

# Technical Architecture & Security

*Open Source Security Bastion*

Version 1.1 — January 2026 (update: 06/01/2026)

Richard Ben Aleya

## Table of Contents

- 1. Architecture Overview
- 2. Technology Stack
- 3. Microservices Architecture
- 4. Inter-Service Security
- 5. Data Model
- 6. Authentication Flow
- 7. Implementation Roadmap

## 1. Architecture Overview

The Vauban architecture adopts a hybrid model combining Axum (Tokio-based), Diesel, and Askama for the web interface and REST API, with Rust microservices for performance-critical and security-critical components. This approach allows us to benefit from Rust rapid development while ensuring maximum security and performance for connection processing.

### 1.1 Guiding Principles

- **Defense in Depth:** Multiple independent security layers
- **Zero Trust:** Authentication and authorization at every level
- **Least Privilege:** Minimum access required for each component
- **Fail Secure:** In case of error, the system denies access by default
- **Total Auditability:** All actions are tracked and timestamped

## 2. Technology Stack

### 2.1 Frontend & API Layer (Rust)

Component	Technology
Framework	Axum (Tokio-based) + Diesel + Askama
API	Axum (Tokio-based) + Diesel
Authentication	Argon2, JWT, totp-rs, oauth2, openidconnect, samael
Frontend	HTMX + Alpine.js (lightweight, progressive)
WebSocket	Axum (Tokio-based)
Cache	Valkey (sessions, cache, pub/sub)

## 2.2 Rust Services Layer

Component	Technology
Async Runtime	Tokio
SSH Proxy	russh (thrussh fork)
RDP Proxy	IronRDP
Serialization	serde + MessagePack/CBOR
gRPC	tonic (with mTLS)
Crypto	ring, rustls (TLS 1.3 only)

## 2.3 Infrastructure

Component	Technology
Database	PostgreSQL 18 (with encryption)
Message Queue	RobustMQ (persistence)
Secrets	OpenBao
Internal PKI	step-ca (Smallstep)
Sandbox	Capsicum
Logs/Metrics	OpenTelemetry → Loki/Prometheus

# 3. Microservices Architecture

The architecture is divided into 7 main services, each with a unique responsibility and clearly defined interfaces.

## 3.1 Services Overview

Service	Lang.	Responsibilities
vauban-web	Rust	<ul style="list-style-type: none"><li>User interface (admin &amp; users)</li><li>REST/GraphQL API</li><li>Web session management</li><li>Real-time dashboard</li></ul>
vauban-auth	Rust	<ul style="list-style-type: none"><li>Authentication (MFA, SSO)</li><li>LDAP/AD/OIDC integration</li><li>JWT token management</li><li>Connection audit</li></ul>

Service	Lang.	Responsibilities
<b>vauban-rbac</b>	Rust	<ul style="list-style-type: none"> <li>• RBAC engine (Casbin)</li> <li>• Policy evaluation</li> <li>• Authorization cache</li> <li>• Decision API (PDP)</li> </ul>
<b>vauban-proxy-ssh</b>	Rust	<ul style="list-style-type: none"> <li>• Transparent SSH proxy</li> <li>• Session recording</li> <li>• Key injection</li> <li>• Command filtering</li> </ul>
<b>vauban-proxy-rdp</b>	Rust	<ul style="list-style-type: none"> <li>• RDP proxy (NLA support)</li> <li>• Video session capture</li> <li>• Credential injection</li> <li>• Watermarking</li> </ul>
<b>vauban-vault</b>	Rust	<ul style="list-style-type: none"> <li>• Secret management</li> <li>• Automatic rotation</li> <li>• HSM integration</li> <li>• SSH keys/certificates</li> </ul>
<b>vauban-audit</b>	Rust	<ul style="list-style-type: none"> <li>• Log collection</li> <li>• Secure storage (WORM)</li> <li>• Search and replay</li> <li>• Real-time alerts</li> </ul>

## 4. Inter-Service Security

Communication between services follows the Zero Trust model with mandatory mutual authentication.

### 4.1 mTLS (Mutual TLS)

- Each service has its own X.509 certificate issued by the internal PKI (step-ca)
- TLS 1.3 only with modern cipher suites (AEAD)
- Automatic certificate rotation (lifetime: 24h)
- OCSP/CRL verification for immediate revocation

### 4.2 Communication Protocols

Type	Protocol	Usage
<b>Synchronous</b>	gRPC + mTLS	RBAC calls, auth verification, vault queries
<b>Asynchronous</b>	RobustMQ	Audit events, notifications, session events
<b>Streaming</b>	Bidirectional gRPC	Session recording, live monitoring

## 4.3 Service Mesh Pattern

Each Rust service integrates a lightweight sidecar implementing:

- **Circuit Breaker:** failure isolation (via tower-rs)
- **Rate Limiting:** protection against abuse
- **Retry with backoff:** resilience to transient errors
- **Distributed tracing:** request correlation (OpenTelemetry)

## 4.4 Service Authentication

In addition to mTLS, each request includes a SPIFFE (Secure Production Identity Framework for Everyone) token enabling fine-grained service identification.

# 5. Data Model

## 5.1 Main Entities (PostgreSQL)

Entity	Description
User	Users with MFA, preferences, connection history
Group	User groups (mapped from LDAP/AD)
Asset	Target servers/equipment (SSH, RDP, VNC)
Credential	Stored credentials (Vault reference, never in plaintext)
Policy	RBAC rules (Casbin format)
Session	Active and historical sessions with metadata
AuditLog	Immutable logs with cryptographic signature

## 5.2 Data Separation

- **PostgreSQL:** relational data (users, policies, metadata)
- **Vault:** secrets (passwords, SSH keys, certificates)
- **Object Storage (S3/MinIO):** session recordings
- **Valkey:** cache, web sessions, ephemeral data

# 6. Authentication Flow

## 6.1 User Authentication (Web UI)

- User accesses the Rust web interface
- Login/password entry (or SSO via OIDC/SAML)
- MFA challenge (TOTP)
- Signed JWT creation (claims: user\_id, groups, permissions)
- Refresh token stored in database, access token in HttpOnly cookie

## 6.2 Asset Connection (SSH)

- User selects a server in the interface
- vauban-web queries vauban-rbac (gRPC) to verify authorization

- If authorized, ephemeral session token creation
- WebSocket client connects to vauban-proxy-ssh
- Proxy retrieves credentials from vauban-vault
- SSH connection established to the asset
- Bidirectional streaming + recording to vauban-audit

### **6.3 Just-In-Time Access**

The system implements an approval workflow for sensitive access:

- Access request with justification
- Notification to approvers (email, other means)
- Temporary policy creation (configurable TTL)
- Automatic revocation on expiration

## **7. Implementation Roadmap**

### **Phase 1: Foundations (Q1 2026)**

- Monorepo setup (Cargo workspaces)
- PKI infrastructure (step-ca) and Vault
- vauban-auth: basic authentication + MFA
- vauban-rbac: Casbin engine with simple policies
- CI/CD with security tests (SAST, dependency audit)

### **Phase 2: SSH Proxy (Q2 2026)**

- vauban-proxy-ssh: basic connection
- Integration with vauban-vault for keys
- Session recording (asciinema format)
- Web interface with xterm.js terminal

### **Phase 3: Audit & Monitoring (Q3 2026)**

- vauban-audit: WORM collection and storage
- Real-time dashboard (WebSockets)
- Session replay
- Alerts and notifications

### **Phase 4: RDP & Enterprise (Q4 2026)**

- vauban-proxy-rdp with IronRDP
- LDAP/Active Directory integration
- SSO (OIDC/SAML)
- Documentation and deployment guides

# Appendix: Architecture Diagram

