OpenBao: Horizontally Scaling Secrets Management

OpenBao
Secrets Management

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Alex Scheel
Head of OpenBao Development
alex.scheel@control-plane.io
@cipherboy

controlplane

What is OpenBao?

OpenBao

Open Source Secrets Management[1]

OpenBao

Static secrets

- Encrypted key/value storage for sensitive application data
- Version history, metadata, and check-and-set semantics in KVv2

Dynamic secrets

- Just-in-time database credentials
- On-demand cloud identities
- TOTP generator & provider
- Rich plugin ecosystem

Encryption services

- PKI certificate issuance with ACME support
- SSH certificate & password generation
- Encryption as a service (Transit)

Governance & integrations

- Open governance under the OpenSSF
- Helm Chart, OpenTofu, CertManager, ESO, and more
- Growing ecosystem adoption

Centralized rotation

Lower exposure risk

Transport security

Better incident response

1: OpenBao is licensed under the MPL and has open governance under the OpenSSF subproject of the Linux Foundation.

pc: Alex Scheel

Same API; better operator experience

OpenBao is the open-source continuation HashiCorp Vault, keeping API compatibility but improving operator experience:

- Declarative self-initialization solves the day zero setup problem
- Pagination, transactions allow for better performance and backup recoverability
- Pluggable KMS devices, operator-defined workflows, and per-namespace sealing in OpenBao v2.6.0

OpenBao

OpenBao's TSC

OpenBao

controlplane

SAP REPLY LIQUID

Adfinis

wallix

IOTech

Who is ControlPlane?

controlplane

Cloud Native and Open Source security consultancy established in 2017. Our diverse culture empowers and develops individuals with talent and integrity.

Accustomed to work in highly-regulated environments and strategic programmes

Industry Leading

Kubernetes (first threat model for Financial Services User Group (FSUG)

Hacking Kubernetes O'Reilly, authors and Trainer

Defending Cloud Native - SANS, SEC 584 authors and trainers

Future Focussed

Linux Foundation's Security Technical Advisory Group (co-chair, CNCF)

OpenUK (CISO, pro bono)

Community organisers and collaborators (i.e. KubeCon, CloudNativeCon, and FluxCD)

ControlPlane Enterprise for OpenBao

Horizontal Scalability

1. What pieces did the fork have?
2. How did we support Raft in v2.5.0?
3. How will we support PostgreSQL and more going forward?

OpenBao

Horizontal Scalability

1. **What pieces did the fork have?**
2. How did we support Raft in v2.5.0?
3. How will we support PostgreSQL and more going forward?

OpenBao

Vault OSS's HA Mode: Cold Standby

Had in OpenBao v2.4.x

- Data replication support via HA storage backends
- Mechanism for single active leader selection
- Cold standbys which are unsealed by the operator to participate in election, potentially the data replication process
- All requests are either internally GRPC forwarded to the active or the client is redirected to the active node's advertised address

Wanted in OpenBao v2.5.x

- A way for standby nodes to serve read requests, becoming hot standbys
  - Need to define read requests similarly to HashiCorp Vault Enterprise

OpenBao

What is a Read Request for OpenBao?

HashiCorp Vault Enterprise roughly defines a read request as an operation which does not cause any storage writes.

Write (Forwarded):

- `GET /pki/crl/rotate` —> causes writes due to CRL building
- `POST /auth/userpass/login` —> causes writes due to token writing

Read-Only (Handled Locally):

- `GET /kv/my-secret` —> guaranteed no write operations on KVv1
- `POST /pki/issue/no-store-role` —> PKI can configure whether certificates are retained

OpenBao

Quirks of the Storage Model

Integrated Storage (Raft+BBolt)

- The Raft consensus protocol requires an odd number of voting nodes to handle elections
- Raft gives us a strong WAL local to all nodes: detailed information about every change is present on every OpenBao node, at the cost of continual communication with the active node.

PostgreSQL

- PostgreSQL can handle any even or odd number of nodes: a lock table is used, maintaining a row-per-lock with expiration and first-to-acquire semantics
- As PostgreSQL handles all data replication; OpenBao HA cold standbys only need to talk to their database instance
- PostgreSQL has a weaker default WAL which cannot be read on read replicas

OpenBao

Hints from the Existing Code

```
func (b *backend) invalidate(ctx context.Context, key string) {
    isNotPerfPrimary := /* ... elided ... */
    switch {
    case strings.HasPrefix(key, legacyMigrationBundleLogKey /* "config/legacyMigrationBundleLog" */):
        // This is for a secondary cluster to pick up that the migration has completed
        // and reset its compatibility mode and rebuild the CRL locally. Kick it off
        // as a go routine to not block this call due to the lock grabbing
        // within updatePkiStorageVersion.
        go func() {
            b.Logger().Info("Detected a migration completed, resetting pki storage version")
            b.updatePkiStorageVersion(ctx, true)
            b.crlBuilder.requestRebuildIfActiveNode(b)
        }()
```

https://github.com/openbao/openbao/blob/**fork-point**/builtin/logical/pki/backend.go#L519-L533

OpenBao

Invalidation must be...

- Definite, operating on specific keys
- Asynchronous

OpenBao

Horizontal Scalability

1. What pieces did the fork have?
2. **How did we support Raft in v2.5.0?**
3. How will we support PostgreSQL and more going forward?

OpenBao

Design (+Invalidation)

- Start by loading the current state on the standby node
- Allow requests to be processed rather than immediately forwarded
  - Adding semantics for conditionally forwarding when a write occurs
- Build an invalidation handler, hooking into the storage layer

OpenBao

Load the Current State

```
func (c *Core) runStandbyOnce( ... ) bool {
    // (editor) Prevent writes
    c.barrier.SetReadOnly(true)
    // (editor) Grab the local state lock (to prevent active<->standby transitions)
    if err := c.runStandbyGrabStateLock(stopCh); err != nil { ... }
    // (editor) wipe any existing state
    if err := c.preSeal(); err != nil { ... }
    // (editor) Run postUnseal to bring up all relevant subsystems
    atomic.StoreUint32(c.replicationState, uint32(... /* read-enabled standby */))
    if err := c.postUnseal(..., readonlyUnsealStrategy{}); err != nil { ... }
    // (editor) Read requests can now be handled!
```

https://github.com/openbao/openbao/blob/v2.5.3/vault/ha.go#L498-L533

OpenBao

Process Read Requests on Standbys

```
func handleLogicalInternal(...) http.Handler {
    if core.HAEnabled() && core.Standby() && !core.StandbyReadsEnabled() {
        forwardRequest(core, w, r)
        return
    }

    // (editor) Otherwise, opportunistically handle the request
    resp, ok, needsForward := request(core, w, r, req)
    switch {
    case needsForward:
        // (editor) Processing the request locally failed; send to active
        forwardRequest(core, w, r)
        return
    }
```

https://github.com/openbao/openbao/blob/v2.5.3/http/logical.go#L364-L404

OpenBao

Define an Invalidation Storage API

```go
// CacheInvalidationBackend is an extension to the standard physical
// backend to support cache invalidation. OpenBao can serve read requests
// from standby nodes, but only if the storage backend can inform standby
// nodes when writes happen on the leader (to ensure caches are flushed)
type CacheInvalidationBackend interface {
    HookInvalidate(hook InvalidateFunc)
}

type InvalidateFunc func(key ...string)
```

https://github.com/openbao/openbao/blob/v2.5.3/sdk/physical/physical.go#L64-L72

OpenBao

Dispatch Invalidations from the Storage Layer

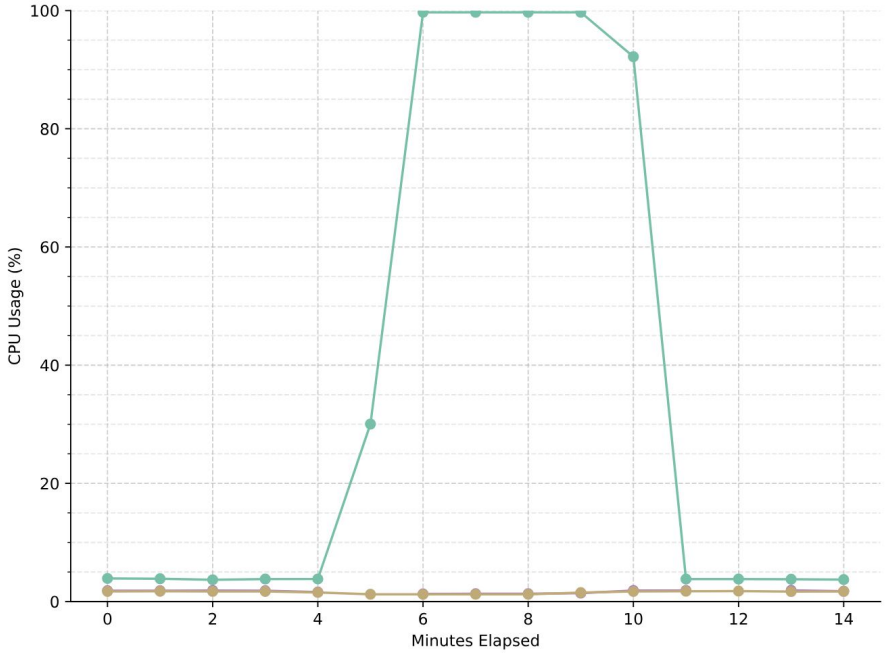
```
// invalidationManager is a long-lived subset of Core which is used to handle
// storage-level invalidations.
type invalidationManager struct {
    // Invalidate stages pending invalidations into this queue when enabled.
    enabled       atomic.Bool
    pending       []string
    // quitCh notifies that we should stop actively processing invalidations.
    quitCh      chan struct{}
    // dispatcher handles processing events from the invalidation queue to
    // subsystems. This is handled separately so that the storage layer
    // doesn't need to make asynchronous calls to the hook, while allowing
    // actual invalidation processing to take longer.
    dispatcher      *fairshare.JobManager
}
```

https://github.com/openbao/openbao/blob/v2.5.3/vault/core_cache_invalidate.go#L43-L67

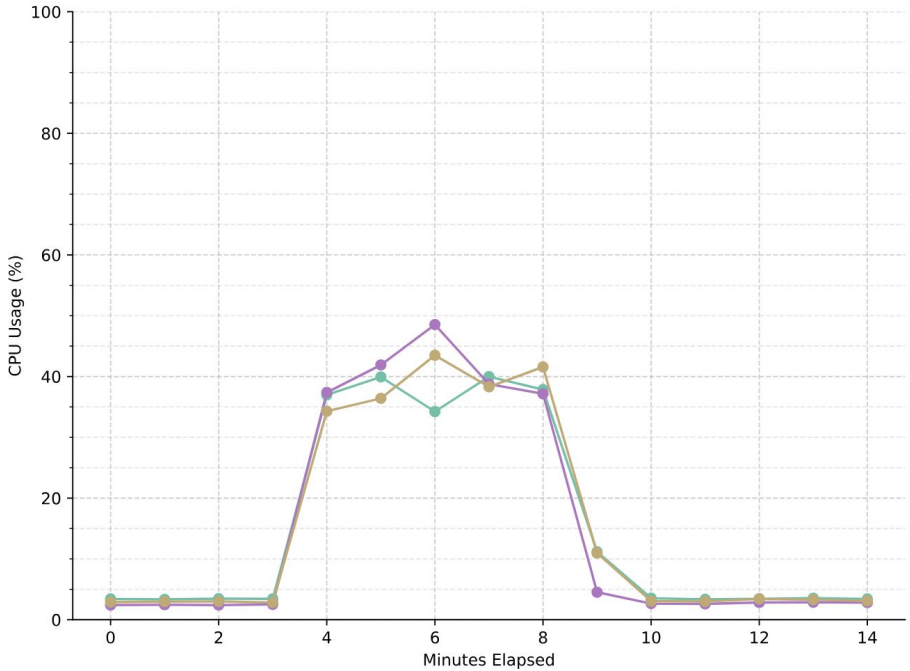
OpenBao

Results?

On 2.4.4 cluster the primary spikes to 100% CPU usage, while the standbys don't see any change (because in contrast to the KV benchmark there are no writes happening, so they have no raft updates to apply).



On the 2.5.1 cluster the load is spread pretty well at roughly 40%.



OpenBao

Read More on the Blog!

version	mean	95th%	99th%	count	rate	success
2.5.1	566.77 ms	1976.84 ms	3575.30 ms	15000	5.00	100.00 %
2.4.4	11668.62 ms	27152.59 ms	38203.77 ms	3990	1.33	99.27 %

https://openbao.org/blog/improved-horizontal-scalability/

OpenBao

Thanks to our developers!

Fatima Patel;
2025 OpenBao Mentee
Philipp Stehle;
Adfinis
Alex Scheel;
ControlPlane

Horizontal Scalability

1. What pieces did the fork have?
2. How did we support Raft in v2.5.0?
3. **How will we support PostgreSQL and more going forward?**

OpenBao

How to Match Invalidation Semantics?

OpenBao

LISTEN+NOTIFY does not span PostgreSQL read replicas. And if OpenBao disconnects, it cannot tell if it has missed events.

LISTEN+NOTIFY

OpenBao

OpenBao cannot listen to WAL replication notifications on read replicas. Cleanup is needed when a WAL watcher disconnects permanently, moving the behavior closer to Raft. And, PostgreSQL need to be configured to have `wal_level=logical` to contain enough information.

PostgreSQL WAL

OpenBao

A `modified_at` column cannot detect deletions. Maintaining a manual WAL table has the same problems with client disconnects and removal, in addition to putting garbage collection pressure on the database (as entries are short-lived on average).

Table Changes

OpenBao

TTL-Based Expiration

OpenBao relies on “list” invalidation: writes to net-new entries need to be visible to standby nodes.

The mount table and similar structures are immediately reloaded on invalidation (rather than on next use), so TTL-based expiration leads to continuous reloading which in turn exacerbates locking issues.

OpenBao

So… we'll implement a GRPC backed wait-for-index approach: standby nodes will stream invalidations from the leader (solving client tracking issues: this will be scoped to their standby state). The active node will then send writes coupled with the respective storage index for the standby to await on its local PostgreSQL instance.

How to Match Replication Semantics?

OpenBao

Good News!

This works for any (indexed) replicated storage backend!

Coming Soon™!

pc: Katherine Mayo

Roadmap

- Better consistency semantics for clients through index headers
- Per-namespace storage backends and active node definitions, allowing better write scaling

OpenBao

How can you get involved?

Weekly **community calls**:
10 AM US Central/5 PM GMT+1

@OpenBao

openbao@lists.openssf.org

#1974

Contribute to development direction ideas on **GitHub**

Working Groups

Namespaces, KMS, Horizontal Scalability, and Supply Chain Security

Like

Share or author OpenBao blog posts on social media and join our **ecosystem**

Use

Start using OpenBao inside your company, project, or team today!

OpenBao

Thank you!
Questions?

openbao.org

Alex Scheel

Head of OpenBao Development
alex.scheel@control-plane.io

@cipherboy

controlplane

OpenBao