further assumed that the **Compute Engine default service account** is used. However, you can assign a different service account to the Compute instance if desired.

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 is "Compute Engine default service account". Copy and store the email for later.



## Privilege DevOps Vault GCE Authentication Provider setup

Using any computer with Admin Privilege DevOps Vault access, we now want to setup the Privilege DevOps Vault Authentication Provider. Create a file named 'auth-gcp.txt' in the following format and substituting your 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. To view the resulting addition to the config file, you use:

`dsv config auth-provider <name> read -e yaml` where the example name we 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@example.com
id: xxxxxxxxxxxxxxxxxxxxxxxx
lastModified: "2020-05-
18T03:58:15Z" lastModifiedBy:
users:thy-one:admin@example.com
name: gcloud-gce
properties: projectId:
```

```
myfirstproject-xxxxxxx
type: gcp version: "0"
```

## Privilege DevOps Vault GCE Metadata Service Account/ Privilege DevOps Vault 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@example.com",
  "externalId": "2XXXXXXXXXX3-compute@developer.gserviceaccount.com",
  "id": "19709b4e-2a13-4164-a930-81997b568036",
  "lastModified": "2020-04-09T12:59:44Z",
  "lastModifiedBy": "users:thy-one:admin@example.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, 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 is `gcloud-gce:gce-test`.

NOTE: Adding a user to the admin policy is not security best practices. This is for example purposes only. Ideally, 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:
xxxxxxxxxxxxxxxxxxxxxx
xx
  meta:
null
resource
s: -
<.*>
```

```
subjects
:
- users:<gcloud-gce:gce-test|admin@example.com>
<snip>
```

## GCE Authentication

SSH into the GCE and download the latest Privilege DevOps Vault CLI from this website [DSV CLI](#)

For example, `curl https://dsv.thycotic.com/downloads/cli/1.8.0/thy-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) Thycotic One (federated)      (4) AWS IAM
        (federated)
    (5) Azure (federated)
    (6) GCP (federated)
    (7) OIDC (Federated)
```

Run `dsv auth` to verify authentication. A token is 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 (that is a virtual server) authenticate to Privilege DevOps Vault, there is 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 can authenticate to Privilege DevOps Vault.

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 enables the ability to pull an image from GCP registry. In this example, we created a service account named `dsv-gce`

## Privilege DevOps Vault Authentication provider

Using any computer with Admin Privilege DevOps Vault access, we now want to setup the Privilege DevOps Vault Authentication Provider Create a file named 'auth-gcp.txt' in the following format and substituting your GCP .

```
{
"name": "gcloud-gce",
"type": "gcp",
"properties": {
  "ProjectId": "myfirstproject-273119"
}
}
```

Run `dsv config auth-provider create --data @auth-gcp.txt` to implement the Authentication Provider.

## Privilege DevOps Vault 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 get a response like this:

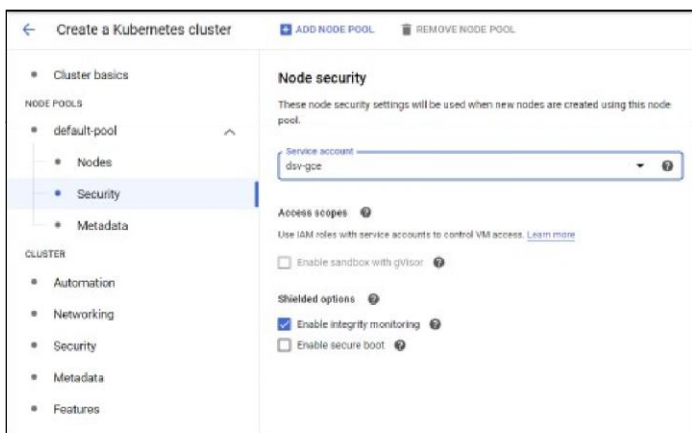
```
{
  "created": "2020-04-09T12:59:44Z",
  "createdBy": "users:thy-one:admin@example.com",
  "externalId": "dsv-gce@gcp-project-id.iam.gserviceaccount.com",
  "id": "19709b4e-2a13-4164-a930-81997b568036",
  "lastModified": "2020-04-09T12:59:44Z",
  "lastModifiedBy": "users:thy-one:admin@example.com",
  "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 enables 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 nav, select **defaultpool** then **Security** where you 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 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 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**

The screenshot shows the Google Cloud Kubernetes Engine console. On the left is a navigation menu with options like Clusters, Workloads, Services & Ingress, Applications, Configuration, Storage, and Object Browser. The main area is titled 'Kubernetes clusters' and contains a table of clusters. One cluster named 'cluster-dsv' is highlighted, and a red arrow points to its 'Connect' button. A modal window titled 'Connect to the cluster' is open in the foreground. It offers two options: 'Command-line access' with a terminal window showing the command `$ gcloud container clusters get-credentials cluster-dsv --zone us-central1-c --project hallowed-tape-276802` and a 'Run in Cloud Shell' button (indicated by a red arrow), and 'Cloud Console dashboard' with an 'Open Workloads dashboard' button. An 'OK' button is at the bottom right of the modal.

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 looks 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
    } else {
        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/serviceaccounts/default/ident
ity", 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
} else {
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 to escape out. Then provide executable privileges.

```
chmod +x main.go
```

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 to escape out. Then provide executable privileges.

```
chmod +x Dockerfile
```

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 to escape out. Then provide executable privileges.

```
chmod +x k8.yml
```





External Authentication Provider Settings

Description  
Azure AD

Provider URL

Client ID

Secret

Callback URL

Enabled

Save Cancel

## 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 IBM Security cloud manager portal. Follow the instructions to add these URL's to the OAuth consent screen.

- Dashboard
- Library
- Credentials**
- OAuth consent screen
- Domain verification
- Page usage agreements

Name \*

DSV

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.



The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorized domains](#).

### Authorized JavaScript origins ?

For use with requests from a browser

URIs

[+ ADD URI](#)

### Authorized redirect URIs ?

For use with requests from a web server

URIs

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 must be set to `https://accounts.google.com`

The image shows a dialog box titled "External Authentication Provider Settings" with a close button (X) in the top right corner. The dialog contains several input fields:

- Description:** A text box containing "Google Identity".
- Provider URL:** A text box containing "https://accounts.google.com".
- Client ID:** A text box containing a blurred value followed by ".apps.googleusercontent.com".
- Secret:** A text box containing a blurred value.
- Callback URI:** A text box is partially visible at the bottom of the dialog.

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 Privilege DevOps Vault 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 Privilege DevOps Vault with the CLI.

### Organization Credential ✕

---

**Name**

**Post-Login Redirect URIs** +

✕

✕

**Post-Logout Redirect URIs** +

✕

**Credentials**

**Endpoint**

**Client Id**

 ♻️

Revoked

Save
Cancel

## Creating a User in Thycotic One and Privilege DevOps Vault

In order to login using OIDC, the user must exist in the external provider, Thycotic One, and in Privilege DevOps Vault.

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 Privilege DevOps Vault simultaneously, do the following steps:

1. In the Privilege DevOps Vault CLI run `dsv user create --username useremail@example.com --provider thy-one`
2. This creates a user record in Privilege DevOps Vault and syncs it to Thycotic One. The User gets 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 on 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) Thycotic One (federated)
- (4) AWS IAM (federated)
- (5) Azure (federated)
- (6) GCP (federated)
- (7) OIDC (federated)

When prompted for the authentication provider hit Enter to accept the default of `thy-one`

If you are on Windows or Mac OS the CLI automatically opens a browser to the Google login page, otherwise it prints out a URL that you can copy and paste into a browser to complete the process.

Login using your Google credentials and your browser redirects to `http://localhost:8072/callback`, the CLI is listening on that port and submits the returned authorization code to Privilege DevOps Vault to finish the login process.

Verify the login by running (omit the `--profile` flag if you overwrote your config):

```
dsv auth --profile profilename
```



## 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.

### Register an application

---


#### \* Name

The user-facing display name for this application (this can be changed later).

dsv 

#### Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (  only - Single tenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

[Help me choose...](#)

#### Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web  

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 is not 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 is part of your provider URL in the format <https://login.microsoftonline.com/{directory id}>

Home > dsv-sign-on


**dsv-sign-on**

Search (Ctrl+/) « Delete Endpoints

Display name : dsv-sign-on  
 Application (client) ID :   
 Directory (tenant) ID :   
 Object ID :

Welcome to the new and improved App registrations. Looking to learn how it's changed from App registrations (Legacy)? [Learn more](#)

**Call APIs**



Build more powerful apps with rich user and business data from Microsoft services and your own company's data

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**
12. When you sign into Thycotic one again you 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 IBM Security's Cloud Manager portal following the directions at **Authentication:OIDC**
2. Login 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 dropdown and the **OpenID Connect** radio button. Click **Create**

### Create a New Application Integration ✕

Platform

Sign on method

Secure Web Authentication (SWA)  
Uses credentials to sign in. This integration works with most apps.

SAML 2.0  
Uses the SAML protocol to log users into the app. This is a better option than SWA, if the app supports it.

OpenID Connect  
Uses the OpenID Connect protocol to log users into an app you've built.

7. On the resulting screen, provide an **Application name** and optional logo. Enter the IBM Security callback URL in the box labeled **Login redirect URIs**. Click **Save**.

## Create OpenID Connect Integration

### GENERAL SETTINGS

Application name

Application logo (Optional)

Requirements

- Must be PNG, JPG or GIF
- Less than 1MB

For Best Results, use a PNG image with

- Minimum 420px by 120px to prevent upscaling
- Landscape orientation
- Transparent background

---

### CONFIGURE OPENID CONNECT

Login redirect URIs

Logout redirect URIs

8. To the right of General Settings click **Edit**. Check the **Implicit (Hybrid)** box and it expands. 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` in. Click **Save**
10. Copy the Client ID and Client secret for entry into the IBM Security Cloud portal

General   Sign On   Assignments   Okta API Scopes

---

General Settings

**APPLICATION**

Application label: DevOps Secret Vault


Application type: Web


Allowed grant types:

- Client acting on behalf of itself
  - Client Credentials
- Client acting on behalf of a user
  - Authorization Code
  - Refresh Token
  - Implicit (Hybrid)
    - Allow ID Token with implicit grant type
    - Allow Access Token with implicit grant type

---

**LOGIN**


Login redirect URIs : <https://login.thycotic.com/signin-oidc/09e75fff-620e-4bfc-8266-d35afd9a10fe>

Logout redirect URIs 



Login initiated by: App Only

Initiate login URI: <https://portal.thycotic.com>

**Client Credentials** Edit

Client ID:  

Public identifier for the client that is required for all OAuth flows.

Client secret:   

Secret used by the client to exchange an authorization code for a token. This must be kept confidential! Do not include it in apps which cannot keep it secret, such as those running on a client.

## Retrieve the Issuer URL

11. In the second menu bar from the top, click **Sign On** and in the third box down, titled "OpenID Connect ID Token", take note of the URL by **Issuer** for entry into the IBM Security Cloud portal. It is generally something like <https://company.okta.com> or <https://company.oktapreview.com>

← Back to Applications



DSV

Active ▾



View Logs

General

Sign On

Assignments

Okta API Scopes

## Settings

### SIGN ON METHODS

The sign-on method determines how a user signs into and manages their credentials for an application. Some sign-on methods require additional configuration in the 3rd party application.

Application username is determined by the user profile mapping. [Configure profile mapping](#)

OpenID Connect

## Token Credentials

Edit

Signing credential rotation ⓘ

Automatic

## OpenID Connect ID Token

Edit

Issuer https://[redacted]okta.com

Audience 00a[redacted]z4x6

Claims Claims for this token include all user attributes on the app profile.

Groups claim type Filter

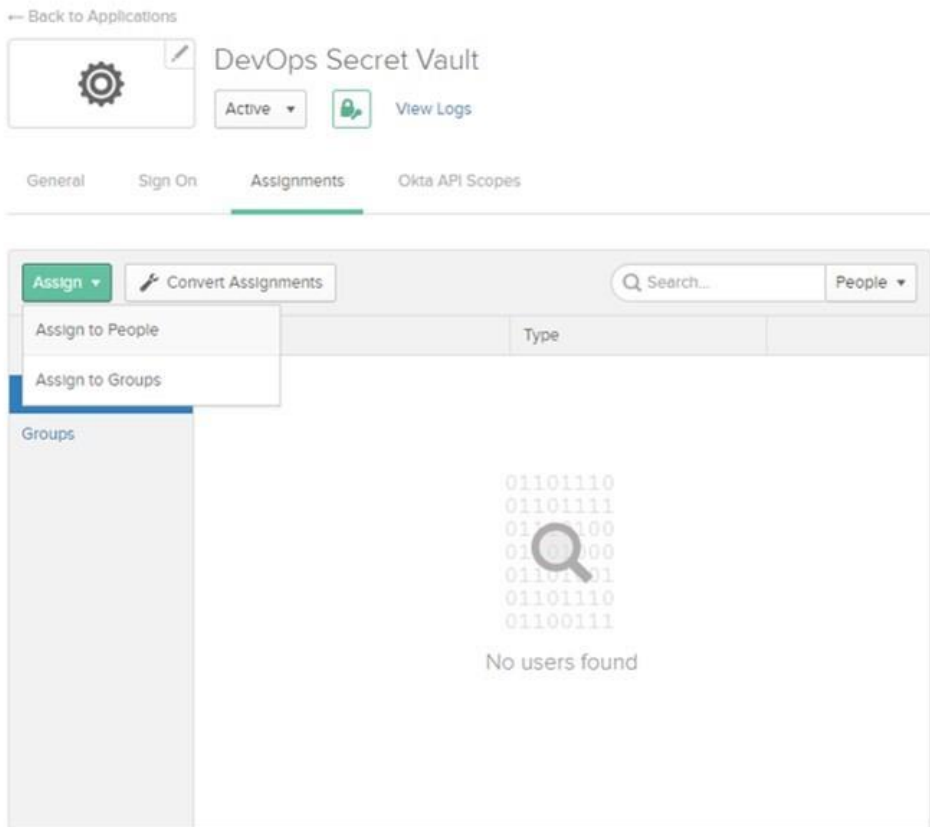
Groups claim filter ⓘ None

Using Groups Claim

## Add Okta Users and Groups to the Privilege DevOps Vault 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 use IBM Security Verify Privilege DevOps Vault. Of course, you can always come back, and add/remove people as needed.



### Finish the Connection on the Thycotic One side

14. Go back to the IBM Security Cloud Manager Portal where we started. Fill-in a Description and the issuer/provider URL from step 11.

15. Fill-in the Client ID and Client Secret from step 10.

16. Check the **Enable** box.

17. Click **Save**

## External Authentication Provider Settings ✕

**Description**

**Provider URL**

**Client ID**

**Secret**

**Callback URL**

Enabled

18. Click **Back to Organizations** 19. Click **Credentials**

20. Click **Edit** and a window pops-up

21. To the right of "Post-Login Redirect URIs" click the **+** and a new empty box appears. In this new box, type `http://localhost:8072/callback.`

Note: If you have already added this call back for another auth provider, then it is still there so you can skip these last steps (18-21).



## Organization Credential



### Name

DevOps Secrets Vault [REDACTED].secretsvaultcloud.com

### Post-Login Redirect URIs



https://[REDACTED].secretsvaultcloud.com/signin-oidc



http://localhost:8072/callback



### Post-Logout Redirect URIs



https://[REDACTED].secretsvaultcloud.com/signout-callback-oidc



### Credentials

#### Endpoint

https://login.thycotic.com/

#### Client Id

ct[REDACTED]3



Revoked

Save

Cancel

## Dynamic Secrets

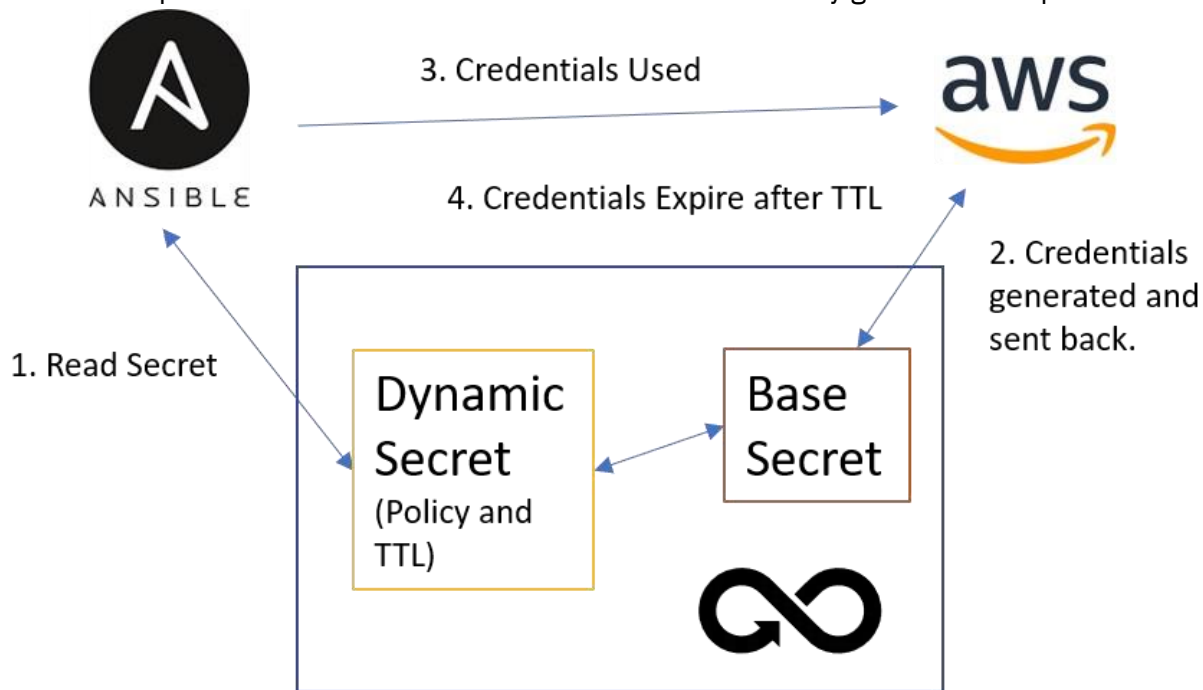
Dynamic Secrets are automatically generated at the time of request. This differs from the standard Secret store read request where the credentials remain the same until changed by a user. They can be used when you need to provide credentials to a user or resource, like a configuration tool, but the access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

Supported Types:

- AWS
- Azure
- GCP

## Linking

For Dynamic Secrets to be generated, they rely on a Base Secret stored in Privilege DevOps Vault 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
<code>linkConfig</code>	link type and path to the linked Secret.
<code>linkConfig.linkType</code>	The only valid value is "dynamic"
<code>linkConfig.linkedSecret</code>	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> --searchlinks`

## AWS Dynamic Secrets

AWS Dynamic Secrets generate a temporary access key, secret key, and session token. AWS security token service (STS) provides either `federate` or `assumeRole`. `federate` 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 Privilege DevOps Vault 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 must 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 are the intersection of the IAM User and the policy ARN specified on the Dynamic Secret. 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 Privilege DevOps Vault with the AWS IAM user access key, secret key, and region.

Create a file named `secret_root.json` substituting your values:

```
{
  "accessKey": "AIA2RAVTSMNW437LM",
  "region": "us-east-1",
  "secretKey":
  "SpN5Ipjvgepz0/q0ZNGmFhhLkUr+Uie5+D3CE" }
```

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 defaults to 900

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
```

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": "ASIAZTRAVTSMN5P6P",
    "expiration": "2020-02-06T18:49:17Z",
    "secretKey": "Is5L79Y1LgtOistJv+x0yVZ2/KLPWUUsUUj",
  }
}
```

```

    "sessionToken": "FwIv...Zggfj+6nbiT9IOrEw==",
    "ttl": 1200
  },
  "description": "",
  "id": "db38e569-5d7f-4ad8-954c-ac846d528947",
  "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 boto3.exceptions import ClientError

sess = boto3.Session(
    aws_access_key_id="ASIAZTRAVTSMN5P6P",
    aws_secret_access_key="Is5L79Y1LgtOistJv+x0yVZ2/KLPWUUsUUj",
    aws_session_token="FwIv...Ay93XTqVBGyeuodcw=="
)

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 looks 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 that user assumes 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 must 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 are 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 has permissions that intersect between the IAM user policy(ies) and the role policy(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 Privilege DevOps Vault with the AWS IAM user access key, secret key, and region.

Create a file named `secret_root.json` substituting your values:

```
{
  "accessKey": "AIA2RAVTSMNW437LM",
  "region": "us-east-1",
  "secretKey": "SpN5Ipjvgepz0/q0ZNGmFhhLkUr+Uie5+D3CE"
}
```

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, it defaults 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
```

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": "ASIAZTRAVBIVK5SLU",
    "expiration": "2020-02-06T18:49:17Z",
    "secretKey": "Xh/xqw5Ipjvgepz0i6un+ZUUsUUj",
    "sessionToken": "FwIv...Zggfj+TexEiLtE3h1R1Uv1lXCHzk5==",
    "ttl": 1200
  },
  "description": "",
  "id": "34fb64d7-18da-453d-9487-3d1c082ba372",
  "version": "0"
}
```

## Azure Dynamic Secrets

IBM Security Verify Privilege DevOps Vault relies on Azure service principals to provide Dynamic Secrets.

For Privilege DevOps Vault 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 Privilege DevOps Vault to perform API calls to Azure to query roles and create/delete service principals.

Attribute	Description
subscription_id	Required - The subscription ID holding the resources you wish to access using Azure Active Directory.
tenant_id	Required - The tenant ID for Azure Active Directory. Azure lists it in places as "Directory (tenant) ID"
client_id	Required - The OAuth2 client ID to connect to Azure. Azure lists it in places as "Application (client) ID"
client_secret	Required - The OAuth2 client secret to connect to Azure.
environment	Optional - The Azure environment. If not specified, Privilege DevOps Vault uses Azure Public Cloud.

Create a file named `secret_base.json` substituting your values:

```
{
  "subscriptionId": "6ca2adeb-7b44-4c7f-93fc-2d5b9729a8c1",
  "tenantId": "11f54b31-ffb9-42b5-8fda-76c734a7796c",
  "clientId": "4d95b358-079d-4d6d-85c4-943c0f1d91cd",
  "clientSecret": "tMQ5ZEP?.sj46e15123ba3b5b]"
}
```

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 Privilege DevOps Vault 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 Privilege DevOps Vault can either use a service principal that you have already setup (existing) or Privilege DevOps Vault 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 Privilege DevOps Vault 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. The dynamic Secret doesn't 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 defaults to 900

□

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": 360
}
```

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"
```

Now anytime you read the dynamic Secret, the data is populated with the temporary Azure access credentials.

```
dsv secret read --path azure/dynamic/api-account
```

Returns a result like:

```
{
  "id": "6e7de928-5027-4afb-bbff-b3ee59f9c24f",
  "path": "dynamic:azure:sp-static",
  "attributes": {
    "appId": "f81b3c6d-2ce9-47d4-ad2d-fef8390792a2",
    "appObjectId": "5fe218ee-cb58-4089-ac9f-b1b68971ad73",
    "linkConfig": {
      "linkType": "dynamic",
      "linkedSecret": "azure:base:api-account"
    },
    "roleName": "Contributor",
    "ttl": 360
  },
  "data": {
    "appObjectId": "5fe218ee-cb58-4089-ac9f-b1b68971ad73",
    "client_id": "f81b3c6d-2ce9-47d4-ad2d-fef8390792a2",
    "client_secret": "bfe6ac86-3671-4fd9-8f76-8f2e0f22495d",
    "role": "Contributor",
    "subscription_id": "6ca2adeb-7b44-4c7f-93fc-2d5b9729a8c1",
    "tenant_id": "11f54b31-ffb9-42b5-8fda-76c734a7796c",
    "ttl": 360
  },
  "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 is returned along with the task id that fired in the background for role assignment. You need to wait to use that temporary service principal or check via the Azure portal or via the Privilege DevOps Vault API (provided below)

Attribute	Description
roleName	Optional - If no "roleID" is assigned, Privilege DevOps Vault tries 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 Privilege DevOps Vault attempts to look up the Azure built-in role by "roleName". However, role ID takes precedence. One of 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 defaults to 900.

Note: Azure built-in role names and IDs can be found [here](#)

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/b24988ac6180-42a0-ab88-20f7382dd24c",
  "scope": "/subscriptions/<Azure Subscription ID>/resourceGroups/<resource group name>",
  "ttl": 36000
}
```

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"
```

Now anytime you read the dynamic Secret, the data is populated with the temporary azure access credentials.

```
{
  "id": "27a405c6-14b4-4d4b-b566-9fe23f1012c2",
  "path": "dynamic:azure:ac-api",
  "attributes": {
    "linkConfig": {
      "linkType": "dynamic",
      "linkedSecret": "azure:base:api-account"
    },
    "roleId": "/subscriptions/6ca2adeb-7b44-4c7f-93fc2d5b9729a8c1/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",
    "client_id": "945d25cb-7697-4648-b574-e8a660154269",
    "client_secret": "ce1d072d-449d-4052-9a81-0d7ef982f7a4",
    "role": "Contributor",
    "roleAssignmentId": "/subscriptions/6ca2adeb-7b44-4c7f-93fc2d5b9729a8c1/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-20f7382dd24c",
    "roleAssignmentStatus": "created",
    "roleAssignmentTaskId": "task_3da0a37c-0a1c-4ebd-8829-dbe7b988b36f",
    "spObjectId": "1782611c-99c2-418b-b672-783e3cf8bd14",
    "subscription_id": "6ca2adeb-7b44-4c7f-93fc-2d5b9729a8c1",
  }
}
```

```

    "tenant_id": "11f54b31-ffb9-42b5-8fda-76c734a7796c",
    "ttl": 36000
  },
  "created": "2020-02-12T20:57:44Z",
  "lastModified": "2020-03-04T19:27:45Z",
  "version": "12"
}

```

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 Privilege DevOps Vault API:

Use the `roleAssignmentTaskId` from above response

method	path
GET	/v1/task/status/

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.

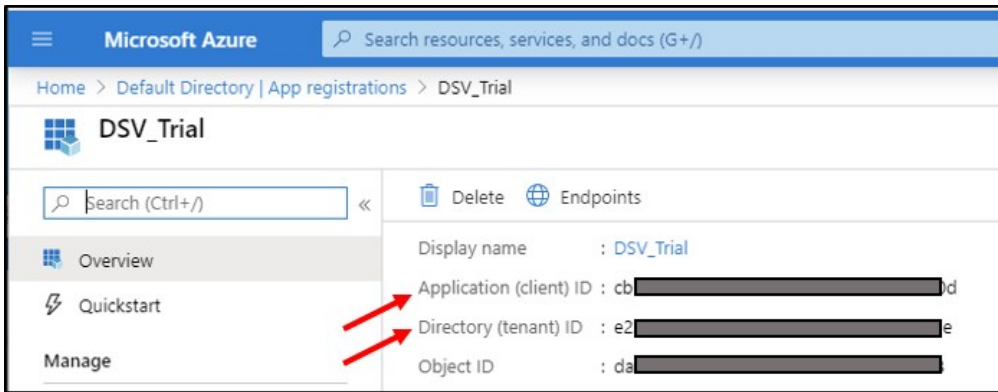
An Azure service principal is an identity created for use with applications, hosted services, and automated tools to access Azure resources.

These are the links to azure documentation on service principal:

- [Service Principal](#)
- [Create Service Principal](#)

## Creating a Service Principal for the Privilege DevOps Vault 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 Privilege DevOps Vault Base secret `clientId` and `tenantId` parameters respectively.



5. Select **Certifications & secrets** then **New client secret**. Enter a description and when it expires. Click **Add**.
6. Take note of the newly generated secret which is the `clientSecret` parameter in the Privilege DevOps Vault 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.

What type of permissions does your application require?

**Delegated permissions**  
Your application needs to access the API as the signed-in user.

**Application permissions**  
Your application runs as a background service or daemon without a signed-in user.

Select permissions [expand all](#)

Permission	Admin consent required
<ul style="list-style-type: none"> <li>&gt; Directory</li> <li>&gt; Group</li> <li>&gt; Member</li> <li>&gt; Policy</li> </ul>	
<p>▼ <b>User (1)</b></p>	
<input checked="" type="checkbox"/> <b>User.Read</b> Sign in and read user profile ⓘ	-
<input type="checkbox"/> <b>User.Read.All</b> Read all users' full profiles ⓘ	Yes
<input type="checkbox"/> <b>User.ReadBasic.All</b> Read all users' basic profiles ⓘ	-

10. Select **Application permissions** and expand the **Application** and **Directory** accordions. Check the **Application.ReadWrite.All** and **Directory.ReadWrite.All** boxes.



What type of permissions does your application require?

**Delegated permissions**  
Your application needs to access the API as the signed-in user.

**Application permissions**  
Your application runs as a background service or daemon without a signed-in user.

Select permissions [expand all](#)

Permission	Admin consent required
<b>Application (1)</b>	
<input checked="" type="checkbox"/> Application.ReadWrite.All Read and write all applications ⓘ	Yes
<input type="checkbox"/> Application.ReadWrite.OwnedBy Manage apps that this app creates or owns ⓘ	Yes
<b>&gt; Device</b>	
<b>Directory (1)</b>	
<input type="checkbox"/> Directory.Read.All Read directory data ⓘ	Yes
<input checked="" type="checkbox"/> Directory.ReadWrite.All Read and write directory data ⓘ	Yes
<b>&gt; Domain</b>	
<b>&gt; Member</b>	
<b>&gt; Policy</b>	

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.

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

[+ Add a permission](#) [Grant admin consent for Default Directory](#)

API / Permissions name	Type	Description	Admin consent req...	Status
▼ Azure Active Directory Graph (3) ...				
Application.ReadWrite.All	Application	Read and write all applications	Yes	⚠ Not granted for Default ...
Directory.ReadWrite.All	Application	Read and write directory data	Yes	⚠ Not granted for Default ...
User.Read	Delegated	Sign in and read user profile	-	...
▼ Microsoft Graph (1) ...				
User.Read	Delegated	Sign in and read user profile	-	...

13. Navigate to **Home > Subscriptions** and take note of the **Subscription ID** that you are using. This is the `subscriptionId` in the Privilege DevOps Vault Base Secret.

Microsoft Azure Search resources, services, and docs (G+)

Home > Subscriptions

## Subscriptions

Default Directory

+ Add

Showing subscriptions in Default Directory. Don't see a subscription? [Switch directories](#)

My role ⓘ 8 selected Status ⓘ 3 selected

[Apply](#)

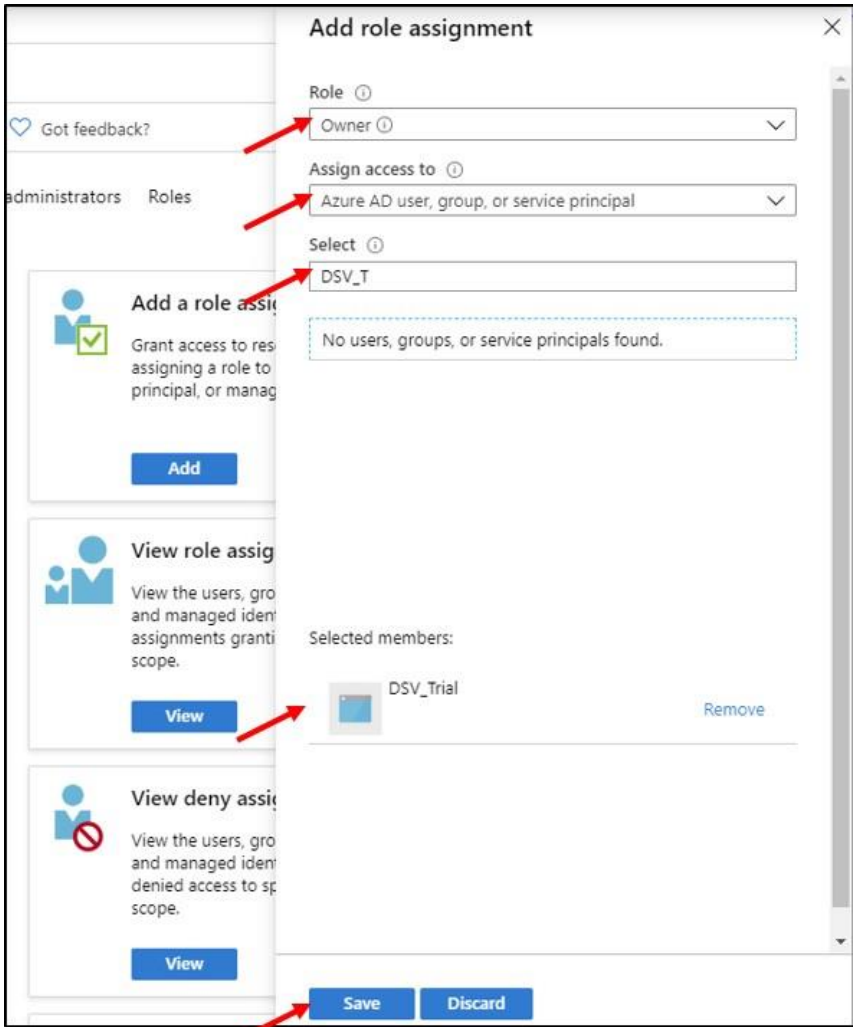
Showing 1 of 1 subscriptions:  Show only subscriptions selected in the [global subscriptions filter](#) ⓘ

Search to filter items...

Subscription name	Subscription ID	My role
Azure subscription 1	3a[redacted]98	Owner

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** dropdown.
16. Select **Azure AD user, group, or service principal** in the **Assign access to** dropdown.
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.

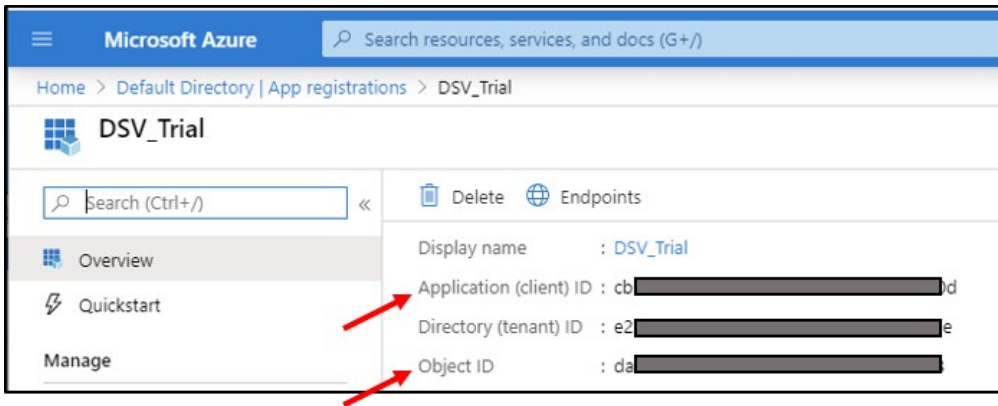
## 18. Click **Save**



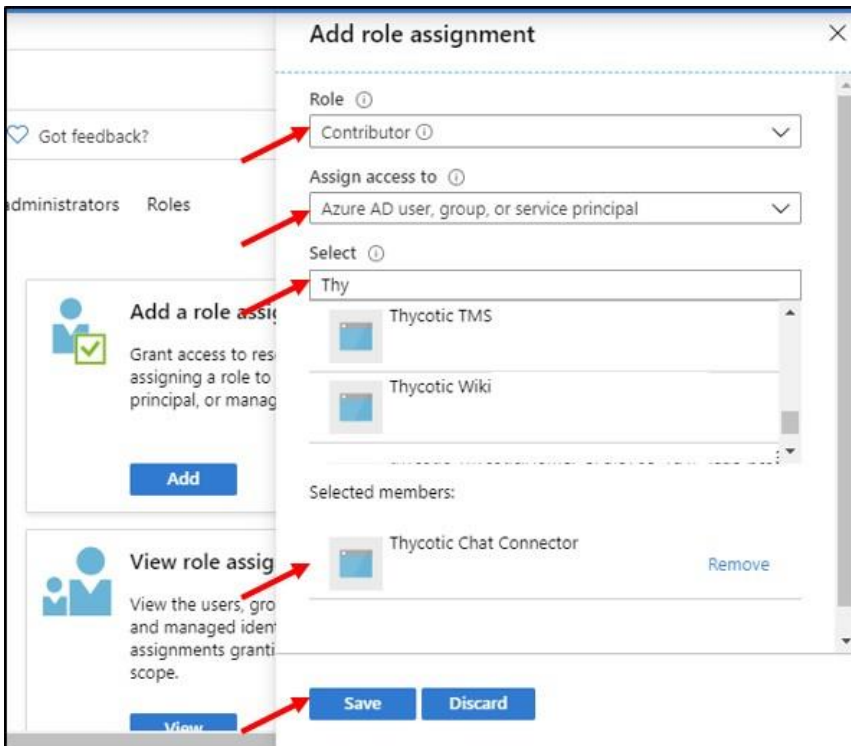
## Creating a Service Principal for a Privilege DevOps Vault Dynamic Secret

In the **Azure Dynamic Secrets** section, we discuss Privilege DevOps Vault using an "existing service principal" vs Privilege DevOps Vault 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 Privilege DevOps Vault 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 Privilege DevOps Vault 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** dropdown, select the role you wish to provide. In this example, we use **Contributor**.
8. Select **Azure AD user, group, or service principal** in the **Assign access to** dropdown.
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**



## 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 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 is used for testing the dynamic secrets

### Create the Base Secret

Next create a Secret in Privilege DevOps Vault 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": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "privateKey": "-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----\n",
  "clientEmail": "dsv-test@test-project-1234.iam.gserviceaccount.com",
  "clientId": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "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 <a href="#">scopes</a> 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 a temporary access token that is valid for 1 hour.

```
dsv secret read --path dynamic/gcp/token
```

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": "ya29.c.Ko8ByAfMsL-...JbFloC6tOiUCOM6vXn7YNhZA",
    "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, substituting 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 generate a new key in json format for the service account. This creates a new key in GCP that can be used to authenticate with the gcloud CLI or other SDK's. Once the ttl for the dynamic secret expires the key is removed.

Service accounts in GCP are limited to 10 keys per account. If you exceed this you 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 ttl's 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 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
```

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-
prog1234.iam.gserviceaccount.com/keys/0e4c690b713bfe0ed517ed56cba4
814afd35a8ad",      "privateKeyData":
    {
      "client_id": "xxxxxxxxxxxxxxxxxxxx",
      "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": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
      "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"
}

```

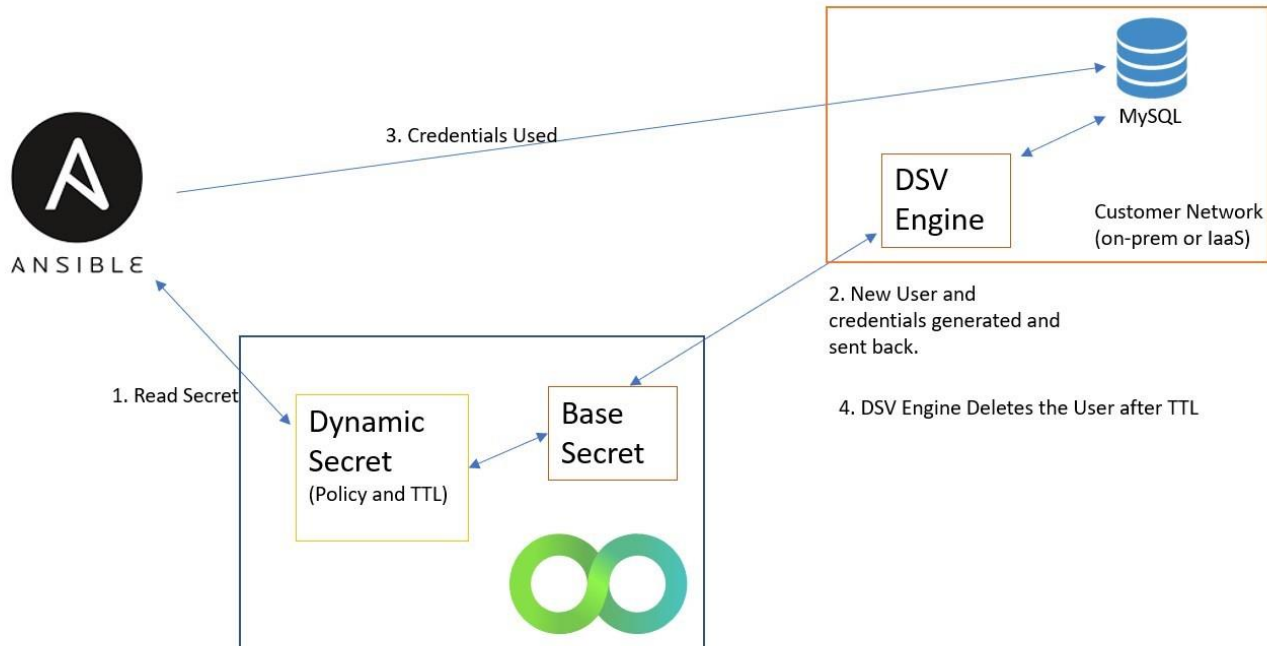
Copy the inner JSON of `privateKeyData` into a file and name it `svc-account.json`. Then using the `gcloud` CLI run `gcloud auth activate-serviceaccount --key-file svc-account.json` to test the generated key is valid. If so, you get a reply similar to 'Activated service account credentials for: [service account email]'

After the ttl expires you can check the keys on the service account and they are removed. Note that there may be some delay between when the ttl expires and when the key is removed from the service account.



## MySQL Dynamic Secrets

Database Dynamic Secrets are similar to IaaS Dynamic Secrets in that the idea is to provide temporary credentials for very specific uses. The possible damage done by leaked credentials is severely limited due to granular policies and short time-to-live. However, IaaS platforms provide mechanisms for ephemeral credentials with fine-grained policies, and most databases do not. Therefore, Privilege DevOps Vault provides a way to provide ephemeral credentials by creating and deleting users in a just-in-time manner.



### Privilege DevOps Vault Engine Required

MySQL Dynamic Secrets requires the deployment of the Privilege DevOps Vault Engine. See the instructions at [Privilege DevOps Vault Engine](#)

### Dynamic Secret Setup

In the CLI, create a base secret containing the credentials of the MySQL account that is responsible for creating new MySQL accounts on a given MySQL server.

The secret could look like the following:

```
{
  "path": "db:mysql:root",
  "attributes": {
    "type": "mysql"
  },
  "description": "mysql root credentials",
  "data": {
    "host": "database-1.cjqpjhsaz53.us-east-1.rds.amazonaws.com",
    "password": "P@ssword!",
    "port": 3306,
    "username": "admin"
  }
}
```

```
}
```

The path is arbitrary, as is the description, of all secrets. To mark a secret as a MySQL root secret, ensure its attributes contain a key `type` with a value of `mysql`. All fields in the data object are required.

Then create a new dynamic secret linked to the root secret. The secret could look like the following:

```
{
  "path": "db:mysql:dyn1",
  "attributes": {
    "grantPermissions": {
      "what": "SELECT",
      "where": "*.*"
    },
    "linkConfig": {
      "linkType": "dynamic",
      "linkedSecret": "db:mysql:root"
    },
    "pool":
"pool1",
    "ttl": 1000
  },
  "data": {},
}
```

The path is arbitrary. There is no secret data when creating the dynamic secret. All the necessary information is in the attributes, where all the fields are required. In the `linkConfig`, be sure to specify the path of the root secret as the value of the `linkedSecret` key. The value of `linkType` is always `dynamic` for dynamic secrets.

The `grantPermissions` object specifies the permissions assigned by MySQL to the new user account. The `ttl` specifies the number of seconds for which the new account exists before the engine automatically deletes it.

The attributes may also include an optional `userPrefix` key whose value is a string prepended to all MySQL account usernames created from the dynamic secret.

## **Sending a MySQL task to an engine**

Read the MySQL dynamic secret. A randomly chosen engine in a pool of engines receives the task and performs 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, some engine receives a MySQL account revocation task and delete the account once its TTL expires.

## **Privilege DevOps Vault 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 Privilege DevOps Vault 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 Privilege DevOps Vault Engine is to support database dynamic secrets. In this use-case, a user or application requests access to a database. Privilege DevOps Vault has a "base" secret that gives Privilege DevOps Vault access to the database and permission to create users along with permissions and credentials. Privilege DevOps Vault provides those new credentials to the user or application for use. Then when the TTL expires, Privilege DevOps Vault goes 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 Privilege DevOps Vault Engine include additional authentication methods and password rotation.

## Customer Firewall

The Privilege DevOps Vault Engine uses secure websockets (wss) on port 443 TCP outbound. Since most users already have this port open for web access, they do not need to make firewall changes.

## Registering a pool and an engine

Users can create engines as other entities (like roles, users) in Privilege DevOps Vault. Privilege DevOps Vault organizes engines in pools, so an engine must be assigned to an existing pool. Using the [DSV API](#), users first create a pool, then an engine assigned to that pool. An engine can only be assigned to one pool. A pool can contain many engines.

## Starting an engine

To start an engine in a container, pull the appropriate image and run a container from it. The result depends 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:

```
docker run -e ENGINE_NAME=engine1 -e DSV_POOL=pool1 -e DSV_TENANT=bob -e DSV_URL=secretsvaultcloud.com -e DSV_TOKEN=eyJhbGcxNjAKadw dsv-engine
```

You 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 ENGINE_NAME=engine1 -e DSV_ENDPOINT=bob.ws.secretsvaultcloud.com -e DSV_PRIVATE_KEY=LS0tLS1CRU1BSkFURS dsv-engine
```

In either case, on successful engine start, you get a message saying that the engine is ready and waiting for messages.

## List of environment variables for engine Docker container

- ENGINE\_NAME
- DSV\_POOL
- DSV\_TENANT

- DSV\_URL
- DSV\_TOKEN
- DSV\_PRIVATE\_KEY
- DSV\_ENDPOINT

## Running the Privilege DevOps Vault -engine binary directly

The container encapsulates the operations of the `dsv-engine` binary, which is a client-side CLI program to register and run an engine. It exposes two commands: `register` and `run`. Standard help is available with `dsv-engine register -h` and `dsv-engine run -h`.

## Certificate Issuance

IBM Security Verify Privilege DevOps Vault provides the ability to generate and sign leaf (end-entity) certificates or to create and sign a certificate from a certificate signing request (CSR).

All certificates assume **RSA 2048** key-pairs and **SHA-256 Hashing**

A signing certificate is required, and it may be generated in Privilege DevOps Vault or imported from an outside Certificate Authority (CA). This documentation often refers to the signing certificate as the "root" certificate. However, in the case of a signing certificate being imported from an outside CA, best practices are to use an intermediate certificate as the Privilege DevOps Vault 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
<code>commonname</code>	Required - The domain name of the root CA
<code>rootcapath</code>	Required - Path and name of a secret that contains the signing certificate
<code>domains</code>	Required - List of domains that this signing certificate is allowed to sign leaf certificates
<code>maxttl</code>	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
<code>crl</code>	Optional - URL where customer-supported certificate revocation list (CRL) resides
<code>country</code>	Optional
<code>state</code>	Optional
<code>locality</code>	Optional
<code>email</code>	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 retrieve the root certificate and private key, run `dsv secret read --path ca/myroot`

```
{
  "attributes": {
    "type": "CA"
  },
  "created": "2020-04-09T20:29:41Z",
  "createdBy": "users:thy-one:dsvtest9519@mailinator.com",
  "data": {
"cert":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSURnakNDQW1xZ0F3SUJBZ01FTVp4NWJqQU5CZ2txaGtpRzl3M
EJBUXNGQURCaE1Rc3dDUVlEVlFRR0V3SlYKVXpFTE1Ba0dBMVVFQ0JNQ1NVRXhEakFNQmdOVkIBY1RCVUp2YjI1bE1
STXdFUVlEVlFRS0V3cEdiMjldWVhJcwpTVzVqTVFrd0J3WURWUWFMXkdBeEZUQVRCZ05WQkFNVERIU91V052ZEdsa
kxtTnZiVEFlRncweU1EQTBNRGt5Ck1ESTVOREZHRncweU1EQTFNakV4TWpJNU5ERmFNR0V4Q3pBSkNt1ZCQV1UQWx
WVE1Rc3dDUVlEVlFRS
UV3SkoKUVRFt01Bd0dBMVVFQnhNR1FtOXZibV4RXpBUkNt1ZCQW9UQ2tadmIwSmhjaXhKYm1NeENUQUhCZ05WQkF
zVApBREVTUJNR0ExVUVBeE1NZE0dNVkyOTB
hV011WTI5dE1JSUJJakFOQmdrcWhraUc5dzBCQVFFRkFBT0NBUThBCk1JSUJDZ0tDQVFFQXRUVjFKaDZ4UkdRYVZ0O
WhvaUdvWjdiN3JTVzk3YVZhRnprK2VESUNhZ
ThFsJFpYkdsQlAKVFJMUZHLZlnMUtNTFhPUjArcDRWSH1vYjhzVWhSb0tYeHZZa2t4eXM4RjBoVvdeblUxZHJFVXh
rZGk0R3BhdQpObEJJJaWhmblpRdmtnY0txMzF
oYktpSlIwaTU0b0NnNjhyNVY2VUY4bVpNQWloa2cya012emFJMFE0TGE2d3FaCj1SR1FSU1JLRkIzNEx6SudnaFpDS
ldTUkY2UDZnSWJpM2VOck1KRWdsaUdqblFYW
jJlanJ1RURWaHhQ295WjYKdmdUdDIza2dxWnNOQUxxUE9CazJGeGZZQ3FuS2d3TTdRYTNRdmdNeVE0eG5KSTBqTUJ
aVWpFU0IvSmRiRVo5eQplckhsZGpSYnFSUjhh
rR0RsYksweDBkUW1jNHpUQitOc0JRSURBUUFcbzBjd1FEQU9CZ05WSFE4QkFmOEVCQU1DCkFvUXdIUU1EVlIwbEJCW
XdGQV1JS3dZQkJRvUhd01HQ0NzR0FRVUZCd
01CTUE4R0ExvWRFd0VCL3dRRk1BTUIKQWY4d0RRWUplb1pJaHJzTkFRRUxCUUFEZ2dFQkFbcEzNYWhFM1FINHQ3U0g
zczNNK1ZUSGJpSWhrUnVxazVVZQozK1M2Ykp
iL3ROckRVTE5lSFkyaDBPRGpmcWI3QWk5RElSMjc3dW8vVkh0QW1zWno1bEJ5TjJLZSs3YUxXY2FTClVoek1FVUt6c
m4vMW90T2Q5S2RuVWJ1cS8xNEVCVmUyb0t4Y
1k1cHdJZTznMkpVMW5oSGM2SENEJmJVNVRnVmgKbZnWclJ0NVA5VUs4aWsrUlDbktObVRJUWhsRDVhZ2VJeVp0UmY
yQ01xdzR0TldMRzU4b011UTQrcjVwY2Vqegp
FSGI1UHpiR29wMGI3NUdyQVFZbWhFU2d4SnVUZWI3WnZiTUixbG5QdnFyWWNCN09MR2VyaDY4bHZ4K1NadVh2CmE2N
ld0RmNobjFlR3c0WlQxdz14Vh5VOVhqrndvbjRqaG9VdlRrR0k0L2c0NlJVY1NoZz0KLS0tLS1FTkQgQ0VSVElGSUN
BVEUtLS0tLQo=",
    "domains": ["foo.org", "bar.org"],
    "maxTTL": 1000,
    "privateKey":
"LS0tLS1CRUdJTiB1SU0EgUFJJVkfURSB1RVktLS0tLQpNSU1Fb3dJQkFBS0NB0UUVBdFVSMUpoNnhSR1FhVnQ5aG9pR
29aN2I3clNXOTdhUWFGemsrZURJQ2FlOEVKMw1iCkdsQlBUUkxRkcvOWcxS01MWE9SMCtwNFZiEw9iOHNVWFJvSlh
4dllra3h5czhGMGhV0RuVTFkckVVeGtkaTQKR3BhdU5sQklpaGZuWlF2a2djs3EzMWWhiS2l
KUjBpNTRvQ2c2OHI1VjZVRjhtWk1BaWhrZzJrTXZ6YUkwUTRMYQo2d3FaOVJGUUVJSUktGQjM0THpJR2doWkNKV1NSR
```

```

jZQNmdJYmkzZU5yTUfFZ2xpR2pvUVhaMmVqc
nVFRFZoeGpDCm95WjZ2Z1R0MjNrZ3Fac05BTHFQT0JrMkZ4Z11DcW5LZ3dNN1FhM1F2Z015UTR4bKpJMGpNQ1pVakV
TQi9KZGIKRVo5eWVySGxkalJicVJSOGtHRGx
iSzB4MGRRbWM0e1RCK05zQ1FJREFRQUJBb01CQUJYYk1UenRhblpTazVKeAo4TFc1MVRKY0w5QmF3cUhLclpLclJrc
jd6S3Ext1EwQmRBSDDvM1FwZz1qby8rbzdvo
GMvTGhBZEwxRVFqc2FiCjkrS1olekk4aTBwb21WUC9PV3R3VEVSRk5jdzFzNXBnU1NKL2xKWGI3RU1xU3E0M1Z1RUD
kYy9rT1duRkpaUncKSWY4OW1vMzJRU21VeWM5Q21FZ09hNVdsa0RmODZLYjJMS2ZscXE1QWkybCs2VVRQTGovejlpT
GhDcTdqTFRtVwpaSzVhcVdaUnpNQ24rVEhnnEdUY2dBeW10VzJnbUo2RFBSW1dzaHJSUUJ2V
VloY1JjSnBKN3FQb3hEOGpMNXIyCmVXV0UzZGs1bzJSdG5aZFRZU095N0o4ZFM5c2F0Sk1UQmdxN0ZNbkFRR2E0S2p
iYStkQ3RuVjRPaGhiV240dGIKR2NtUjJvRUN
nWUVBeFd1QnpvR3p2RnJqSG13ZmYraV1YUnFvcEJRr0VBd0gvUC9SUzFQMGnNbjNuYkFKdzZOegpEbW1SSH1DNHhFQ
XhOzZvQz25mdkMvYS9UcnZxMy9JY3doZzdMM
UtIajh6d2NrOGFvWddoZFNjWVFCZ2w0bU1CCnDaVpicmdwblVBbHUxZFRlZ3BULzVYZzBERX1HUE15VkpIZmF3cGp
rV0p1QTdSejRtYjczdkVDZ11FQTZ4SzyKWHZUVWFzcfk10WV5emhFU1RhNEdzVTFMRzMrTJCb0owS1h6dEQ4TkvTm
kZTM1RjQ2Jsbk9Rakxod2RpU3E5MgptNnZXEjVpVG1teGwvMHI2cEhZL3Q1RmJOOEV5eHVzZ
GdCbDBNTkr1THM2bTRubU5uWXPvST1qOUF6ajg1alVPCmdaTTJ1S01zMDNqMGZudG1vejQzYXRnV0M5R3EvOE12eG0
wVXhsVUNnWUFGeWcxU2d4ZEVWTjRJU243NS8
xWkkKbExtU1puSjVFZ0ZCK0RhcE1PTXdyUDU0RDJ1WjR6ZENTdkg0bWJzUmRsaDdIMXpudktDMEZ4NXhMcTBVa0Vnc
gpwZzVHU3dOU3drM2k3Rkw1b11CbENTcDY1c
nBsczBXZ1p2Rm8vOW1vbFBNR1ZYOUk0bG1vVERyMW9CdTZBNWZjC1J4TG9UcnV3emRRd0k2Q3FhUjdGNFFLQmdGNm1
oOHC4SUZ0dlppVFR3UGNnQUpldWc1dF1WK21
WaVNIS09qRjgKNE1ldTY0Q0VBS3UreEp6RnZqNUV3RTU2TnFXRHlPb2RZcnpYm21MTFNyWmtaaz1hSf1XNFRWWk3R
VEvM3Z6NQpRc04xSExKVud2WU5vMnZRaklweWtFMS80TFNBb0hXajM4YnE1Y213WmlHWFpsaE1jTnZnYmVBTWFDSGE
rb21XCjJrcVJBb0dCQUkyQW8zdk1Uei84ckc3Sfd6b1VML0w1OWZwaUMrVXJvUXUwcUxxR1BDtkQ2d2kyUy91NkFFS
1IKMWhQRWJ1b1NvUG4vaExhadNHL3VsWk9tMmU3d1Z6dHpoblRiBUk0WGZrbENaUWV4Q3BQOE9wUD1KUDZHZZVVOQp
MbHpaSkFjZHVfck5zb2pXcTluYVhCZkdZUFkyd0kvOXZyQ29HUGhDMXVVMURnVf1QNk9ZCi0tLS0tRU5E1FJTQSBQU
klWQVRfIEtFWS0tLS0tCg=="

```

```

},
"description": "",
"id": "90delc6b-3c85-42cf-9d6a-758b48f1daf5",
"lastModified": "2020-04-09T20:29:41Z",
"lastModifiedBy": "users:thy-one:dsctest9519@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 IBM Security Verify Privilege DevOps 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 contains 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 is an error. For example, if this signing certificate has an expiration date next week, the maxTTL maximum 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-----
MIIDnjCCAoagAwIBAgIJAM0hi74h41RqMA0GCSqGSIb3DQEBCwUAMGQxCzAJBgNVBAYTAlVTMQswCQYDVQQIDAJJTD
EQMA4GA1UEBwwHQ2hpY2FnbzEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMRMwEQYDVQQDDApmb29i
YXIub3JnMB4XDTIwMDQxMDA0MjMyOFoXDTI1MDQwOTAxMjMyOFowZDELMAkGA1UEBhMCMVVMxMzAJ
BgNVBAGMAklMMRAwDgYDVQQHDAdDaGljYWdvMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQxEzARBgNVBAMMCMZvb2JhcnVvcwggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQcXdninSZ/wDyXCRCRCAGhdGxP8/YW4sX1OcStj1
qOjVVCGER0wrLG0rDFb/KxVJ3WVM4lh381ZUT/N6qcRr12ZPupPh9P9jjU5NkJIS
x2wIsuptRFz4nsBoIdDdMun0CDbscEuWUIjEdsC5kj7DPLa16u6ic0xxAH9RW
YzQoV92hsjmIZvHtZpCoVMsUMF70Nbzh54wZgajzMPV0jaGKrqlMnLs5010+AY4k03NlfsTSNsOA8a+jjXXG331j
muQpH4UphcmUfMjpeFw6x/qwSrxKz07k6dWKKcmJzqAj/MXA7coOvwj7L39uv/cMVzk/MTeLYW2Jbz7h07CBAGMB
AAGjUzBRMB0GA1UdDgQWBTRG8SIEqc672Onj/fPAQss3eA1pjAfBgNVHSMGDAWgBTRG8SIEqc672Onj/fPAQss3e
A1pjAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQCuomjUQVYMGcPz1wzc2GJw57dTONnNyLXUdiOp
GORxheplveFkCQmgrxAMu7Ky ZNEoINmkHY1f00p7hAzKIWpFBSpMwDZg/1vamjE0riJ+JxGwo2C34WZqRJHbunK5
cBmZBeER93L76Pc8k6eC/01cus+hiqs2Mg7Ugg0RsV+fEs6BEL0KQqh+VG+rPq6C
WH9GJr9PiLD+gG6rxOZRRxt6gx1XOoK6REj1W5wMaxeS2+SKOHGPhaRE+z1xXC9z
7Y8j7UnAeE9dikJipfgj48zWskUexW6rxYK7hiz5nX3VCP1XpZp5uFhXmegJ1fmDQx0dZF6QQRiK4MNGZ2mg1y3F--
---END CERTIFICATE-----
```

Create a file with this corresponding private key and name it `key.pem`

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAsQ54p0mf8A81wnEQgIB3RsT/P2FuLF9TnErY5aJo1VQhhK9MKyxtKxwW/ysVSd11TOJYd/NWVE
/zeqEa5dmT7qT4fT/Y41OTZCSEsdsCLLqBURC
7sOJ0gaCHQ3TLp9Ag27HBL1lCIxHbAuZI+wzy2jderuonDscQB/UVmM0KfFdobI5
iGbx7c6QqFTLFDBezjW84eeMGYGo8zD1di2hiq6izJ7oS7OttTvgGOJNNzZX7E0j
bDgPGvo411xt99Y5rkD4eFKYXJLHzI6RH1sOsf6sEq8Ss905OnQ1iinJic6gI/zF
w03KDr8I+y9/br/3DFc5PzE3i2FtiW8+4dOwgQIDAQABAoIBAEBGUXVcadlR/X2
pN+OQDu9+UkeaibOfgdGJUvMbpwlyXhnSoSMvh4Wf2hiUXqaUE6EA0mdVeKJlBsZ
7ACEVQxwkYU7LokJ2rZJ1snb+Hh7vprjabr52oYP+J7kypUsFPTeenpbcrcUGMNU
vkKMUGvrXh3qB3qT9V/MbXrgzCgriHazR27/pPLJALnOAusu7C0XGSa7eJSY6ysO
neKwKwtJiPwa3wTp9LHxeHrkYbEd4cx2G3no1SM4IUdOUjAkHJ2OyShkyn/vXUn9
Aygnlp0s26MIgXgk46AqoR0WIwRYu68FqdXdC16GRmcByALKA5XJ4Hqz9Q8ufoJf/R9PwjECgYEA5cvcHTX+OCbzgU
rtODz3ymHK2q2fSoMGPPiBHQiQIhaVtprCpMp6hIy4Vv/D2rHbWj+idMufnvAPjr+qJPRzId0VmRkDyLHGq2WjBv4
0wc5u4Pw+sa9YPHqNDmCu4wABvc4lbKueP7OtAcp04nLSk3B9ZLBN0jQNMmDvim2db0CgYEAxT8M
XawhG9LpL7tFtIQsvIXtVYlFimC5+CmnFLjckD/1jqz8rVJSLCetPZnh2tDcifxh
yo8UA+/nWHy0tF6JIIhfH+DqUWwWCPxJc5djwM8Zs3TrnawIBYwcl3wUM7X6FLSX
v5unb61XjPYWMU6z64cVaCH20sCUXing9Sh4qBUCgYAOXZUwGkz/M6grYAS+bElN
VJm62/nGTbSW4MAzarm1l/iVz2e7rIGFSYf2wH6JtzIqa9LlyNbyP0hAW63J2hvW
fm1ObU44CAOMbmen8K04hY4dY90vwDbclgllimbalKC3zsKx0Q7JL5y6cmwx9j5I
Md47POZvqbpCYoqcW1U1vQKBGQC6oxnUWNdLOJqlK5KdaKPCFPv30DgY48WUZ/VM
yk6nVz3HLzA34DkYwJvKOh1Xq2HCvyjZPeE2iH5jYDysnvcP7WBXdh7BxIBLKDNo
Smt+2Xf8Mpnvq6Q7dV3iiomktIBZrzgXefVI2sCJBSGirLHYfwlmZxzh9o9tOjs+
PnlMsQKBgAUCVf5yqUGETwkv17I/2Fn+17Hw3Yv8Ced1WKB6bwoF5Hd1lr01LgpF
q10bc+NezxCPQd+dBNBgfbcWpWvYPdfte2u6G94G80qiOXczwu7Z3iI6puukV4Uy
8Nz6NxjrgibNpB/nui0i36HKAYDwmo57mc7UofPCEieIK/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`

Flag	Description
<code>commonname</code>	Required - The domain name that this certificate uses. This must match a domain in the signing certificate's list.
<code>rootcapath</code>	Required - Path and name of a secret that contains the signing certificate. It does not matter if the signing certificate was generated by Privilege DevOps Vault or imported.
<code>ttl</code>	Optional - Time to live in hours. If not specified, then the <code>maxttl</code> of the signing certificate is used.
<code>store-path</code>	Optional - Path and name of a secret that contains this leaf certificate and private key. If not specified, then Privilege DevOps Vault does not store the leaf certificate and private key and there is no way to retrieve them after the initial stdout is deleted.
<code>country</code>	Optional.
<code>state</code>	Optional.
<code>locality</code>	Optional.
<code>email</code>	Optional.
<code>organization</code>	Optional.

For this example, we 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":
  "LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSURaaKNDQWs2Z0F3SUJBZ01FR11XNFRUQU5CZ2txaGtpRzl3M
  EJBUXNGQURCaE1Rc3dDUVlEVlFRR0V3S1YKVXpFTE1Ba0dBMVVlFQ0JNQ1NVRXhEakFNQmdOVkIBY1RCVUp2YjI1bE1
  STXdFUVlEVlFRR0V3cEdiMj1DWVhJcwpTVzVqTVFrd0J3WURWUWVFMRXdBEEZUQVRCZ05WQkFNVERlUm91V052ZEdsa
  kxtTnZiVEFlRncweU1EQTBNVEF3Ck1qSTVNVGhhRncweU1EQTBNVEV3TWpJNU1UaGFNR0F4Q3pBSk1JnTlZCQVlUQWx
  WVE1Rc3dDUVlEVlFRS
  UV3SkQKUVRFV01CUUdBMVVlFQnhNT1UyRnVJRVP5WVclamFYTmPiekVQTUEwR0ExVUVDaE1HU05dlFtRn1NUWt3Qnd
  ZRlPwUWVFMRXdBEEVEQU9CZ05WQkFNVEIySmh
  jaTVqYjIwd2dnRW1NQTBHQ1NxR1NJYjNEUUVlFQVlFQ0SUJEd0F3CmdnRUtBb01CQVFDdUdNbm1ITjM4TjRGTGdBN
  H1ESEZTVWYrekxjREFGUW11SGZ1eTNDME5VL
  3RZeHNRtNnRczkKQUJkZGJyUTBMbjNVWkRNL2hVcUZIR2prSGRkUVROSTJMY2IzRGk4QWdLVU85OHVhOHVpWSttTDZ
  ZK211TE9XegozejVNNnRFOGdFbHn1QUJ4Vkf
  wT29hTG1EZV14MUxWOUdSU1VoZmlhZ1RFNFV4V3pmdTVKU0wyYVd2M3RreUhmCnpFandiaGFDVHV0d0gxM1NrczN5O
```



```

UNwZ091MW1qV1N3WmU0cjRGY284KzdMMEUvS
DZLcG9zQk1mWTV5N24wbm0KeU5NL2ZKM2d3eCtpSkJKa1o1RnJqRWxnNVIyZUs0aG5QdU1zeGFvY05FSElROGNXa1N
TOG0zWnpNRnVjYVdFMQpKN1NTSDQrd0ZXazB
ZdzA1cTRTznQreEhGK1VocFdmZkFnTUJBQUdqSnpBbE1BNEdBmVvKRHdFQi93UUVBd01ICmdEQVRCZ05WSFNVRUREQ
UtCZ2dyQmdFRkJRY0RBakFOQmdrcWhraUc5d
zBCQVFzRkFBT0NBuuVBzdtTjExRFAk3c5Y3vtWXJ1VzdzUEFSSWxUcHBwMStIY1BNa0JhL0JvZUwrOEdtM3JDZWg
yQnM4b09YQXhyVmVWSkZ5K0VNQQpIZjhQsJf
HazlMeHNzSDJQazk00TNGMzJ1VGhxUWo0d0RuQzG0TkpJZz1Ym1pNSkpDSFBjC0wvVU9kenZraEhLCnkVShk0bD15Y
0dQdGtudmtURkVkTVdKZ2hOcFgvSkxrTF1QZ
WthNzFORjFPOEFaMfZVbXJXMDR0YV1DYzZ5UvAKMv1JbXhSd1FLNVJiYWMxSWUxVEI5VWc5Z2dvUnhZOUpFKyt5aFR
oMU5SK0tYUTZucWVNbk1SdStxaERONjRxVwp
mMzhBU11LOmklqRndnTVBEK3E5R3JodW12REYxc051cDvDeFEZdi83S2dtNDNHTFFhZ3o2T0pib1NLbmYrM211Cit3M
TQxUXZJT1pDZDRnPT0KLS0tLS1FTkQgQ0VSV ElGSUNBVEUtLS0tLQo=", "privateKey":
"LS0tLS1CRUdJTiBSU0EgUjFJVkFURSBURVktLS0tLQpNSU1Fb3dJQkFBS0NBuuVBcmhqSjRoemQvRGVCUzRBT01ne
HhVbEgvc3kzQXdcVUcrUjMzc3R3dERWUdXtWJKCKRiRUxQUUFYWFc2ME5DNtKxR1F6UDRWS2hSeG81QjNYVUV6U05
pM0c5dzR2QU1DbER2ZkxtdkxvbVBwaSttUG8Kbml6bHM5OCtUT3JSUElCSmJIZ0FjV1FLVHFHaTRnM21NZFMxZ1JrV
VZJWDVtb0V4T1VNVnMzN3VTVWk5bWxyQOo3Wk1oeTh4StHhNFdnazdyY0I5ZDBwTE44dlFwWURydFpvMMwtzR1h1Syt
CWEtQUHV5OUJQeCtpcWFMQVRIMk9jCnU1OUo1c2pUUDN5ZDRNTWZvaVFTWkd1UmE0eEpZT1Vkbml1SVp6N2pMTVdxS
ERSQn1FUEhGcEVrdkp0MmN6QmIKbkdsae5TZWtraCtQc0JWcE5HTU5PYXVfbjdmC1J4ZmxJYVZuM3dJREFRQUJBb01
CQUdMVUdZNXRHcXE1aTRFagpnV3R4MnNhRFcrY01Hdm92T1pVbktOeDAXbkpSY1VaVkdMn1d1TzE0NXNwU5GM0c0c
EUyREUyTH11REVYdHJZCkFjBEl3ckFVem5TaXJaWFljVnFNmMh6c3RaTloxK1FSNFJRaG9vZTRPL0tIL2gwZEtoRVV
FaFJEUt1LZE9ReWcKSFVpK1h3U1R2MUczK0J
oNExFdZRRouP3Uks1K1YwRysyZjlqbjQ0M05BZGRTVWZ1UFRpVXVqelRTaWNGS1BKdwp0a1hYeU01VkpzVjN0VEZ2a
3ZkVE43WfVhUDNLQ3ZOdU9XWfUrbG1BS21qc
2xXSDBIRUJhS0NvWWVqMyt4ZURtCnFFR1A5bXc2eFZVY0hTalgzT1BHVfJrbnR3bXNkRkQ4Z2ZJYi9RZXpVRGVnV0V
vM21xstJpQ2RLbDUwWURLUWkKSUXzNHY1RUN
nWUVBeFdxOEdPMGRcrZBkbGjtTWpEUtE5NnQ0ckhGUjHObHNzOXZ6Wnd0VzZ4Z0c4d0NFwNfHTwPvNU1VeXd4YwXBL
0xQVmJtdVNHQm54Sy9FQTYrZVJ2cTlxOU5Uc
Ew5UDBDc3dpVldiMhpWdUNDQ1ZYRitar3diCkRkCvB0ZHdlb0dxNVZOaUhfUkVEemRuM0RWMVAxZzFyU09wR3BmT0w
5OVpYNU9IcGoraEhob2tDZ11lFQTRjsjgKRWh
zds9jc1ZSTjc0MGxsdszRQTU5HMFUxz01YaT1JVkZ5dkdtQUIxQ3FGUmpZeUtFTHZqQ1h6UFN2ZTRGczRvZQpRY1Uza
UVnU1djeEFFSmJ6VTB5Sit6ZHdITkpJOFJMM
zhxcTB6dVJUSG1pc2Y4cnhGZUt2QU8ONTE1N2R6WmJHC1R6MTMwUTRnc1RKbUxyR2xST0MrMHV5UkRqQm92RU12V2k
wV11TY0NnWUE0MwdYw1YwcW5YNUxJN0dhZVp
0bXkKdUZkQnJrNWMvUHZpdkV4VE9seUh5cDhWanV5UGNSeEF5eW5aVzNFb2QzT1g4VXN4cVlMiyitGV3hsYzBZcVFUN
AppSGZGUzJSRnRhVUhNq0MyWW5TV1VpWnFKd2F3ZXI4KzNiRetOdGxLYmU5MwMtmRxc1S2tudHJ6OX1BT11LTHplCmZ
UUmh5c0JkVmdSd0RPeGxXQVpmb1FLQmdCRmEwQXJjU0JwK2VCNfPqZQXQ5c0syN1FYR3RPbFd4NEthSGNEd1AKbzRFe
XZxTU9DYTNmUTJZUS9YQXdIYTA0R1B3ZVRBRW1WZ1NGOWRNdFhtZG9FMEIrQzhWaUY1NC9sQmZrSzJkZQpOQ1FMZ1Z
CREg2K2JQRGxBZWMrs2dLdlFyS0JYVE50ZwtFMWoxUm55RstUWEJ5dHFVNEVIYw9jNnRYSnpiQXGwCmx0blZBb0dCQ
UladjU2cGNrbXR0MkjqZzdDdnPja2VxbHhBeUxKWU1aaW5sYjhjTdjU5mV1NEQ5Wm0yNHdFOGkKV1N6OEwwUmF1K01
Idk85bXlrcKvubHhDcHd5aFuvL05tUD1ENmZGY1h1MWpCb1h4ZU1JRwt1Wk9LdzI5Rm1MMgpFSitKV2MrRkY0cGdpZ
HBUMctQL25oc2ZTVGt4TmtZaWdCSzJ1dmVBdTJIU0NtRWN1RjB1Ci0tLS0tRU5EIFJtQSBQUklwQVRFIETfWS0tLS0
tCg=="
}

```

## 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.

Flag	Description
<code>csrpath</code>	Required - Path to a PEM file containing the certificate signing request.

rootpath	Required - Path and name of a secret that contains the signing certificate. It does not matter if the signing certificate was generated by Privilege DevOps Vault or imported
subjectaltnames	Optional - List of alternative domains. They must match a domain in the signing certificate's list.
tll	Optional - Time to live in hours. If not specified, then the maxttl of the signing certificate is 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 sign it with the sample root certificate we generated at the top of this page.

```
-----BEGIN CERTIFICATE REQUEST-----
MIIC1DCCAXwCAQAwTzELMAkGA1UEBhMCVVMx CzA JBgNVBAGMAk1MMSEwHwYDVQQKBHJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQxEDA0BgNVBAMMB2Zvby5vcmcwggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCdcmthlMQcfWwZmKZr1G7aYuTLb
j/hCTILGEhGDcp0e1AEenzWGLdFusbIMdb7Z10/SEJLb9cVHGgcf9U67s9+1hqUPY
/xwCbHJ7JYfLHZm3XHT5oA2QumMNqwZlh/YTWUDUr9NYs1TZOUm4y6smzfO5TVOCZ9SFETi3ZfPsknQQ3EEmPso2yJ
U0yqxHkgozm2bYOitdlySEOM4R0JLQEBSgLLo4
QLtxJJZiKKVvuhGZ7SZUcXft4RxBq41uv1YyffWeZYa0b/h7hcb7Gj+pnaI/1PWm
vxdkW6cXnpAmL5k0PX1fQARGkBUfyF3DQGDfT41UfSHE9qWi0gA6wfhXvCFAGMB
AAGgADANBgkqhkiG9w0BAQsFAAOCAQEAmL2JDxGpKmIU60uMUsQXtylObyyIMW0q
bmmqrfccfxdV/WNLLOrm/8g0Rp/eWwAGkQY8tZJn1N+BPK6yFpx1TYW6z2aPGTUT
TgKnaheDwnpCPLkRJRqEIHye9B+vFvEJX111U7pA4FGIsvN+1R2TTG4nBp8Nx7NgLWCFT4m90R39wCxXEJMoUOIii8
mfeaFwLzstyb/pAPuQoWYebOMCTHxJsxRsr/w9
PBJS TPM+USH1xTUTtbEgY4SGFG7C+SYluFHj9c5hhH40TPv0NH9cmMHxSsbNKbou
wmq9DFjzRXDVjAMLb2fsbBBpQ7/aT30pJWjr9jAX0/FH1Ymg2aIK89w==
-----END CERTIFICATE REQUEST-----
```

```
dsv pki sign --rootcapath ca/myroot --csrpath @internalSite.csr --ttl 24
```

The signed certificate comes back in base64 encoding

```
{
  "certificate":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSURZakNDQWtxZ0F3SUJBZ01FRm1oYmV6QU5CZ2txaGtpRzl3M
EJBUXNGQURCaE1Rc3dDUV1EV1FRROV3S1YKVXpFTE1Ba0dBmVVFQ0JNQ1NVRXhEakFNQmdOVkIBY1RCVUp2YjI1bE1
STXdfUV1EV1FRS0V3cEdiMjldWVhJcwpTVzVqTVFRd0J3WURWUVFMRXdBeEZUQVRZ05WQkFNVERIUUM91V052ZEdsa
kxtTnZiVEF1RncweU1EQTBnVEF3Ck1qRTROVGRXhRncweU1EQTBnVEV3TWpFNE5UbGFNRTh4Q3pBSkNt1ZCQV1UQWx
WVE1Rc3dDUV1EV1FRSUV3SkoKVERFaE1COEdBMVVFQ2hNWVNXNTBaWEp1WlhRZ1YybGtaMmwwY31CUWRIa2dUSFJrT
VJBd0RnWURWUVFERXdKbQpiMjh1YjNKbk1JSUJJaF0QmdrcWhraUc5dzBCQVFFRkFBT0NBUThtBTU1JQkNnS0NBuUV
BM0pyWVpURUhIMXNHClppbWE5UuYybUxreTI0LzRRa31KUmhJUmczS2RIcFFCSjgxaGkzU1ZMR31ESFcrMlpUdjBoQ
1MyLlhGUhVSEgKL1ZPdtdQZnRZYWxEMlA4Y0FteHlleVdIeXgyWnQxeDARyUFOa0ZKakRhc0daWWYyRThGQTFLL1R
XTEpVMlRsSgp1TXVySnMzenVVMVRnbWZVaFJFNHqYWHO3SkowRU54QkpqN0tOc2lWTk1xc1I1SUtNNXRtMkRPTfhkY
2toRGpPCkVq1MwQkFVb0N5Nk9FQzdjU1NXWw1pbGI3b1JtZTBtVkhGMzd1RWNRYXVOYnI5V01uMzFubVdhdEcVNGU
0WEcKK3hvL3FaMmlQOVQxcHI4WFpGdW5GNTZRSmkRwK5EMTVYMEFFUnBBWkZCY2hkdzBCZzMWk05WSDBoeFBhbG90S
QpBT3NINFY3d2hRSURBUUF3C3pRd01qQU9CZ05WSFE4QkFmOEVCCU1DQjRBd0V3WURWUjBsQkF3d0NnWU1Ld1lCCk
RVUhbD013Q3dZRFZSMFJCQV3QW9JQU1BMEduDU3FHU01iM0RRRUJDD
1VBQTRJQkFRQkh1b2FwSk05VTUua0IKcU5Pb0hVNmJ3UmxiOUprRmc50Td3Y0UxU0dKbUNKTUd0ZkJMajZRRk80RnF
JZGU5Qk90N2o0bnZwQUduYXNmaQpzbzBWA09
tK1dyZUpuRXJiL0dMK0RpMEExKbGxSZHduYWJtY2NXTFVKNm5EWWxGYjZLdEdmU3dYQWJyTTh5VVZjCmdqdU1odU15d
1ExOHR1UEFTWGFwRwJwUwU2VyOFd4Q3dUMlgyvR
DhVaGhXR1Ercno5aFV0ZHpUdU5COUdVb21PaGUKb01XZGxHVV1pcm9sQS9GQk9nWjZCT2gxVnQ4S31FN0VLRjZJdU1
wM3kvc2szcGVMUmpUL0dIK0JxRW5PNmhZzWp
```

```
ia3NOcTNGSWROYmN1TExlV3dLWW1ZUEdQYWFuSnZ3NnZWN3MzRlQ0TUhUaUFtVTRkbTRkZVAvNzRpZXVvTXlXCnNpZ
TdESkoxCi0tLS0tRU5EIEENFU1RJRk1DQVRFLS0tLS0K"
}
```

## SSH Key Issuance

In addition to allowing users to generate TLS certificates, Privilege DevOps Vault 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 Privilege DevOps Vault 's specific root CA to sign the credentials which can then be used to access any SSH Server where Privilege DevOps Vault 's root CA is trusted

When users create a regular leaf or root certificate with `dsv pki leaf` or `dsv pki generate-root`, respectively, Privilege DevOps Vault automatically creates and saves an SSH-compatible public key. Privilege DevOps Vault 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 is a field for the SSH public key:

```
"sshPublicKey": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQ4nmHvYaqodYKU2..."
```

### 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 Privilege DevOps Vault 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 Privilege DevOps Vault 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 must 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 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 is valid

This returns an SSH-2 signed certificate. Privilege DevOps Vault 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

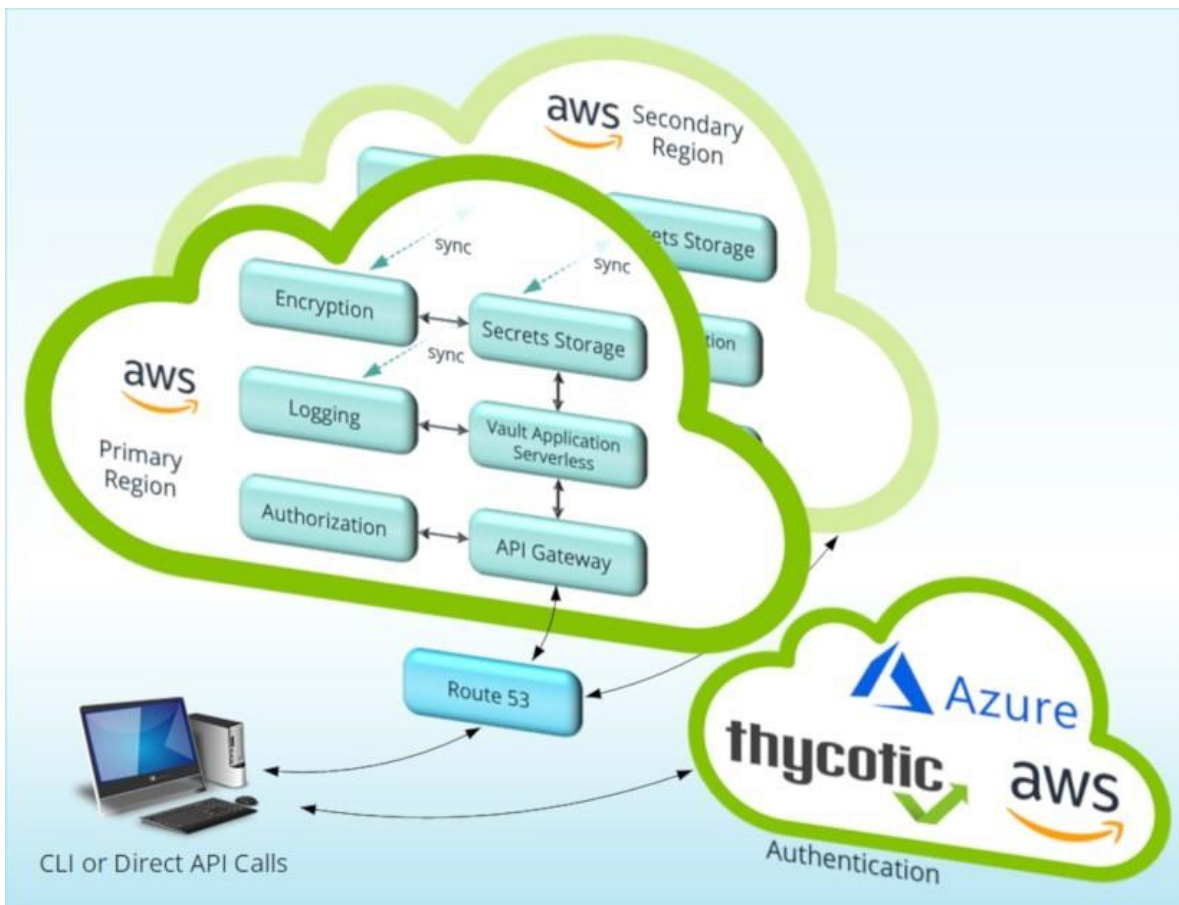
certificate in the same directory where the leaf private key is. For this reason, the certificate file must be named in a certain way: `[private key]-cert.pub`

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 just needs 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. Administrators must not do any additional setup on the server.

## Architecture and Security



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

Within the Privilege DevOps Vault 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 Privilege DevOps Vault functionality and auto-scales to demand.

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

## **Availability**

IBM Security Privilege DevOps Vault supports 99.9% uptime.

## **Business Continuity and Disaster Recovery**

IBM Security Verify Privilege DevOps 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.
- IBM Security 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

## **Confidentiality**

### **Data at Rest**

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

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

### **Data in Transit**

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

## **Client Authentication**

Privilege DevOps Vault 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, Privilege DevOps Vault 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, Privilege DevOps Vault requires re-authentication.

The initial administrator (the one who signs up for a tenant) is always setup with Thycotic One to enable IBM Security 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, Privilege DevOps Vault signs access tokens.

## Personally Identifiable Information (PII) and GDPR

Privilege DevOps Vault 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.

Privilege DevOps Vault 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 Privilege DevOps Vault Users can store anything they choose as a Secret—for example, images, documents, or other files.

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

Only selected, trusted employees of the original product manufacturer, Thycotic, 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, IBM Security customers are the data controllers, and IBM Security is the data processor.

- The customer determines all information (the Secrets) stored in the vault and decides how long to store it.
- Each Privilege DevOps Vault customer entirely controls their Users, their User Roles, and the access to Secrets by their Users, according to the policies of the customer organization. Privilege DevOps Vault 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, Privilege DevOps Vault logs include source IP addresses and time stamps.

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

### **Third Party SOC 2 Conformance Assessment**

Thycotic 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).



## Audits:

Privilege DevOps Vault captures audits of activities and persists them for future reference. If the User wants to ship their audits to a third party logging system (e.g. Security Information and Event Management (SIEM)), they can register an endpoint where Privilege DevOps Vault sends any recorded audit events to that endpoint in near real time.

Audit Fields:

attribute	description	example
id	Audit id	"dxv7389e463s72jbo345"
tenant	Tenant ID	"bjr738973p3s72jbo090"
tenantName	Tenant Name	"test"
principal	Security principal that performed action	"users:user"
principalItemId	Principal item ID	86b1c1aa-907e-41b8-8b02-e8d7dd467d6a"
action	Action performed	"POST"
Status	Response status code	"200"
path	Resource path action performed on	token"
ipaddress	IP Address logged from client	"192.0.2.55"
created	Audit created date	"2020-05-01T01:09:07.225694779Z"
message	Additional details	"login succeeded"

## Logging Format and Transport Protocols supported.

Privilege DevOps Vault supports the following logging output formats: [syslog](#), [CEF](#), and [JSON](#) to a registered endpoint.

Privilege DevOps Vault supports the following transport protocols: transport-level security (TLS) 1.2 over TCP, UDP, HTTP and HTTPS.

## SYSlog

Syslog messages must be in RFC 5424-compliant form. Privilege DevOps Vault truncates messages over 64KB in length.

Syslog	Audit
Timestamp	RFC3339 format
Priority	191
Version	1

Hostname	Privilege DevOps Vault URL (e.g. IBM Security.secretsvaultcloud.com)
MsgID	id
Appname	Privilege DevOps Vault
Message	usertoken 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 tenantName.dsvdomain.com DSV - - [1 action=POST created=2020-06-02T14:51:36.519620577Z ipaddress=192.0.2.55 path=token principal=users:tenantaame principalItemId=f18b5bda-51ea-4bfa-b272-80b12e43b676 tenant=tenant tenantName=tenantName] abcdef "
```

### Configure Syslog

To start a SIEM configuration workflow, use the command:

```
dsv siem create
```

Option	Description
name	required
siemType	required, allowed values: syslog
host	required, url
port	required, numeric
protocol	required, allowed values: tcp, udp, http, https, tls
authMethod	required, allowed values: token
auth	required
loggingFormat	required, allowed values: rfc5424

### Sample values

```
{
  "siemType": "syslog",
  "name": "syslogtest",
  "host": "54.210.93.200",
  "port": 8000,
  "protocol": "tls",
  "authMethod": "token",
  "auth": "abcdef",
  "loggingFormat": "rfc5424"
}
```

## CEF

CEF	Privilege DevOps Vault Audit	Description
Version	0	constant
Device Product	Ibm security	constant
Device Product	Privilege DevOps Vault	constant
Device Version	-	unused by Privilege DevOps Vault
Signature ID	id	audit field
Name	action	audit field
Severity	status	see below for translation
Extension		all other audit fields

Severity status	Severity
200	0
400	1
401	7
403	7
404	0
500	0
Anything else	-

### Sample CEF output

```
CEF:0|thycotic|dsv|-|b40e07d3-6fb9-41e8-9816-356de992b8fa|POST|0|{action:POST,created:2020-0602T17:52:30.841020649Z,id:b40e07d3-6fb9-41e8-9816-356de992b8fa,ipaddress:192.0.2.55,message:login succeeded,path:token,principal:users:user,principalItemId:f18b5bda-51ea-4bfa-b27280b12e43b676,status:200,tenant:tenat,tenantName:tenantName}
```

### Configure CEF

To start a SIEM configuration workflow, use the command:

```
dsv siem create
```

Option	Description
name	required
siemType	required, allowed values: cef

host	required, url
port	required, numeric
protocol	required, allowed values: tcp, udp, http, https, tls
authMethod	required, allowed values: token
auth	required
loggingFormat	required, allowed values: cef

### Sample values

```
{
  "siemType": "cef",
  "name": "syslogtest",
  "host": "192.0.2.55",
  "port": 8678,
  "protocol": "udp",
  "authMethod": "token",
  "auth": "abcdef",
  "loggingFormat": "cef"
}
```

## JSON

Privilege DevOps Vault sends 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\": \"192.0.2.55\", \"message\": \"login succeeded\", \"path\": \"token\", \"principal\": \"users:user\", \"principalItemId\": \"f18b5bda-51ea-4bfa-b27280b12e43b676\", \"status\": \"\", \"tenant\": \"tenat\", \"tenantName\": \"tenantName\"}
```

## Configure JSON

To start a SIEM configuration workflow, use the command:

```
dsv siem create
```

Option	Description
name	required
siemType	required, allowed values: json
host	required, url
port	required, numeric
protocol	required, allowed values: tcp, udp, http, https, tls

authMethod	required, allowed values: token
auth	required
loggingFormat	required, allowed values: json

### Sample values

```
{  
  "siemType": "json",  
  "name": "syslogtest",  
  "host": "192.0.2.55",  
  "port": 443,  
  "protocol": "https",  
  "authMethod": "token",  
  "auth": "abcdef",  
  "loggingFormat": "cef"  
}
```

# Release Notes

IBM Security periodically updates IBM Security Verify Privilege DevOps Vault to provide fixes and improvements and introduce features.

As a Cloud application, Privilege DevOps Vault 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 is always the same across the OS-specific editions of the CLI executable.
- You obtain these updated versions of the CLI executables by downloading them from [IBM Security Verify Privilege DevOps Vault Downloads](#).
- The CLI itself notifies you when a new version is available for download.
- Generally, older versions of CLI executables continue to work, but you want to have the latest executables to benefit from fixes and obtain new features.