

Managing Substrate Vault Secrets with Castle (CLI)

Downloaded from Epic Games Confluence

Date: 2025-07-12 04:07:21

Original URL: <https://confluence-epicgames.atlassian.net/wiki/spaces/CDE/pages/81068353>

Document Level Classification

[200](#)

- [Installation](#)
 - [Installation on MacOS with Brew](#)
- [Basic Usage](#)
 - [Install Vault CLI](#)
- [Creating Secrets](#)
- [Editing Secrets](#)
- [Writing Secrets to Local Files](#)

Starting from August 2024, it is mandatory that all new Kubernetes clusters use External Secrets Operator (ESO). We strongly recommend using ESO, but the Vault Injector sidecar will still remain available with limited support, bug fixes will be provided if needed to maintain core functionality.

Note that in the following documentation you may see references to the Secrets Layout or vault paths. The layout is different depending on which pattern you are using. This documentation references the vault paths used when using the vault-injector pattern. Reference [Using External](#)


[Secrets Operator \(ESO\) in epic-app to inject secrets](#) for the supported vault paths used with ESO.

Introduction

Castle is a custom CLI tool for working with Substrate Vault. It streamlines many of the common workflows for using Substrate Vault including creating, editing, and reading secrets, as well as templating Vault secrets into files on disk to facilitate builds and local development without checking secrets into source control.

Why use Castle? Why not use common community tools?

You are welcome to use other tools with Vault. Castle is implemented specifically to support the workflows Epic developers use with Substrate Vault, in particular handling our specific authentication methods, secrets layout, and some bonus features like templating. Many of these features are available in other community tools but we wanted to make it easy to get all the good stuff in one place, and remove a lot of features that developers don't need.

If you have trouble installing or using castle, or would like to request new features, please reach out to us in  #cloud-ops-support-ext.

Installation

| | | | |
|------------------------------|--------------------------------|------------------------------------|--|
| <u>Linux</u> | <u>Windows</u> | <u>MacOS Intel</u> | <u>MacOS Apple Silicon</u> |
|------------------------------|--------------------------------|------------------------------------|--|

Download the `castle` binary for your platform. You must be connected to the VPN in order to download these files. After download, extract the archive and put the binary somewhere easy to access (on your `PATH` is best).

Source code for castle is available at [substrate/castle](#).

Installation on MacOS with Brew


Castle is available with our substrate homebrew-tap

To use `brew tap`, you must have a public SSH key on file. [Check](#) if you do and/or read [more information](#) on adding one.

```
brew tap substrate/tap git@github.01.epicgames.net:substrate/homebrew-tap
brew install substrate/tap/castle
```

MacOS may block the file after download. You can use these commands to download and unblock it:

```
$ curl -sS https://dl.substrate.on.epicgames.com/binaries/castle.dl/latest
$ xattr -d com.apple.quarantine castle
$ sudo spctl --add ./castle
$ ./castle version
1.1
```

If you have trouble getting the binary to work, or receive a security warning from MacOS, reach out to us in  #cloud-ops-support-ext.

Basic Usage

Here is a current list of commands in `castle`.

Commands

| | |
|---|--|
| <code>login</code> | Login to Vault |
| <code>projects [<brand>]</code> | List your projects in Vault |
| <code>list <project></code> | List all secrets for a project |
| <code>read <secret> [<key>]</code> | Retrieve a secret from Vault |
| <code>open <secret></code> | Open the specified secret in the Vault UI |
| <code>template <file> [<target>]</code> | Populate a template with Vault secrets |
| <code>create [-empty] [<path>]</code> | Create a new secret using the wizard and -empty will create an empty secret |
| <code>edit <secret></code> | Edit the contents of a secret in Vault |

castle login is no longer operational due to the move to oidc for Vault authentication. Therefore, please use the instructions below to obtain a vault token that can be used with castle.

Install Vault CLI

If you do not have `vault`, you can install it with the following:

```
# NOTE: Do not bump to v1.15.0+ as license changed to BUSL-1.1
# - https://github.com/hashicorp/vault/pull/22290
# - https://github.com/hashicorp/vault/pull/22357
# 1. Uninstall the current version of vault (if it's unusable or BUSL 1
# 2. Download the new version via curl
# 3. Remove this line `disable! date: "2024-09-27", because: "will chan
# 4. Run installation, pin the version to not update it, and verify wit

$ curl -O https://raw.githubusercontent.com/Homebrew/homebrew-core/9da3
```

```
$ brew uninstall vault
$ brew install ./vault.rb && brew pin vault && vault --version
$ rm -rf ./vault.rb
```

To get started, you need a vault token. The simplest way of obtaining one is by logging in with the Vault CLI as follows:

```
$ export VAULT_ADDR=https://vault.substrate.on.epicgames.com/
$ vault login -method=oidc
```

Once you have obtained a Vault token, use `castle projects` to list all of your projects in Vault. For example:

```
$ castle projects
substrate projects:

substrate/castle
substrate/codefresh
substrate/example
substrate/k8s
substrate/sumologic
substrate/terraform-enterprise
substrate/vault
```

We can subsequently list secrets under a given project using `castle list`:

```
$ castle list substrate/example
```

Human-readable Secrets

```
substrate/example/use1a/live/human-readable-secret
```

Build Secrets

```
substrate/example/use1a/live/build/team-city-secret
```

Runtime Secrets

```
substrate/example/use1a/live/runtime/kubernetes-secret
```

Note that the different categories, Human-readable, Build, and Runtime correspond to the [Secrets Layout](#).

We can read a secret using `castle read`. Note that secrets in Vault are maps / dicts, so they can each have multiple values.

```
$ castle read substrate/example/use1a/live/human-readable-secret  
my-favorite-cookie = chocolate chip
```

```
my-favorite-dessert = cookies
```

```
$ castle read substrate/example/use1a/live/human-readable-secret my-fav  
chocolate chip
```

Note that if we pass in an extra argument we can read a specific value from a given secret. This can be useful for shell scripting. However, you can also use templates (see below).

Creating Secrets

We use a particular [Secrets Layout](#) in Substrate Vault to organize secrets.

`castle create` guides you through the process so it's easier to get the right path for your secret.

```
$ castle create
Welcome to the new secret wizard. Press Ctrl+C to abort

Brand:
  - substrate

Choose a brand: substrate

Projects:
  - castle
  - codefresh
  - example
  - k8s
  - sumologic
  - terraform-enterprise
  - vault

Choose a project: castle

Region (use1a, global, ...): global

Environment (dev | live): dev

Category (build, runtime, <blank>):

Name: new-test-secret
```

```
secret successfully created: substrate/castle/global/dev/new-test-secret
```

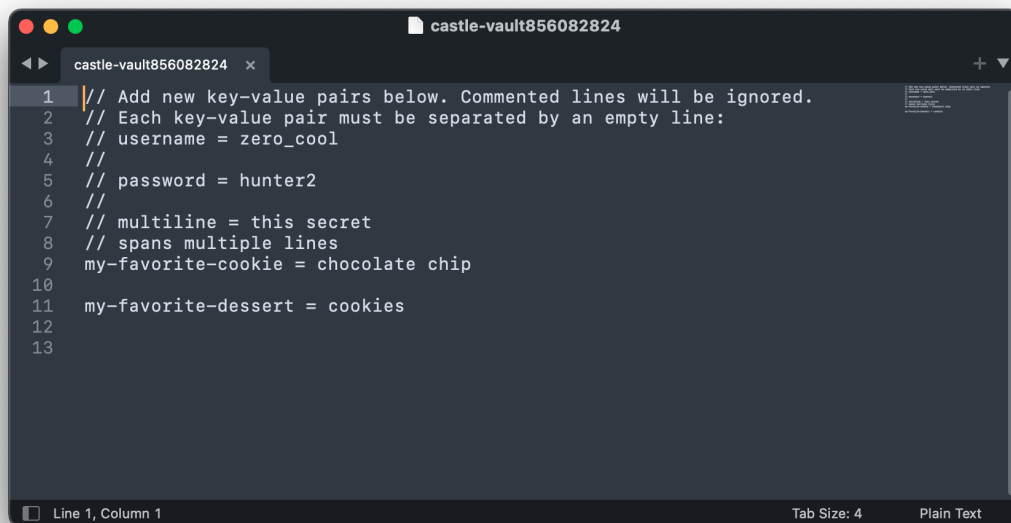
After you enter the name of the secret, castle will open the new secret in your editor. See below for details.

Editing Secrets

Secrets in Vault are maps / dicts, so they can each have multiple values.

`castle edit` retrieves a secret from Vault and opens it in your text editor. By default this is `vim` on *nix and `notepad.exe` on Windows, but you can customize this by setting the `$VISUAL` or `$EDITOR` environment variables. For example, if I want to edit this secret using Sublime Text I can specify `EDITOR=subl` inline:

```
$ EDITOR=subl castle edit substrate/example/use1a/live/human-readable-secret
```

```
1 // Add new key-value pairs below. Commented lines will be ignored.
2 // Each key-value pair must be separated by an empty line:
3 // username = zero_cool
4 //
5 // password = hunter2
6 //
7 // multiline = this secret
8 // spans multiple lines
9 my-favorite-cookie = chocolate chip
10
11 my-favorite-dessert = cookies
12
13
```

Line 1, Column 1 Tab Size: 4 Plain Text

Secrets will be written back to Vault when you save and exit the file in your editor. If you decide you don't want to make changes after all, just exit your editor without saving.

If you prefer to edit secrets using the web UI, use `castle open` instead.

```
$ castle open substrate/example/use1a/live/human-readable-secret
```

Writing Secrets to Local Files

Writing secrets from Vault to local files is a common use-case for builds and local development. Castle supports this workflow via the `castle template` command, which uses Go's text/template package to render secrets.

To start with, we'll need to create a template file. The template file uses `{{ read "path" "key" }}` to interpolate values from Vault into the file.

secrets.json.tpl

```
{
    "username": "{{ read \"substrate/castle/local/dev/example-secret\" }}",
    "password": "{{ read \"substrate/castle/local/dev/example-secret\" }}"
}
```

With this template, we can render the template using `castle template`:

```
$ castle template secrets.json.tpl
{
    "username": "cookiemonster",
    "password": "ilikecookies"
}
```

You can use output redirection to redirect this to a file on disk, or pass in a second parameter to write to that file.

```
$ castle template secrets.json.tpl > secrets.json
or
$ castle template secrets.json.tpl secrets.json
```

The template command can be used in your Makefile or build script to populate secrets from Vault into any type of text file, from JSON to environment files, packer templates, and more.

Make sure to add the rendered file to your [.gitignore](#) or [.p4ignore](#) file so the plaintext secrets are not checked into source control.

If you are using Terraform, use the [Vault provider for Terraform](#) instead of `castle template`.

Page Information:

Page ID: 81068353

Space: Cloud Developer Platform

Downloaded: 2025-07-12 04:07:21