Conceptual Design for a Multi-Tenant, Certificate-Based Authentication System Using HashiCorp Vault, Authentik, Smallstep CA, FreeRADIUS, and NetBox

---

Introduction

Designing an end-to-end, certificate-based authentication system that serves multiple tenants securely and efficiently presents a complex architectural challenge. Organizations aiming for scalability, robust auditability, and flexible automation must integrate several best-in-class tools and frameworks. This report introduces a comprehensive conceptual design that leverages **HashiCorp Vault**, **Authentik**, **Smallstep CA**, **FreeRADIUS**, and **NetBox** to realize a multi-tenant, certificate-driven authentication system.

The architecture emphasizes **HashiCorp Vault** as the central authority for credential generation and storage, **Authentik** for device enrollment leveraging OIDC, **Smallstep CA** for automated certificate issuance, **FreeRADIUS** for robust EAP-TLS authentication, and **NetBox** as the authoritative device metadata repository. Integration and configuration workflows are fully automated via **Ansible**, enabling consistent, repeatable infrastructure-as-code deployments.

The following sections explore the conceptual architecture, major component roles, detailed workflows for both automation and manual interactions, and supply a **Mermaid diagram** visualizing the Ansible automation pipeline with specific roles, subtasks, and handlers.

---

1. Architectural Overview

A well-designed multi-tenant authentication infrastructure ensures strong **isolation**, **automation**, and **observability** while scaling seamlessly across organizational teams or external customers. The following illustration summarizes the relationships among the system's core components:

(Tenants)
|
+--> [Authentik (OIDC Enrollment)]
|
[HashiCorp Vault]
/ | \
[Vault Namespaces] [PKI Engine]---[Smallstep CA]---[Certificate Issuance API]
| \
[Vault API] [Webhook/Automation]
| |
[Ansible Automation] |
| |
[NetBox (Device Metadata)]-----/

|
[FreeRADIUS (EAP-TLS)]

Within this diagram:

- **Tenant isolation** is achieved via Vault namespaces and policy constraints.
- **Automated enrollment and authentication** rely on OIDC flows, PKI APIs, and SAML/OAuth integrations.
- **All interactions and updates are orchestrated by Ansible**, communicating via REST APIs to ensure idempotent infrastructure definition and compliance.

---

2. Component Roles and Functions

The following sections elaborate on each primary system component, delving into configuration, workflows, and integration strategies tailored for multi-tenancy.

2.1 HashiCorp Vault: Multi-Tenancy and Credential Management

Vault Namespaces - Isolated Trust Domains

Vault Enterprise's **namespace** feature creates self-contained "mini-Vaults" to enforce tenant isolation, whereby secrets, policies, tokens, and identity groups are unique to each namespace. This approach supports a "Vault-as-a-Service" model, allowing delegated administration to tenant-specific teams without risk of secrets leakage or policy collision[2][3].
Namespaces provide:

- **Per-tenant Auth Methods**: Separate authentication backends (OIDC, LDAP, etc.) can be configured in each namespace.
- **Isolated PKI Engines**: Certificate issuance is scoped uniquely for each tenant, ensuring tenants cannot retrieve or revoke certificates outside their namespace.
- **Strict Access Controls**: ACLs are enforced per namespace, supporting customer- or team-specific compliance regimes.

Namespace management can be performed via both CLI and API:

```
# Create a root namespace for 'engineering'
vault namespace create engineering

# Create a sub-namespace for 'dev' team
vault namespace create -namespace=engineering dev
```

API Usage Example:

```
curl --header "X-Vault-Token: $VAULT_TOKEN" --request POST \
$VAULT_ADDR/v1/sys/namespaces/<namespace_name>
```

Tokens are namespace-scoped; authentication via the API or UI requires specifying the target namespace either with the X-Vault-Namespace header or in the request path[2].

PKI Secrets Engine - Certificate Issuance and Lifecycle

Vault's **PKI secrets engine** dynamically generates, signs, and manages X.509 certificates needed for device (client/server) authentication, bypassing manual CSR and signing workflows commonly associated with legacy CAs. Key features addressing multi-tenant requirements include:

- **Ephemeral/Short-lived Certificates**: Reduces the attack surface and eliminates heavy revocation record maintenance.
- **Root/Intermediate CA Management**: Each namespace (tenant) can operate its own CA hierarchy, further enforcing trust separation.
- **Policy-based Issuance**: Role configurations strictly define allowed domains, key usages, extensions, and maximum TTL per tenant[5][6].

Certificate issuance workflow:

```
# Enable PKI engine in specific namespace
VAULT_NAMESPACE=engineering/dev vault secrets enable -path=pki pki

# Tune the engine for long TTLs
VAULT_NAMESPACE=engineering/dev vault secrets tune -max-lease-ttl=8760h pki

# Generate a Root CA
VAULT_NAMESPACE=engineering/dev vault write pki/root/generate/internal \
common_name="Engineering Dev Root CA" ttl=8760h

# Define tenants' PKI role (limitations)
VAULT_NAMESPACE=engineering/dev vault write pki/roles/dev-team \
allowed_domains=dev.example.com \
allow_subdomains=true max_ttl=168h

# Issue a certificate
VAULT_NAMESPACE=engineering/dev vault write pki/issue/dev-team \
common_name=device01.dev.example.com
```

API-based certificate management, renewal, and revocation are equally supported.

Integration with Authentik (OIDC) and External APIs

Vault's extensibility allows configuration of authentication backends per-namespace, including OIDC integrations with Authentik. Policies and external group bindings ensure policy decisions and authorization are controlled and auditable across domains[8].

2.2 Authentik: OIDC-Based Device Enrollment

Authentik is a fully open-source identity provider capable of managing authentication for users, devices, and services via OAuth2/OIDC. In this architecture:

- **Device/Entity Enrollment**: Users or devices initiate enrollment using Authentik's OIDC device or browser-based flows, binding identity claims to device metadata.
- **Group and Policy Mapping**: OIDC scopes and claims define authorization and mapping to Vault identities for granular role assignments.
- **Integration with Vault**: Authentik acts as the upstream OIDC identity provider for Vault's oidc authentication method, enabling seamless single sign-on and policy enforcement[8][9].

A typical enrollment flow:

1. Device triggers device code flow with Authentik, receiving a device code and verification URI.
2. User authenticates (in browser or mobile) against Authentik and grants device authorization.
3. Authentik issues an OIDC access token including requested claims/groups.
4. The device exchanges the code for a token, which is then used to log in to Vault's OIDC backend in the appropriate namespace.

Authentik-Vault OIDC integration steps (abridged for clarity)[7]:

- Create OIDC Provider and Application in Authentik, with required redirect URIs.
- Enable oidc method in Vault and configure with Authentik's discovery URL, client ID/secret.
- Map groups via OIDC claims for authorization.
- Assign namespace-specific roles and policies.

The result is automated, secure device onboarding leveraging both user and device identity, with full policy enforcement in Vault.

2.3 Smallstep CA: Automated Certificate Issuance

Smallstep CA ("step-ca") is a cloud-native certificate authority designed for automation and integration in dynamic environments, such as microservices, containerized infrastructure, or multi-tenant SaaS. It fits into the architecture in several ways:

- **API-Driven Certificate Lifecycle**: Exposes JSON/HTTPS API endpoints for programmatically requesting, renewing, and revoking certificates[11].
- **Provisioners for Trust Onboarding**: Supports various provisioners (OIDC, JWK, cloud IIDs) for tightly controlling and automating identity and trust.
- **Short-lived Certificates and Passive Revocation**: Encourages configurations that minimize the risk and overhead of revoked but unexpired certificates.
- **Integration with Vault and Ansible**: Can serve as a downstream CA for Vault's PKI engine, or as a direct certificate issuer for EAP-TLS clients, orchestrated by Ansible modules or roles.

Typical certificate issuance workflow:

1. Device or system bootstraps trust to Smallstep CA by downloading root certificate and verifying against the CA's fingerprint.
2. Device authenticates using OIDC/JWK token (via Authentik or automation tools).
3. Device generates a keypair and CSR, submits to the CA via API or CLI.

4. CA validates identity and issues a signed certificate.
5. Renewal is automated using the Short-lived certificates or active renewal commands.

Sample Ansible role usage (with maxhoesel.smallstep.step_ca collection)[13]:

```
- hosts: ca-servers
roles:
- role: maxhoesel.smallstep.step_ca
step_ca_name: "Org Internal CA"
step_ca_root_password: "supersecret"
```

For clients:

```
- hosts: clients
roles:
- role: maxhoesel.smallstep.step_bootstrap_host
step_bootstrap_ca_url: "https://ca.org.internal"
step_bootstrap_fingerprint: "{{ ca_fingerprint }}"
```

And for issuing certificates:

```
- name: Issue cert for deviceA
maxhoesel.smallstep.step_ca_certificate:
ca_url: https://ca.org.internal
root: /etc/step-ca/certs/root_ca.crt
subject: deviceA.example.org
...
```

This setup streamlines device provisioning and renewal, ensuring every device's identity is proactively managed.

2.4 FreeRADIUS: EAP-TLS Device Authentication

FreeRADIUS is the world's most widely deployed RADIUS server, supporting extensible authentication protocols including EAP-TLS. Its role in this system:

- **EAP-TLS Authentication**: Provides certificate-based authentication for network access (wired or wireless), leveraging certificates issued by Smallstep CA or Vault PKI.
- **Multi-Tenancy via Policies**: Different EAP-TLS configuration profiles or additional virtual servers can ensure tenant separation of authentication domains.
- **Certificate Validation**: Validates presented client certificates (from Smallstep CA/Vault PKI), ensures CRL/OCSP compliance, and can map device identities to network VLANs or access policies.

Workflow:

1. Device presents certificate to RADIUS server during EAP-TLS handshake.
2. FreeRADIUS validates the certificate chain, checks CRL/OCSP for revocation.

3. On successful validation, network access is granted and device metadata (e.g., MAC, tenant group) can be cross-referenced with NetBox or Vault policies.
4. Periodic re-authentication and session termination can be configured for additional security.

Configuration summary:

- Deploy EAP-TLS profile; link CA root/intermediate, client, and server certificates.
- Tune revocation and cipher parameters for compliance.
- Integrate with device metadata source (e.g., NetBox, optionally via custom Python scripts) for adaptive authorization.

Example EAP-TLS settings (in /etc/raddb/mods-available/eap**)**[15][16]:

```
eap {
default_eap_type = tls
tls-config tls-common {
private_key_file = /etc/ssl/private/radius.key
certificate_file = /etc/ssl/certs/radius.crt
ca_file = /etc/ssl/certs/ca.crt
...
}
}
```

Revocation lists and renewal logic are updated in tandem with Smallstep CA or Vault PKI workflows.

2.5 NetBox: Device Metadata Management

NetBox is an extensible, API-driven source of truth for network automation and device metadata. Integration points in this architecture include:

- **Device Inventory and Metadata Store**: Maintains authoritative records on device lifecycle, physical/logical location, ownership, and custom metadata required for authentication and access policy determination.
- **API Integration**: Exposes REST endpoints for querying and updating device objects, custom fields (e.g., certificate serials, enrollment status), and relationships.
- **Dynamic Inventory for Ansible**: The netbox.netbox.nb_inventory plugin imports device inventory and host variables directly to Ansible playbooks for automation.

Common workflows:

- Device registration and metadata update as part of the onboarding process (via Ansible).
- Enrichment of device records with certificate serials, expiration dates, and status.
- Conditioning of FreeRADIUS or network policy assignments based on device data fetched from NetBox.

Example NetBox API interaction:

- name: Create Device in NetBox
netbox.netbox.netbox_device:
netbox_url: "https://netbox.example.com"
netbox_token: "{{ netbox_api_token }}"
data:
name: device01
site: "HQ"
device_role: "Workstation"

REST API Example[18]:

curl -H "Authorization: Token $TOKEN" \
https://netbox.local/api/dcim/devices/

NetBox's automation-ready integrations facilitate closed-loop updates from certificate issuance, renewal, and revocation workflows.

---

3. End-to-End Workflow: Device Enrollment to Authenticated Access

The architecture's **end-to-end device workflow** brings together all primary components, orchestrated by Ansible. The canonical flow for enrolling a new device, issuing credentials, and enabling authenticated network access includes:

1. **Device OIDC Enrollment Initiation**
   - Triggered via Authentik device flow, user authenticates, and policy/claims are recorded.
2. **Device Metadata Update in NetBox**
   - Ansible playbook updates NetBox with the initial device record and links identity/group membership.
3. **Vault Credential Generation (if needed)**
   - Vault PKI engine generates ephemeral certificate or secrets, possibly integrating with Smallstep CA as issuing authority, within tenant namespace.
4. **Automated Certificate Issuance**
   - Device generates CSR and requests certificate via Smallstep CA, authenticating with OIDC/JWK token or delegated Vault policy.
   - Certificate (and key) are securely stored on the device; serial and expiry are pushed back to NetBox.
5. **FreeRADIUS Configuration Sync**
   - Certificates and revocation lists are updated on the RADIUS server(s); EAP-TLS profile configured for new tenant or device.
   - Policy and VLAN mapping may be fetched live via NetBox REST API.
6. **Network Authentication and Access**
   - Device connects to network; FreeRADIUS validates its certificate.

- Upon successful authentication, device is granted appropriate VLAN or access rights.

7. **Ongoing Lifecycle Operations**
   - Certificates are renewed via automation.
   - Revoked or expired certificates are synchronized between Smallstep CA/Vault and FreeRADIUS.
   - Device decommissioning updates NetBox, triggers cert revocation.

---

4. Ansible Automation and Mermaid Workflow Diagram

The entire lifecycle is orchestrated by **Ansible**, using modular roles and tasks for stepwise, auditable configuration and API communication. The following Mermaid diagram illustrates a conceptual workflow, highlighting roles, subtasks, variables, and handlers[19][21].

```
flowchart TD
subgraph Ansible_Playbook
VAULT[Role: Vault Namespace Setup]
VAULT_POLICIES[Role: Vault Policy Management]
PKI[Role: Vault PKI/Smallstep CA Integration]
AUTHENTIK[Role: Authentik OIDC Enrollment]
NETBOX[Role: NetBox Device Metadata]
FREERADIUS[Role: FreeRADIUS Configuration]
end

VAULT --> VAULT_POLICIES
VAULT_POLICIES --> PKI
PKI --> AUTHENTIK
AUTHENTIK --> NETBOX
NETBOX --> FREERADIUS

subgraph PKI_Details
PKI_SETUP["Task: Enable PKI Engine"]
PKI_ISSUE["Task: Issue/Sign Certificates"]
PKI_RENEW["Handler: Schedule Renewal"]
end
PKI --> PKI_SETUP
PKI_SETUP --> PKI_ISSUE
PKI_ISSUE --> PKI_RENEW

subgraph Authentik_Flow
AUTH_ENROLL["Subtask: Device Enrollment via OIDC"]
AUTH_TOKEN["Task: Issue OIDC Token"]
end
```

```
AUTHENTIK --> AUTH_ENROLL
AUTH_ENROLL --> AUTH_TOKEN

subgraph NetBox_Sync
NB_UPDATE["Task: Device Record/Inventory Update"]
NB_META["Subtask: Write Certificate Metadata"]
end
NETBOX --> NB_UPDATE
NB_UPDATE --> NB_META

subgraph FreeRADIUS_Config
FR_CERTS["Task: Sync Certs/CRL"]
FR_EAPTLS["Subtask: EAP-TLS Profile Settings"]
FR_HANDLERS["Handler: Restart RADIUS Service on Change"]
end
FREERADIUS --> FR_CERTS
FR_CERTS --> FR_EAPTLS
FR_EAPTLS --> FR_HANDLERS
```

This diagram communicates:

- **Role separation** for logical modularity and scalability.
- **Flow dependency**, ensuring prerequisites are satisfied (e.g., Vault policies must exist before PKI setup).
- **Subtasks and handlers** to dynamically respond to events such as certificate renewals, device onboarding, or policy changes.
- **Variable passing** (not shown for brevity) between tasks via Ansible's group or host variable structures and contextual facts.

In real-world playbooks, each role contains templates, task files, handlers (e.g., for restarting daemons), and default/overridden variables for maximum reusability[23].

---

5. Detailed Configuration and Implementation Recommendations

Below are recommended practical considerations and strategies to implement and operate this proposed architecture efficiently.

Multi-Tenancy Best Practices

- **Namespace and Policy Segmentation (Vault):** Use hierarchical namespaces and tightly scope ACLs to ensure zero cross-tenant credential leakage.
- **Tenant-level Provisioners and Enrollment Flows (Smallstep CA, Authentik):** Assign provisioners, OIDC providers, and access roles per tenant.

- **API Token Scoping (NetBox, Vault):** Create dedicated API tokens with minimal required permissions and, where available, restrict by IP or tenant.
- **RBAC/ABAC Enforcement:** Use OIDC claims (from Authentik) and integration policies in Vault, FreeRADIUS, and NetBox to centralize and automate access control enforcement[25].

Automation and DRY Principles

- **Ansible Role Reusability:** Structure each role for encapsulation; use templates and defaults for variables.
- **Idempotency in API Calls:** Ensure re-running playbooks is safe, with all APIs supporting PATCH/PUT for update-or-create semantics.
- **Handlers for Event-Driven Updates:** Configure handlers to automatically reload/restart services in reaction to certificate changes or policy updates.
- **Dynamic Inventory:** Integrate NetBox's dynamic inventory as the canonical automation source, reducing config drift between device database and reality.

Observability and Security

- **Audit Logging:** Enable logging (with unique request tracking) across Vault, FreeRADIUS, Smallstep CA, and NetBox API interactions to support traceability and compliance.
- **Certificate Revocation and Renewal Policies:** Automate renewal schedules (e.g., via Smallstep's step ca renew in daemon mode) and enforce short-lived certificates; use CRL/OCSP integration in FreeRADIUS.
- **Incident Response:** Leverage webhooks (e.g., from NetBox or Vault) for rapid notification of certificate expiry, revocation, or access policy changes.

---

6. Summary Table: Component Integration Overview

Component	Role in Architecture	API Integration	Ansible Role/Module Example	Multi-Tenant Feature
HashiCorp Vault	Secure credential/certificate storage	REST API, PKI, Policy	hashicorp_vault_setup	Namespaces, per-tenant ACL
Authentik	OIDC identity/device enrollment	OIDC, OAuth2 API	authentik_enrollment	Tenant providers/flavors
Smallstep CA	Automated certificate authority	Step CA REST API	smallstep_ca_integrate	Provisioners per tenant
FreeRADIUS	EAP-TLS authentication engine	RADIUS, local config	freeradius_install	VLAN/realm segmentation

NetBox	Device metadata management + API inventory	REST API, dynamic inventory	netbox_metadata	Tagging, custom fields

Each integration point is designed to support automation, scalability, and robust tenant isolation.

---

7. Conclusion

By harmoniously combining the strengths of Vault, Authentik, Smallstep CA, FreeRADIUS, and NetBox, this architecture delivers a **secure, scalable, and fully automated multi-tenant certificate-based authentication solution**.
Core benefits realized include:

- Uncompromised tenant isolation-ensured by Vault namespaces, policy scoping, and OIDC multi-provider support.
- Secure device lifecycle automation-via OIDC-based device enrollment, X.509 certificates for EAP-TLS, and passive/active certificate revocation.
- Centralized source of truth and observability-by leveraging NetBox's modern API-backed device inventory and integration with automation tools.
- Extensibility and openness-incorporating best practices from the most trusted open-source authentication, authorization, and automation systems, facilitating future growth and integration.

The conceptual design and workflow-including the detailed **Mermaid diagram**-can serve as the foundational blueprint for further detailed implementation, configuration, and continuous improvement in your organization’s authentication infrastructure.

End of Comprehensive Report

References (25)

1. *Secure multi-tenancy with namespaces - Vault* . https://developer.hashicorp.com/vault/tutorials/enterprise/namespaces
2. *Learn How to Run a Multi-tenant Vault with the New Namespaces Feature*. https://www.hashicorp.com/en/resources/multi-tenant-vault-namespaces
3. *PKI - Secrets Engines* . https://docs.devnetexperttraining.com/static-docs/HashiCorp-Vault/docs/secrets/pki/
4. *Using HashiCorp Vault’s PKI Secret Engine*. https://docs.quarkiverse.io/quarkus-vault/dev/vault-pki.html
5. *Hashicorp Vault - authentik*. https://version-2024-8.goauthentik.io/integrations/services/hashicorp-vault/

6. *OAuth2/OIDC Provider* . https://deepwiki.com/goauthentik/authentik/3.4-oauth2oidc-provider
7. *Integrate with Hashicorp Vault - authentik*. https://integrations.goauthentik.io/security/hashicorp-vault/
8. *Smallstep's step-ca as CA with ACME support*. https://www.networktechguy.com/smallsteps-step-ca-as-ca-with-acme-support/
9. *ansible-collection-smallstep/roles/step_ca/README.md at main ...*. https://github.com/maxhoesel-ansible/ansible-collection-smallstep/blob/main/roles/step_ca/README.md
10. *EAP-TLS: Certificate-based authentication - FreeRADIUS*. https://www.freeradius.org/documentation/freeradius-server/3.2.8/tutorials/eap-tls.html
11. *FreeRadius EAP-TLS configuration - Alpine Linux*. https://wiki.alpinelinux.org/wiki/FreeRadius_EAP-TLS_configuration
12. *API and Integration* . https://deepwiki.com/netbox-community/netbox/7-api-and-integration
13. *NetBox Integrations* . https://netboxlabs.com/docs/console/netbox-integrations/netbox-ansible-collection/
14. *GitHub - teramako/playbook2uml: Create a PlantUML/Mermaid.js State ...*. https://github.com/teramako/playbook2uml
15. *Ansible Roles: Best Practices and Examples*. https://learnansible.dev/article/Ansible_Roles_Best_Practices_and_Examples.html
16. *Architectural approaches for identity in multitenant solutions*. https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/approaches/identity