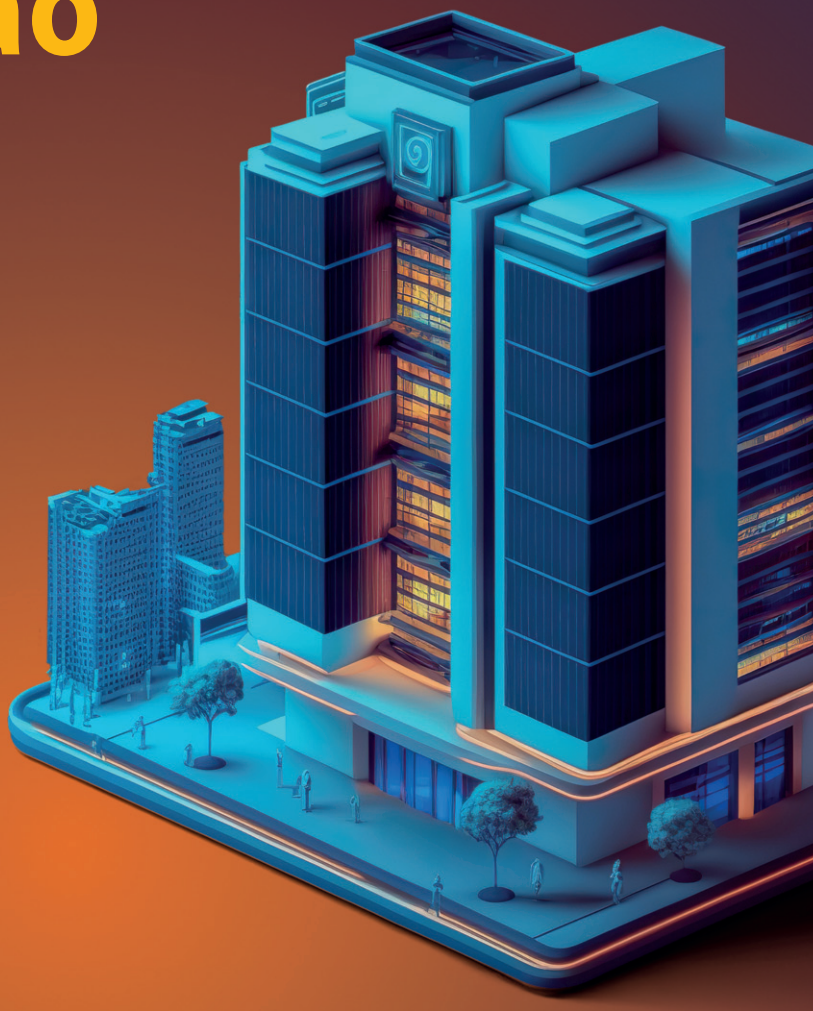
01 | 26 Reprint for Adfinis

IT Administrator
The magazine for professional system and network administration

Secrets-Management with OpenBao



www.it-administrator.de

Secret Management with OpenBao

Don’t tell anyone!

by Markus Feilner

A vault or bank safe is protected by solid walls or thick doors with high-security locks. Often, it takes several keys and several people to open such a lock. The open-source software OpenBao brings a comparable level of security for digital secrets – such as credentials, tokens, or SSH access – to enterprise IT. Our workshop will demonstrate how the software protects and manages secrets.

Source: vladimirfloyd – 123RF

OpenBao [1] originated as a fork after HashiCorp removed the open-source status from its secrets manager “Vault” and placed the software under the Business Source License. As a result, free commercial use was only possible to a very limited extent, and cloud providers without a license were barred from doing so. OpenBao is supported by former HashiCorp community members, as well as IBM and other major companies. Originally a fork of Vault 1.14, the last release under the MPL 2.0, OpenBao is now maintained by the OpenSSF as a sandbox project.

Flexible secret management

OpenBao is a tool that allows for secrets to be managed securely. The feature sets of the original and its fork are very similar today: both store sensitive data such as passwords, API keys, certificates, tokens, and database credentials; they monitor their use and centralize, automate, and secure their issuance.

A secret can be anything that is not to be made public, such as passwords, API keys, TLS certificates, SSH credentials, database credentials, or tokens for cloud providers. OpenBao offers its services via a web interface, a CLI, or an API. The API uses agents or custom scripts on client devices, and the web interface works in all modern browsers. Developer Mode makes it easier to get started and learn.

The tool stores all secrets in an encrypted “Secret Storage” and uses "Dynamic Secrets" to generate access credentials for databases or cloud services, for example – including on-demand and with automatic expiration dates. Policies and access control lists (ACLs) use fine-grained controls to determine who is allowed to access which secrets. Audit logging records all access events in order to meet compliance and security requirements. In addition, OpenBao can serve as an API-based encryption service (Encryption-as-a-Service). This can save a significant amount of cryptographic code in your own applications.

OpenBao is highly flexible: The generated secrets can expire after a specified period (Time To Live, TTL) or be automatically revoked (Secret Leasing & Revocation). For applications that require a database connection, temporary credentials can be provided, and OpenBao generates short-lived IAM tokens for accessing cloud resources such as AWS. For development teams that need access to various APIs, OpenBao manages the keys centrally, just as it does when working with a Public Key Infrastructure (PKI). Eventually, their certificates will also be automatically generated, distributed, and renewed.

Secure, temporary login credentials

For example, to allow an application temporary access to a database without having to store hard-coded credentials in the code or in a configuration file, OpenBao proceeds as follows: An app requests access to the database from the server. The tool now generates temporary login credentials for this purpose – valid for one hour, for example – and sends them to the client. They are valid for 60 minutes, after which they will expire.

This approach no longer requires hard-coded credentials, no developer has access to the actual login credentials, and all compromised data automatically expires, thus significantly enhancing security within the company. To do this, OpenBao encrypts all data in storage and uses SSL/TLS during transmission. It also features a “seal/unseal” mechanism: Upon startup, each storage unit must first be “unsealed”. Similar to a high-security safe in a bank, unsealing (i.e., essentially “opening” or “unlocking” the storage) requires one or more keys, and audit logs record both the use of these keys and every access to a secret.

To achieve this, OpenBao uses tokens that are linked to a policy. Each of these is path-based; rules restrict actions and access to the paths. With OpenBao, tokens can be created and assigned manually; a self-service option where clients can log in and receive a token is also available. However, the plan is actually to use “Auth Methods,” which allow clients to be seamlessly integrated, identified, and revoked as needed.

The OpenBao workflow consists of four phases: Once the client has been authenticated using an authentication method, a token is generated with an associated policy. To do this, OpenBao typically authenticates the client (human or machine) using trusted sources such as identity providers (OIDC or LDAP). The client then uses this token to initiate a query. If the comparison with the OpenBao security policy is successful, the client is granted access to the secrets.

The policies are a set of rules that determine which API endpoints a client can access using its OpenBao token. They grant or deny access to specific paths and processes in OpenBao, as well as to secrets, keys, and encryption functions. To this end, the Secret Manager issues a token that aligns with the client's identity and the associated policies, which the client is then permitted to use in the future.

Install OpenBao

There are several ways to test OpenBao: The software is included in many standard repositories, container images are available in popular registries, and precompiled binaries are provided. Installation from source code or via Helm charts is also supported [2]. Regardless of the method, the installation process usually follows the same pattern: download or generate binaries (or images), verify signatures, install OpenBao and unseal it (this is where the unseal key comes in), and then proceed with the configuration. Next, you need to ensure that the OpenBao server starts automatically and unseals itself.

Depending on the specific case, the configuration process involves numerous steps that vary from one instance to another: Specify the server address, port, and UI settings; the storage backend; API settings (API address and port); TLS certificates; as well as logging settings such as the verbosity level of the logs and their storage location. In addition, there are settings that define authentication and security, such as those for sealing, unsealing, and revoking the "root token".

Many organizations run OpenBao as a cluster of highly available OpenBao servers. In most cases, two or three such clusters are used as a "testing ground," an integration environment, or a production environment.

To get started quickly on Linux, it's best to install OpenBao from the standard repository; for example, on OpenSUSE, use *zypper in openbao*. The command *bao -h* provides an introduction to the wide range of CLI commands. The bao command is used to control the server, but it also performs many of the functions of a client. For each entry provided via the "-h" parameter, running the help command again displays additional details.

You can start a development/test server (with temporary content) using *bao server -dev*; exporting the "BAO_ADDR" variable will save the tester the trouble of typing it out later. Use:

```
export BAO_ADDR='http://
  localhost:8200'; bao server -dev
```

to start the server and receive a root token as well as an unseal key. The web interface is now available on port 8200 of the local computer, and you can access it in your browser at "http://localhost:8200". But be careful: All data in this instance is temporary and ephemeral – once you stop and restart the bao command, you'll be back to an empty server. In the web GUI, enter your "root token" in the "Token" field. After logging in, you will be taken to the "Secrets Engine" page, where you will find a fully functional but still fairly empty OpenBao server.

One more note about the root token: This token allows privileged and anonymous access to OpenBao and is intended solely for initial configuration and emergency ("break-glass") situations. Administrators should always use personal and restricted user accounts. In this context, "least privilege access" means that while administrators can operate the platform, they do not have access to the teams' secrets.

Sealing and Unsealing

You can access the sealing mechanism by clicking the "Seal OpenBao" button – you already received the unseal key during setup. Sealing an instance instructs the OpenBao server to stop responding to access requests until the seal is lifted. Access will only be restored once you reopen the vault using the "Unseal" command (via the CLI, web interface, or API). OpenBao servers typically start up in this locked state. The software knows where its data is stored, which storage backend to use, and so on, but it cannot decrypt the secrets. On the dev server in our example, sealing and unsealing occur automatically for training and testing purposes.

The default configuration of OpenBao uses a Shamir seal, whereby the decryp-

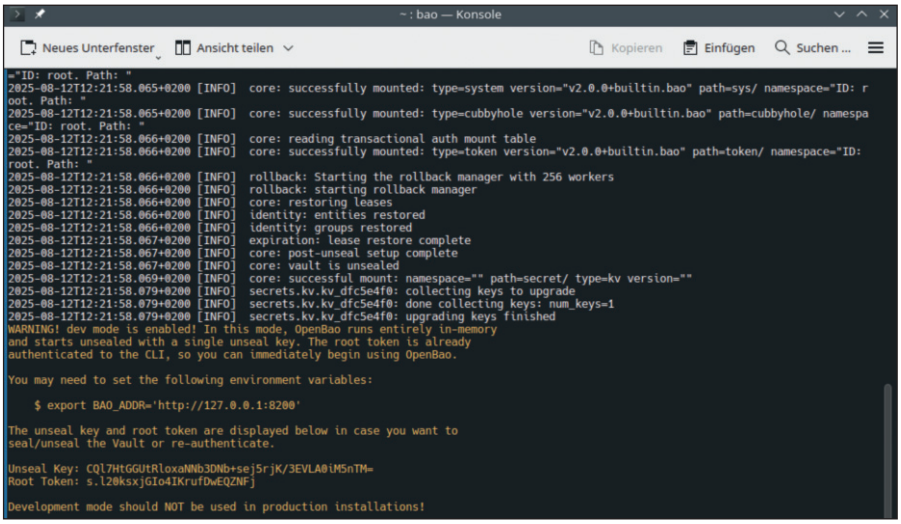


Figure 1: The first OpenBao server has been started. This is still temporary, so restarting the system will delete all data.

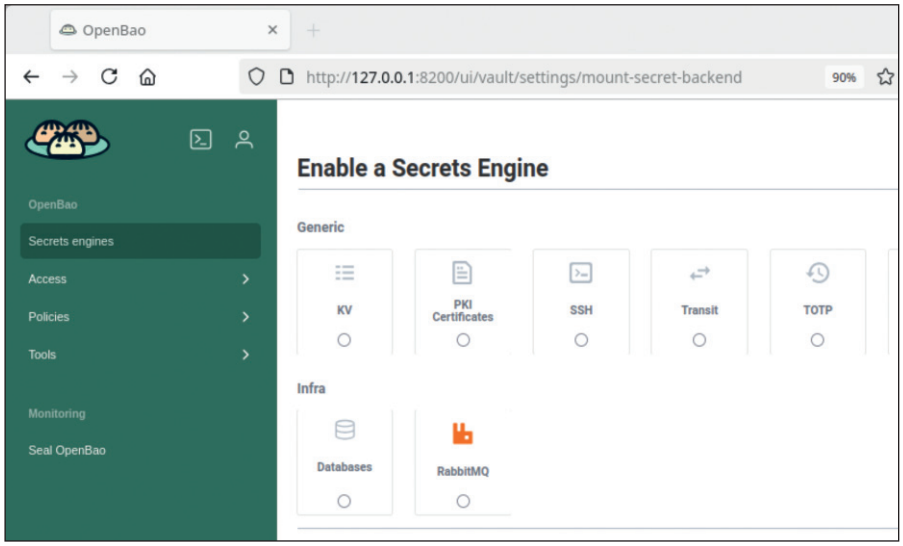


Figure 2: OpenBao offers a selection of available secrets engines in the web interface.

tion key is not distributed directly to one or more users. Instead, it is split into parts according to the "Shamir's Secret Sharing" algorithm [3]. A certain number of parts (the "threshold") is required to reconstruct the unsealing key, which is then used to decrypt the master key. The parts of the key are added one after another (in any order) until there are enough of them to reconstruct the key. The Shamir process ("Key Ceremony") is also used to generate a root token for emergency access. However, in production environments, "Auto-Unseal" is typically used to ensure that servers are operational after a reboot.

The "bao-operator" command is a great help here, as it provides a range of operations for running and managing the system:

- *bao operator init:* Initializes a new OpenBao cluster by generating a root key and preparing the storage backend.
- *bao operator unseal:* Unseals an OpenBao server.
- *bao operator seal:* Seals an OpenBao server, preventing it from performing any further operations.
- *bao operator rekey:* Generates a new set of unseal keys.
- *bao operator validate-config:* Validates the OpenBao configuration.
- *bao operator diagnose:* Enables the diagnosis of problems with OpenBao.

And if you've lost your root token and suddenly found yourself locked out, use bao operator generate-root to generate a new one. You can check the success or failure of your submissions using bao status. All commands are explained in the detailed online documentation [4].

Set up Secrets Engine

For OpenBao to function, it requires a secrets engine – i.e. a mechanism for storing secrets. Two are active on the Developer Mode server: Cubbyhole, a "per-token private secret storage," and "secret," a "key-value secret storage" (usually abbreviated as "KV" in the OpenBao context). Click "Secrets Engines" in the main menu of the browser interface to go to the "Enable a Secrets Engine" dialog box. There, you can enable features such as KV storage, PKI certificates, SSH, databases, RabbitMQ, Kubernetes, and time-limited one-time passwords (TOTP). You can do the same via the CLI using bao secrets enable ssh. The server will then display "Success! Enabled the ssh secrets engine at: ssh/" and will show a third entry for SSH under Secrets Engines in the web interface. Now it's time to configure the backend and create a role for SSH. OTP is also an option here.

Please note, however, that you must also configure the client systems in any case. With SSH, the OpenBao server can use both one-time passwords and PKI functions, for example, if the SSH key must be signed by a CA and only then is access granted. This is possible with OpenBao, but requires the appropriate settings on the clients, such as in the SSH and PAM configurations. The SSH-CA method using OpenBao is the preferred option here because it provides a way to grant temporary SSH access that is both elegant and secure, without having to store public keys or passwords on servers permanently.

In this process, OpenBao (temporarily) signs the public keys that clients use to authenticate with SSH servers. The server only accepts keys signed in such a way if it trusts the CA [5]. For this, the administrator must mark the CA public key as trusted on the server and add it to the file "/etc/ssh/trusted-user-ca-keys.pem" on the target server. To do

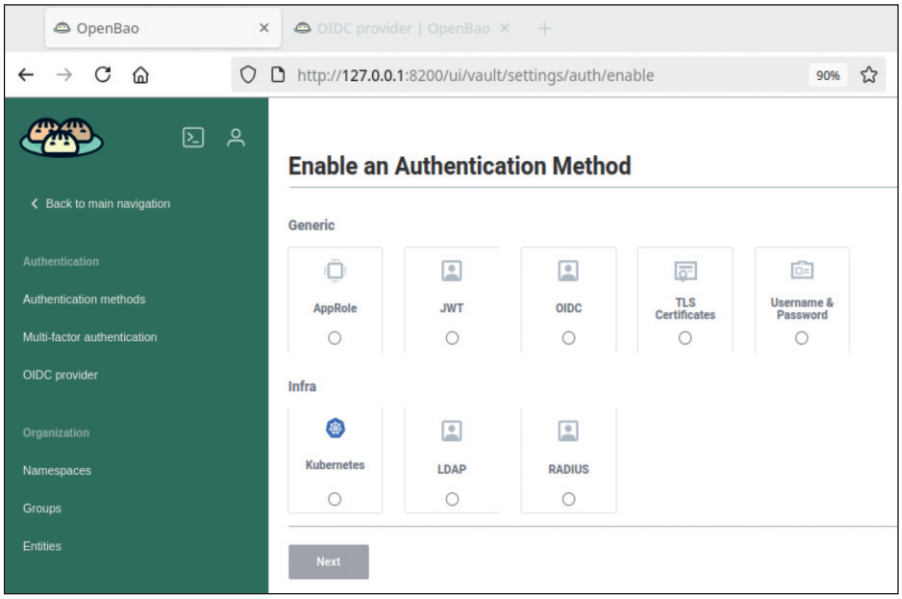


Figure 3: OpenBao also offers a wide range of available authentication methods.

this, the administrator retrieves the CA public key (using root privileges) from OpenBao:

```
bao read -field=public_key ssh/con-
  fig/ca > trusted-user-ca-keys.pem
```

Then set the following values in the SSHD configuration file "/etc/ssh/sshd_config":

```
TrustedUserCAKeys /etc/ssh/
  trusted-user-ca-keys.pem
PubkeyAuthentication yes
```

Next, restart the service using *sudo systemctl restart sshd*. Now you can test the connection using the key provided by OpenBao – if everything works, it will be able to log in to the target server without the client key ever having been stored there. However, you must manually maintain a revocation list here.

OpenBao with GitLab CI Pipelines

OpenBao includes its own OpenID Connect (OIDC) provider [6], offers roles and policies, and supports various authentication methods [7], including multi-factor authentication, OTP, namespaces, access control lists, and group-based rules. Our two SSH examples are intended only as a starting point for understanding OpenBao, as the software also allows for integration with complex mechanisms. However, caution is advised when following many of these online guides, because they use scripts to "hack" the seal/unseal mechanism or even use root tokens. You should use such procedures only for testing purposes and under no circumstances do so in production environments.

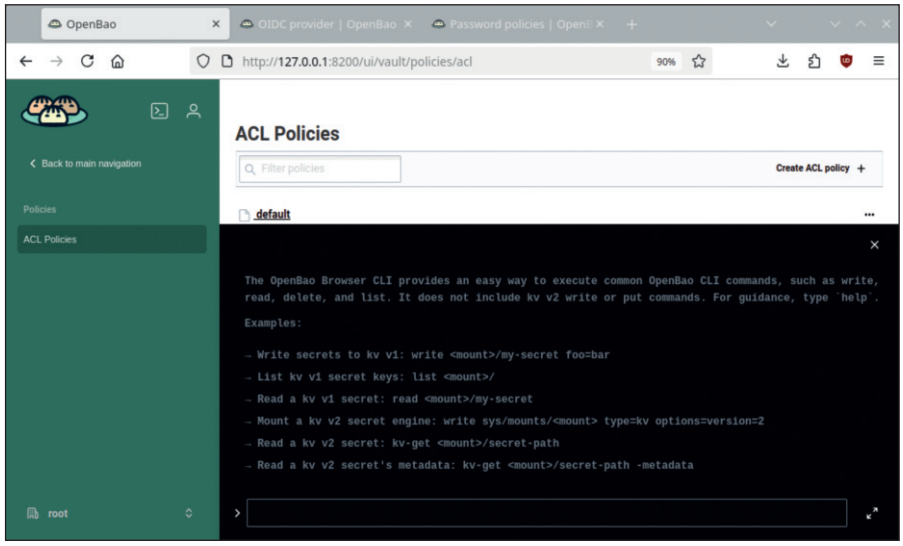


Figure 4: A CLI console can be opened simultaneously within the web interface.

To automate the development and deployment steps, many organizations today use what are known as pipelines. And because GitLab CI pipelines integrate seamlessly and securely with OpenBao, they have become the go-to choice among IT managers. During each pipeline run, the GitLab platform generates a JSON Web Token (JWT), which in turn enables strong authentication via OpenBao. To integrate such a pipeline, you must first enable and configure "OIDC" as the authentication method in OpenBao. By combining roles and policies, it is possible to configure access to GitLab groups, projects, and pipelines with a high degree of granularity. As a final step, you must add instructions to the pipeline to log it in to OpenBao using the JWT and allow it to read the desired secrets at runtime [8]. From now on, OpenBao will communicate transparently in the background with the GitLab platform to verify JWTs and check whether the requested access is permitted.

Summary

OpenBao is a powerful and highly complex tool. Despite extensive documentation and an active community, the learning curve remains steep, and new users can easily get overwhelmed at first by the multitude of configuration options, command-line tools, and features. Even the clean web interface and the well-organized CLI aren't enough to help—the range of features is simply too vast. But this also demonstrates the flexibility of the software presented in this workshop. This is how modern requirements for secrets management can be implemented using open-source software. *(jp)*

Links

[1] **OpenBao**
it-a.eu/pap21
[2] **Install OpenBao**
it-a.eu/pap22
[3] **Shamir's secret sharing**
it-a.eu/pap23
[4] **OpenBao CLI Reference**
it-a.eu/pap24
[5] **Signed SSH certificates**
it-a.eu/pap25
[6] **OIDC provider**
it-a.eu/pap26
[7] **Authentication methods**
it-a.eu/pap27
[8] **JWT/OIDC**
it-a.eu/pap28