A Vault service ends manual credential sharing in Drupal

The client keeps the secrets, the provider configures the integration without seeing them →

Manual credential storage does not scale

Common practice: secrets sit in a file outside the webroot, or as environment variables only PHP can read.

Every password change: means editing those files, and environment variables can require reloading Apache or Nginx.

Every integration: means passing that password along, from whoever manages the external service to whoever manages where the secret lives. A **chain of trust** that can break at any link.

Every new integration multiplies the risk

THE MANUAL HANDOFF

When Drupal integrates with Service A and Service B, the client contacts each provider, gets the keys, and hands them to whoever manages Drupal. The process is exposed at every handoff, and it repeats for every new service. Each rotation drags along coordination with the provider, verification across environments, and a deployment.

THE COST

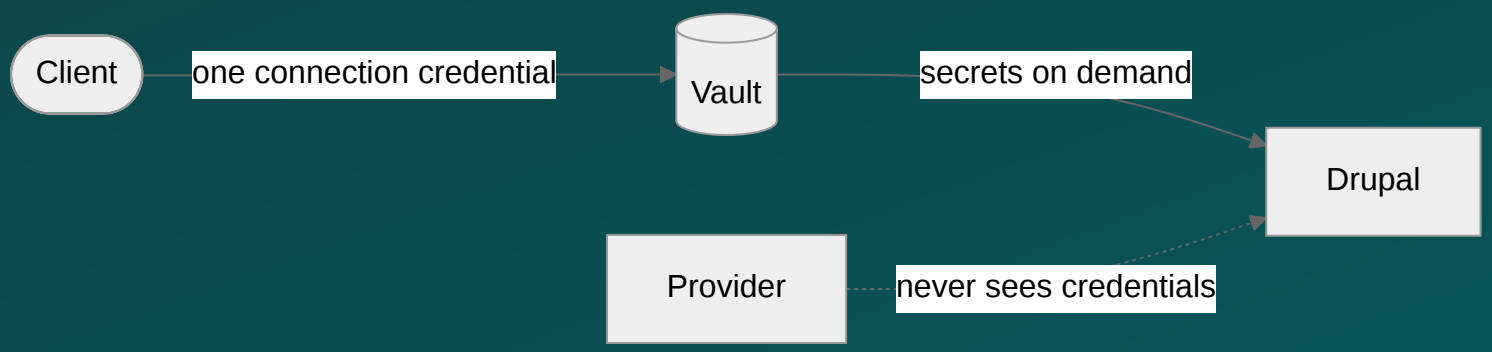
Coordinate, verify, deploy.

Every rotation, every service.

A Vault service becomes the trusted go-between

The client shares **one connection credential** for the Vault, never the credentials of each service. From then on, Drupal pulls each service's secrets directly from the Vault.

Configuring Drupal never requires seeing them.



What happens when credentials expire

WITHOUT VAULT

Every rotation means the client contacts the provider, who then updates Drupal by hand.

WITH VAULT

The client rotates credentials inside their own Vault. Drupal picks up the new secret automatically, with **zero action** from the provider.

Access can be scoped to exactly one provider

> A new service means new credentials in the Vault, granted only to the role_id that needs them.

Every other provider stays **locked out,** by design.

What running a Vault actually takes

Ownership: self-hosted or managed, the secrets never leave the client's control. If no Vault exists yet, standing it up is part of the project.

Availability: Drupal depends on the Vault at runtime, so it needs monitoring like any other critical service.

Cost: running one is real infrastructure work, taken on as a **deliberate security investment.**

Turning this into a real Drupal integration

Vault and Key: the base modules that connect to the Vault server and retrieve secrets as keys.

Encrypt (optional): encrypts the temporary leases Drupal holds in memory or the database.

An authentication module: approle is the most common method, though others can fit an organization better.

Modules built for Hashicorp Vault and OpenBao. Other products share the model but change the API.

Credentials arrive on demand and nothing changes hands

Once configured, adding a new service means one thing: creating a key in Drupal that points to a path in the Vault. The secret arrives on demand. No credentials change hands.

Every rotation goes from a coordinated change across client and provider to **nothing at all.**

Credential security is part of how every integration should be built

A Vault service removes the manual handoff of credentials between client and provider, cutting both the **exposure risk** and the cost of every rotation.

metadrop

Credentials nobody shares

At **Metadrop** security is part of how every Drupal integration gets built, from the first sprint.

Follow us for more on **Drupal security,** and share this with someone who might need it.

Save Share Comment
