

Conversación con Gemini

Exported: 15/6/2026 9:09:45

Jordi H.

Has dicho me gustaría conocer cómo se hace la gestión de secretos en python. la idea es tener cierta independencia y mantener, si es una opción, el servicio de vault en mi servidor. Y todo con software libr

Gemini

Gestionar secretos de forma segura es un paso crítico para cualquier analista de datos o desarrollador, especialmente cuando trabajas con credenciales de bases de datos o APIs sensibles. Para mantener la independencia y usar software libre, la combinación ganadora suele ser **HashiCorp Vault** (versión Community) junto con librerías específicas de Python. Aquí tienes una guía de cómo estructurar esta solución en tu propio servi

1. El Servidor: HashiCorp Vault

Vault es el estándar de la industria para gestionar secretos. Es **Software Libre** (bajo licencia BSL, pero gratuito para uso personal y autogestionado) y permite almacenar, cifrar y generar credenciales dinámicas

- **Persistencia:** Puedes configurarlo para que guarde los datos de forma cifrada en tu sistema de archivos o en una base de datos propia
- **Acceso:** Funciona mediante una API REST, lo que lo hace extremadamente compatible con Python.

2. Integración con Python: `hvac`

Para interactuar con Vault desde Python, la librería más robusta y utilizada es **hvac** (HashiCorp Vault API Client)

Instalación

Bash

CODE

```
pip install hvac
```

Ejemplo de flujo básico

Imagina que ya tienes tu servidor Vault corriendo en localhost:8200. Así es como recuperarías un secreto:Pytho

CODE

```
import hvac

# 1. Conexión al servidor
client = hvac.Client(
    url='http://localhost:8200',
    token='tu_token_de_acceso' # En prod, usa métodos de autenticación más seguros
)

# 2. Lectura de un secreto (Key-Value Engine v2)
read_response = client.secrets.kv.v2.read_secret_version(
    path='mi-proyecto/datos-analitica',
    mount_point='secret'
)

# 3. Acceso a los datos
credenciales = read_response['data']['data']
api_key = credenciales['api_password']

print(f"Conexión establecida con la clave: {api_key[:4]}****")
```

3. Estrategia de "Independencia"

Para no depender de que el servidor de Vault esté siempre disponible durante el desarrollo local, o para evitar exponer el "Token Maestro", se suelen usar estas prácticas

A. Autenticación por AppRole

En lugar de usar un token generado a mano, Vault permite crear un **AppRole**. Es como un usuario y contraseña para aplicaciones. Python presenta un RoleID y un SecretID para obtener un token temporal

B. Inyección por Variables de Entorno

Si no quieres que tu script de Python llame directamente a la API de Vault (para mantener el código "limpio" de lógica de seguridad), puedes usar herramientas como **envconsul**

Esta herramienta de software libre extrae los secretos de Vault y los expone como variables de entorno justo antes de ejecutar tu script: Bas

CODE

```
envconsul -secret secret/mi-proyecto python mi_analisis.py
```

Dentro de Python, solo usas `os.environ.get('DB_PASSWORD')`.

4. Comparativa de opciones Open Source

Herramienta	Rol	Ventaja
HashiCorp Vault	Servidor de Secretos	El más potente y escalable.
Infisical	Alternativa a Vault	Interfaz de usuario más moderna y
Python-dotenv	Desarrollo local	Ideal para cargar archivos .env (que
Sealed Secrets	Kubernetes	Si decides mover tu analítica a

Recomendación para tu flujo de Analítica

Como te estás centrando en **Python y Analítica de Datos**, lo ideal es que tus notebooks o scripts no contengan

ninguna clave en texto plano

1. Levanta Vault en un contenedor Docker en tu servidor.
2. Usa hvac para pruebas rápidas.
3. Para producción, usa variables de entorno inyectadas por el sistema, manteniendo tu código de Python totalmente agnóstico de dónde vienen las llaves

¿Tienes ya el servidor configurado o te gustaría saber cómo levantar una instancia básica de Vault en Docker para empezar a probar

Jordi H.

Has dicho he visto que hashicorp ha cerrado su producto de terraform y la comunidad se está migrando a tofu. No quiero tener la misma experiencia. qué debo usar que no sea de hashicorp

Gemini

Es totalmente comprensible que quieras evitar el "efecto Terraform". HashiCorp cambió su licencia de código abierto (MPL) a una de fuente empresarial (BSL), lo que generó mucha incertidumbre. Si buscas **independencia** de software puramente libre (OSI-approved), tienes dos caminos claros en 2026:

1. OpenBao: El "OpenTofu" de Vault

Si te gusta la arquitectura de Vault pero rechazas la nueva política de HashiCorp, **OpenBao** es tu solución. Es un fork nacido directamente de la comunidad y bajo el paraguas de la **Linux Foundation**

- ¿Qué es?: Es el código de Vault (versión 1.14) antes del cambio de licencia, mantenido por la comunidad.
- Ventaja: Es 100% compatible con las librerías de Python actuales (como hvac). Si tu código busca un servidor de Vault, OpenBao responderá exactamente igual
- Licencia: MPL 2.0 (Código abierto real).
- Uso en Python: Usas el mismo cliente hvac, solo cambias la URL hacia tu servidor de OpenBao.

2. Infisical: La alternativa moderna

Si buscas algo nacido con una mentalidad más actual y menos "pesada" que Vault, **Infisical** se ha convertido en el favorito de muchos desarrolladores

- ¿Qué es?: Una plataforma de gestión de secretos de extremo a extremo, diseñada para ser sencilla de usar y autogestionable
- Ventaja: Tiene una interfaz mucho más amigable que Vault y su integración con Python es nativa mediante su propio SDK
- Software Libre: Es open-source y puedes levantarlo en tu servidor con Docker en minutos.
- Python SDK: Python

CODE

```
from infisical_client import InfisicalClient

client = InfisicalClient(host="https://tu-servidor.com")
client.auth.login.kubernetes(...) # O cualquier método de auth
secret = client.secrets.get_secret(name="DB_PASSWORD", project_id="analitica")
```

3. Bitwarden Secrets Manager

Aunque Bitwarden es conocido por su gestor de contraseñas, tienen un **Secrets Manager** específico para desarrolladores que es código abierto

- Ideal si: Ya usas Bitwarden para tus contraseñas personales y quieres una infraestructura similar para tus scripts de Python
- CLI: Es muy potente para inyectar secretos en tus procesos de análisis de datos sin tocar el código.

Resumen: ¿Cuál elegir para tu servidor?

Si buscas...	Usa...	Por qué
Continuidad total	OpenBao	Es el sucesor espiritual y libre de
Simplicidad y UI	Infisical	Es más fácil de configurar para una
Minimalismo	SOPS	Si no quieres un servidor encendido

Mi recomendación: Si vienes con la idea de usar Vault, vete directo a **OpenBao**. Tendrás la potencia de un estándar industrial con la tranquilidad de que la Linux Foundation protege su carácter libre. ¿Te gustaría que te ayude con un archivo docker-compose.yml para levantar OpenBao en tu servidor
