Sampurna Secure Vault (SSV)

INDEX

Serial Number	Topic Name
1.	Problem Statements
2.	Requirements / Objective
3.	Structure
4.	Maintenance
5.	Tool Selection

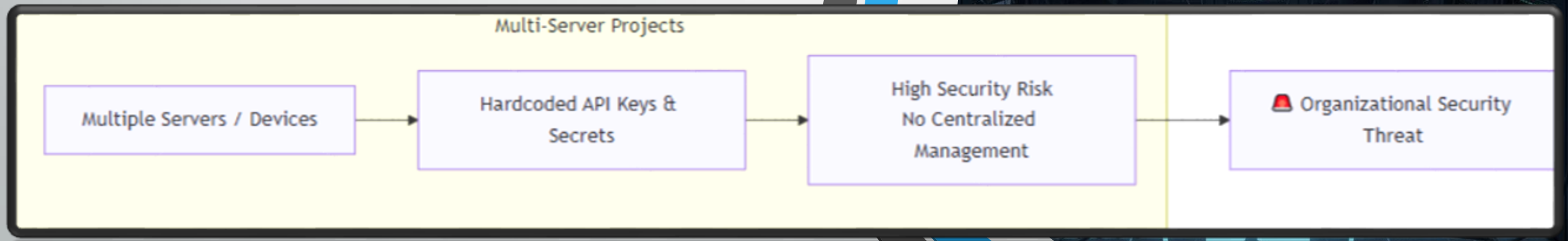
1. Problem Statements

1.1 Multi-Server Project Credentials

Across multiple servers and projects, sensitive credentials (Tenant ID, Client ID, Client Secret, etc.) are currently hardcoded in scripts.

This leads to:

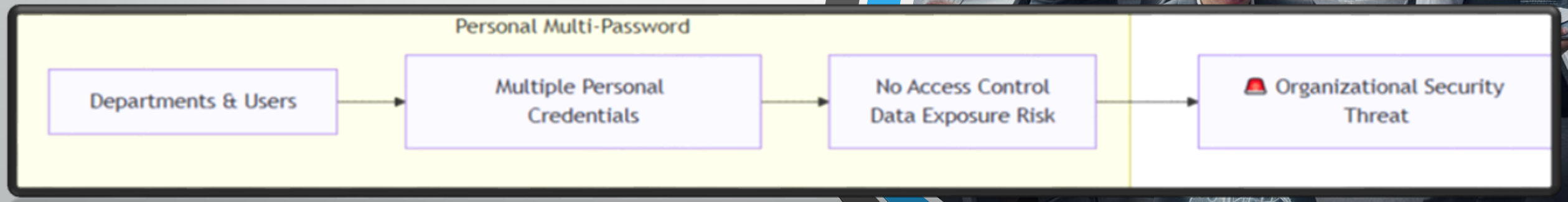
- Data leaks or accidental exposure (e.g., in GitHub)
- Device-specific environment issues
- Complex credential rotation and maintenance
- 🔒 **Core Issue:** No centralized, secure credential management system.

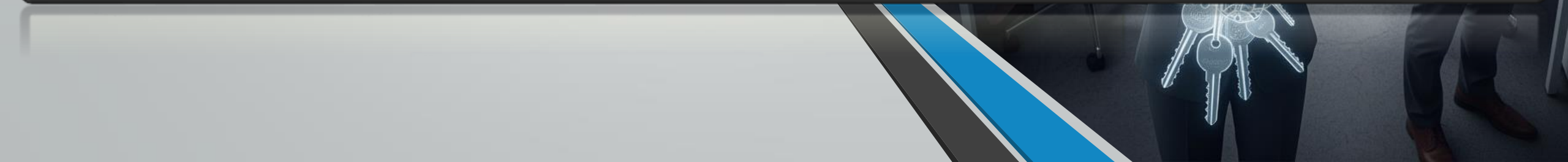


1.2 Personal Multi-Password Challenge

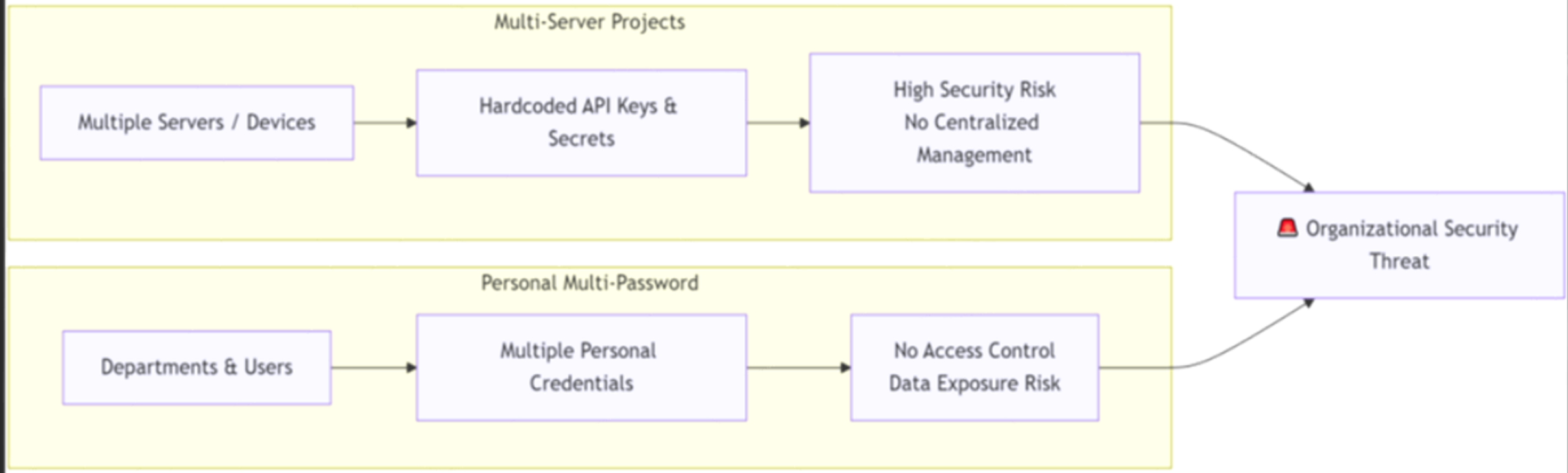
Users across departments (HR, IT, Accounts, etc.) maintain multiple personal credentials on their devices. This causes:

- Security risks if devices are compromised
- No access control or audit visibility
- Difficulty in recovery and credential updates
- 🔒 **Core Issue:** No isolated, role-based credential storage for users.





SCMS – Problem Overview (Visual Flow)

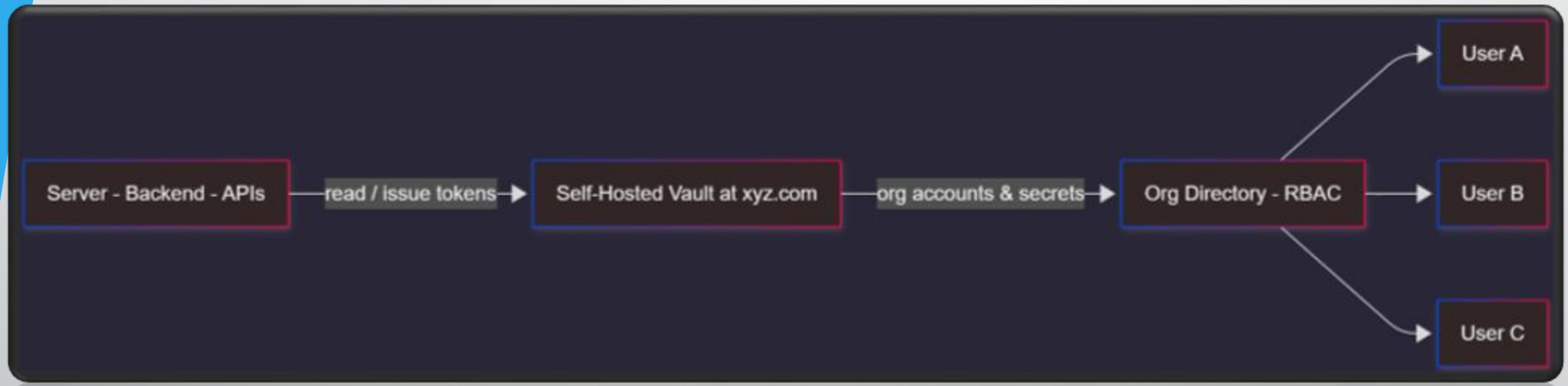


2) Requirements / Objective

- 2.1 Primary Objective

Establish a **self-hosted, centralized credential vault** that is **free to use** and **low-maintenance**, integrating with our servers and users while preventing secret exposure.

2.2 High-Level Flow (Target)



- **Server/Backend (APIs):** Services programmatically fetch **time-bound secrets/tokens** from the vault, never storing raw secrets in code.
- **Vault Frontend (xyz.com):** Web UI for admins/users to manage secrets, with org accounts provisioned.
- **Multiple Users:** Users authenticate to view only what their role/policies allow.

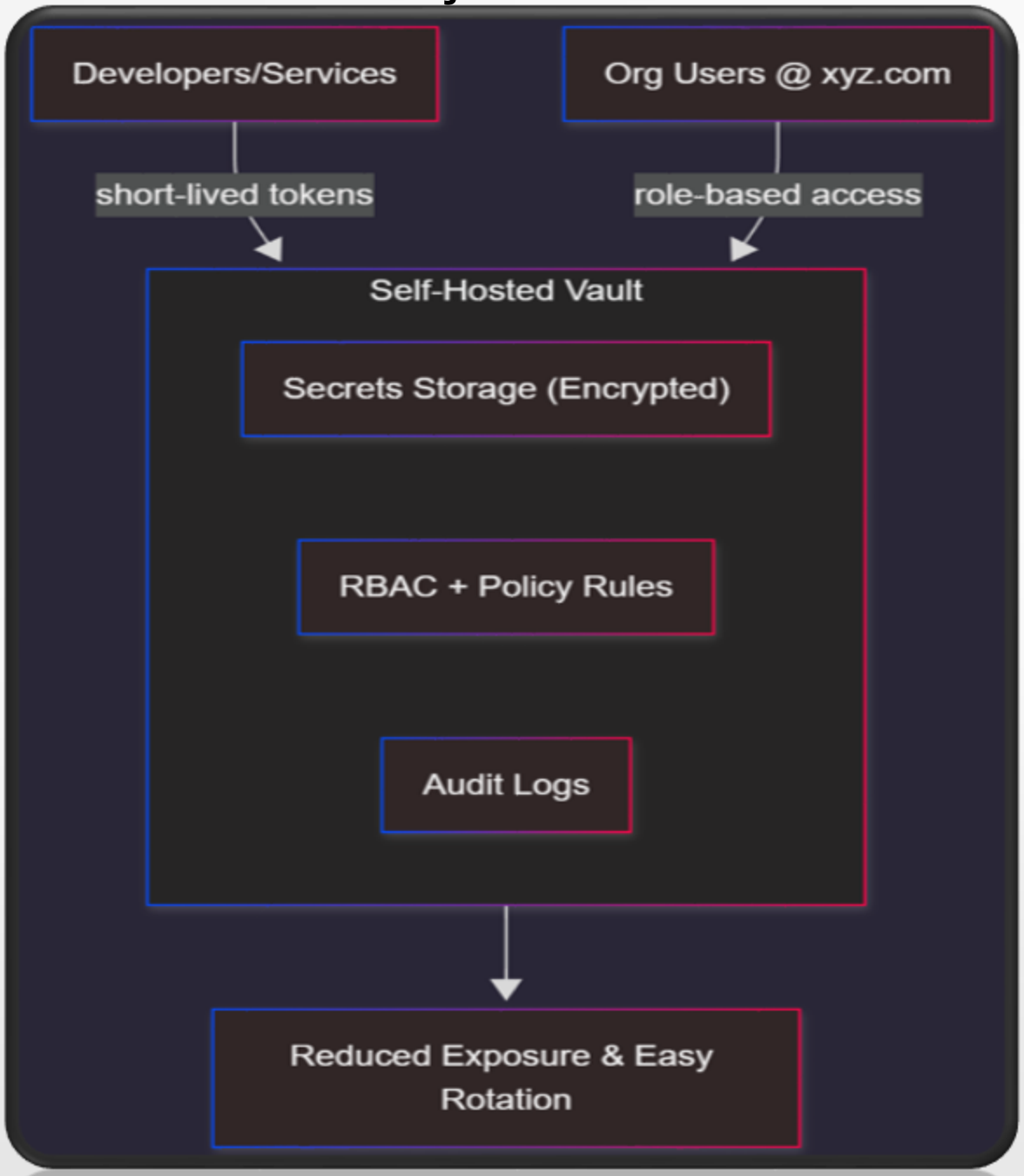
2.3 Access Control Requirements

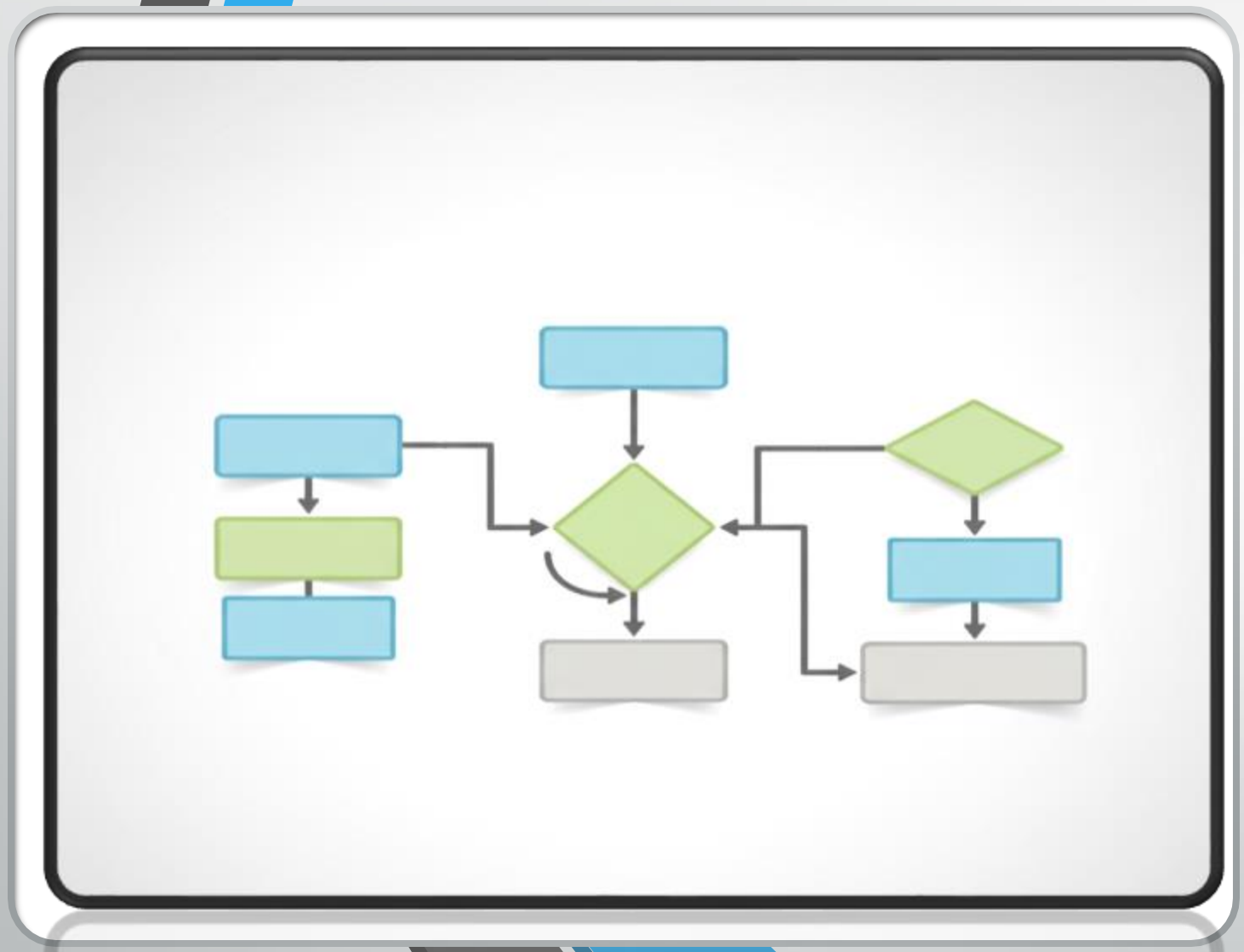
- **RBAC (Role-Based Access Control):** Map org roles (CEO/Director/AGM/Manager/Executive/Admin) to permissions (read/write/share/audit).
- **Rule/Policy-Based Control:** Fine-grained constraints (e.g., **who, which secret, from where (IP/device), when (time), how (api/ui)**).
- **Segregation of Duties:** Admins who manage access ≠ users who consume credentials.

2.4 Operational Requirements

- **Self-Hosted & Free:** Prefer open-source with zero licensing cost.
- **Low Maintenance:** Simple updates/backup; minimal infra overhead.
- **Security Essentials:** Audit logs, secret versioning/rotation, MFA/2FA support, encryption at rest & in transit.
- **Developer Safety:** CLI/API so secrets never sit in source code or long-lived env vars.
- **Multi-Environment:** Support dev/stage/prod namespaces.

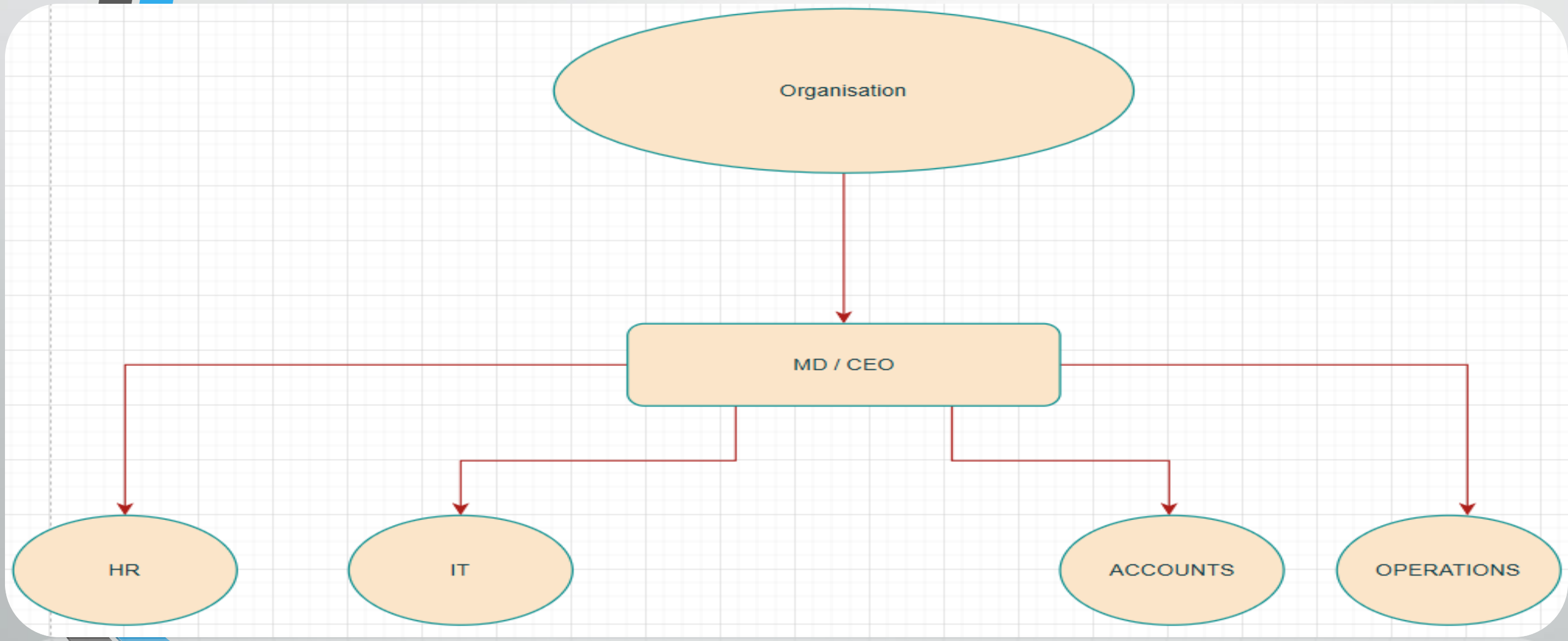
Visual – Objective at a Glance



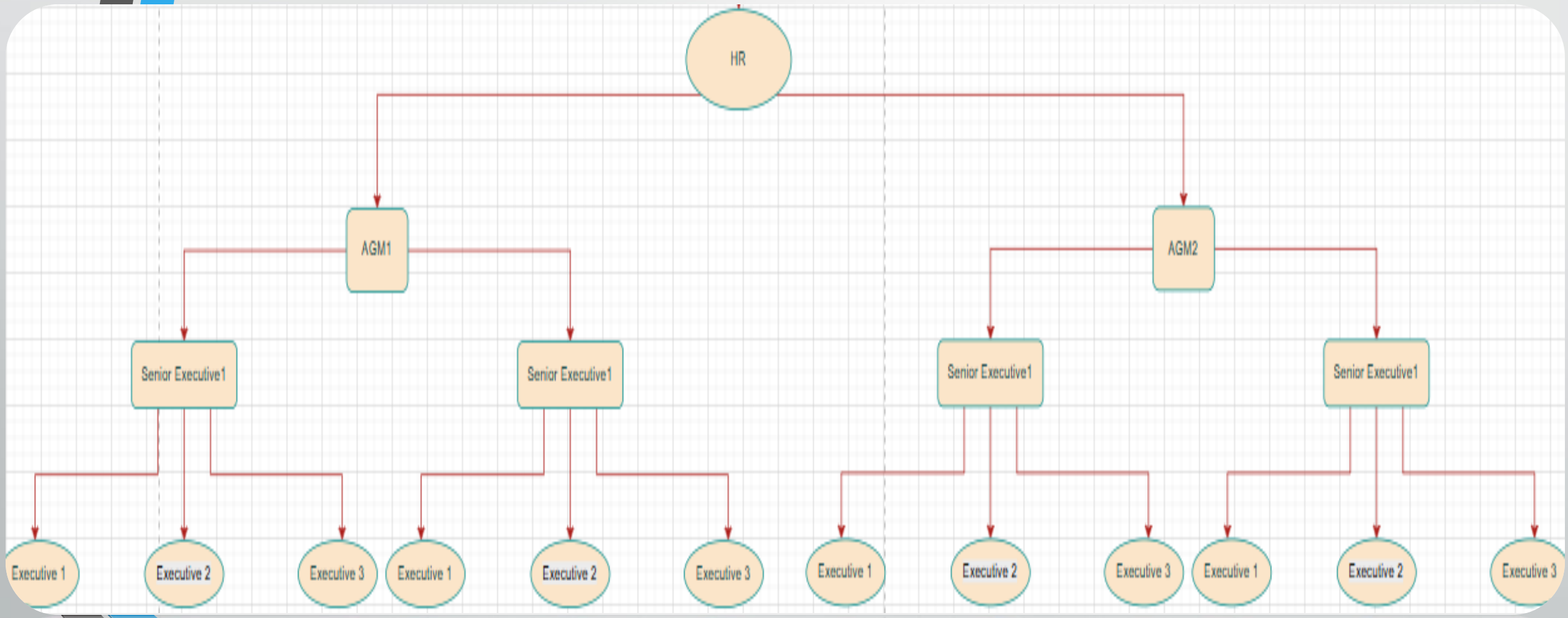


3. Structure

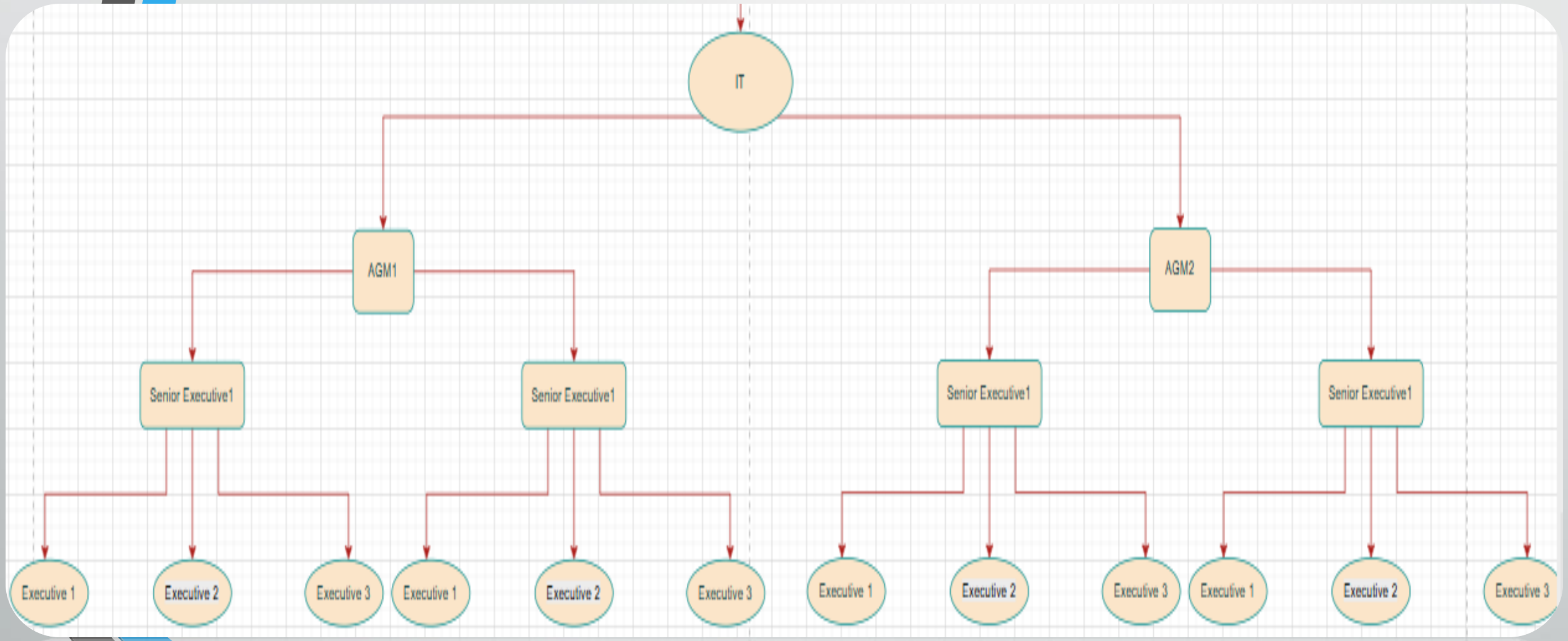
3.1 LEVEL 1 Structure



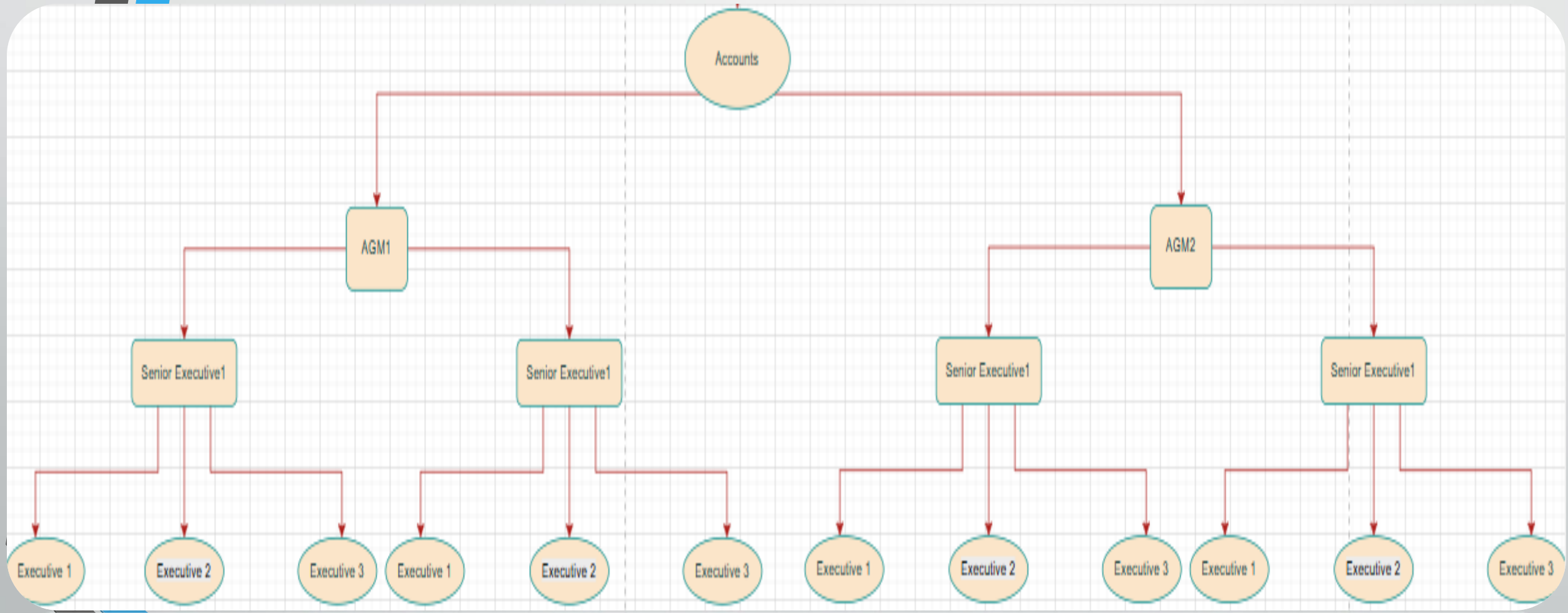
3.2 LEVEL 2 Structure



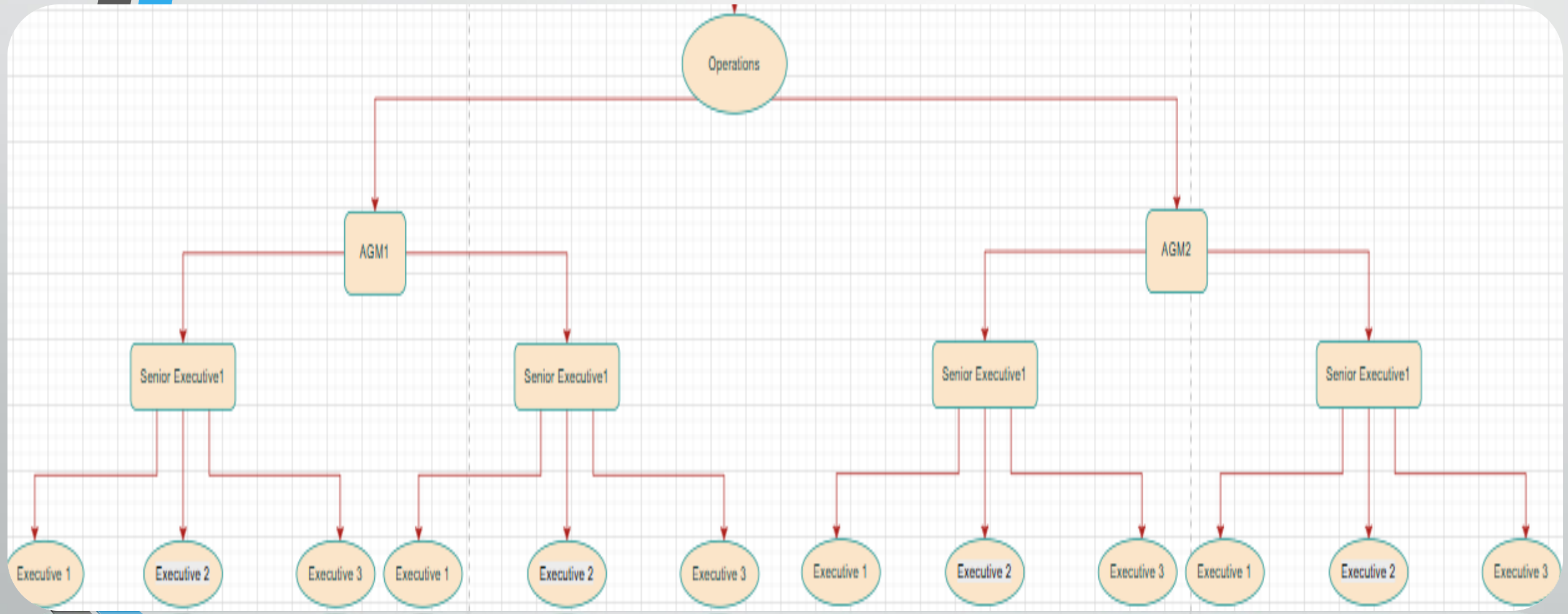
3.3 LEVEL 3 Structure



3.4 LEVEL 4 Structure



3.5 LEVEL 5 Structure



4. Maintenance

- This section explains how the Sampurna Credential Management System will be operated day to day.

4.1 User & Account Management
- Only Super Administrator can create, delete, update users.
- Bulk user operations supported.
- No self-registration allowed.
- Immediate deactivation on exit or role change.

4.2 First Login Setup
- Super Administrator provides initial credentials.
- User must set new password and enable multi-factor authentication at first login.

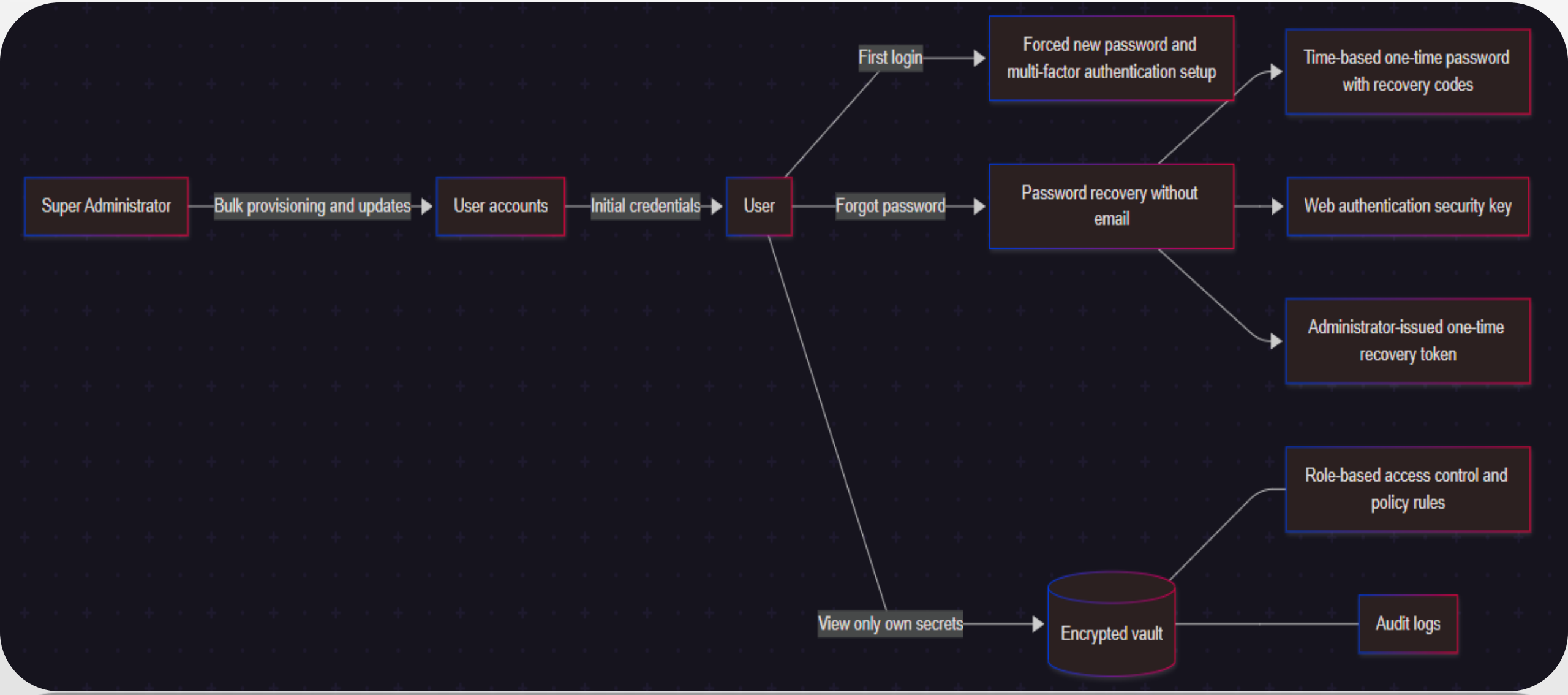
4.3 Password Reset (No Email)
- Reset via time-based one-time password or recovery codes.
- Option for hardware security key or admin-issued recovery token.

4.4 Access Visibility
- Users can view only their own secrets.
- Admins cannot see user secrets (zero-knowledge).
- Shared secrets controlled by role-based policies.

4.5 Audit & Compliance
- All actions logged (login, view, modify, reset).
- Logs tamper-proof and exportable.
- Regular access review and credential rotation.

4.6 Operational Maintenance
- Automated backups and updates.
- Backup includes config, encrypted data, and keys.
- Enforce security best practices (TLS, rate limits, policies).
- Separate environments for development, staging, and production.

4.7 Visual — Maintenance lifecycle



5. Tool Selection

Your SCMS requires a **self-hosted, free** solution with **strict admin-controlled accounts (no self-signup), first-login password change, MFA, policy/RBAC, audit logs, versioning**, and a **password-reset path that does not depend on SMTP**. It must model **organization → departments → groups → users**, keep secrets out of code, and support multiple environments (dev/stage/prod).

5.1 Chosen toolset:

- **Keycloak** (Identity & Access Management) → handles **user lifecycle**: admin-only user creation, temporary passwords, force password change, MFA, **no-SMTP "reset" via admin-set temp password or recovery codes**.
- **HashiCorp Vault (Community)** (Secrets platform) → handles **secrets**: RBAC/policies, versioning/rotation, audit, API/CLI, namespaces for **dev/stage/prod**, "no secrets in code" pattern.
- This split matches your **segregation of duties** and **policy-based control** goals: IAM in Keycloak; secrets and policies in Vault.

5.2 Why Keycloak + Vault matches SCMS—point by point

- **Self-hosted, open-source** → both tools are OSS and deploy easily with Docker (fits "Self-Hosted & Free, low-maintenance").
- **Admin-only accounts, no self-registration** → Keycloak lets **Super Admin** create users; you can disable self-registration. Matches your "Only Super Administrator can create users; No self-registration allowed."
- **First login setup** → Temporary password + **force update on first login**; enable **MFA**. Exactly your "User must set new password and enable multi-factor at first login."
- **Password reset without email** → Use **admin-initiated temp password** or **recovery codes**; you can run the full flow **without SMTP**.
- **RBAC and policy-based control** → Map Keycloak **groups/roles** to Vault **policies**; enforce who/what/where/when/how.
- **Segregation of duties** → IAM admins ≠ vault policy owners; users who view/use secrets ≠ people who manage access.
- **Zero secrets in code; short-lived retrieval by apps** → Apps authenticate, call Vault, fetch time-bounded secrets/tokens.
- **Environments** → Vault **namespaces** let you mirror dev/stage/prod as requested.
- **Audit, versioning, rotation** → Vault: audit devices.. Keycloak: admin events and login events. Matches your **audit + rotation** asks.

5.3 Differential comparison (what you asked to self-host)

Requirement (from SCMS)	Keycloak + Vault (Chosen)	Vaultwarden / Bitwarden	Passbolt CE	Infisical OSS	TeamPass / Psono (alts)
Self-hosted, free (OSS)	✅ **Both OSS. Low infra.**	✅ OSS (Vaultwarden)	✅ OSS	✅ OSS core	✅ OSS
Admin-only users; no self-signup	✅ **Admin creates users only.**	⚠️ Invite-based; usually email	✅ Admin creates	✅ Workspace admins; usually email	✅ Admin creates
First login: force password change + MFA	✅ **Temp password + forced update + MFA.**	⚠️ Force change needs admin/CLI dance; MFA ok	⚠️ Password change OK; MFA plugins	✅ MFA; forced update varies by flow	⚠️ Varies / plugins
Password reset w/o SMTP	✅ **Yes: admin temp password or recovery codes.**	❌ Usually email; no true "forgot" offline	❌ Email-centric reset	❌ Typically email	⚠️ Often email-centric
Zero-knowledge / admins can't see user secrets	✅ **Vault enforces policy; admins don't see user secrets.**	✅ End-to-end	✅ OpenPGP model	✅ Client-side crypto	⚠️ Depends on product
RBAC + policy rules (who/what/where/when/how)	✅ **Keycloak roles + Vault policies.**	✅ Roles/collections; less "where/when/how"	✅ Roles, fine-grained	✅ Good dev-centric RBAC	⚠️ Basic RBAC
Org → Dept → Groups → Users	✅ **Keycloak groups/realms map cleanly**	⚠️ Orgs/collections; depts via naming	✅ Teams/groups	✅ Projects/envs focus	⚠️ Manual structuring
Secrets for servers / APIs (no secrets in code)	✅ **Designed for this.**	⚠️ Possible via API, but user-first tool	⚠️ Team sharing focus	✅ Great for CI/CD	⚠️ Mixed
Versioning / rotation	✅ **Vault KV v2 + rotation.**	⚠️ Basic item history	⚠️ Limited	✅ Yes	⚠️ Varies
Audit logs (tamper-proof, export)	✅ **Vault audit + KC events.**	✅ Basic logs	✅ Logs	✅ Audit trail	⚠️ Varies
Multi-environment (dev/stage/prod)	✅ **Vault namespaces.**	⚠️ Collections workaround	⚠️ Manual	✅ Native	⚠️ Manual
End-user usability	⭐⭐⭐ **Good (Keycloak UI + optional Vault UI)**	⭐⭐⭐ good	⭐⭐ Clear team UI	⭐⭐ Dev-oriented UI	⭐ to ⭐⭐

5.4 Bottom line:

- **Vaultwarden/Bitwarden** shines for end-users—but fails your **"no-SMTP password reset"** and **admin-issued first login password** requirements.
- **Passbolt** is strong for team password sharing—but its reset and invite flows are **email-centric**.
- **Infisical** is superb for **developer/app secrets**—but **user lifecycle and email-free recovery** aren't its strength.
- **Keycloak + Vault** is the **only** combination that satisfies **all** our hard requirements simultaneously (including **no-SMTP reset** and **strict admin control**) while keeping "secrets-in-apps" safe and short-lived.

5.5 Features:

- **Two moving parts instead of one:** Use a single docker-compose, health checks, and a simple runbook (backup, update, rotate).
- **Learning curve:** We'll publish a 2-page "Day-1 Guide" for helpdesk (create user → temp password → force change → MFA), and a simple "How apps fetch secrets from Vault" sheet for developers.
- **No-email reset requires helpdesk availability:** Mitigate by issuing **recovery codes** during first login, per your "Password Reset (No Email)" section.

5.6 Deployment sketch (concise)

- **Keycloak**: Realm = Sampurna; Groups = Departments (HR/IT/Accounts…); Roles = Viewer/Editor/Admin; Password policy = temp password + force change + MFA; Self-registration **off**.
- **Vault**: Enable OIDC auth (Keycloak); Namespaces = dev/stage/prod; Policies mapped to Keycloak groups; Audit device to file/syslog; Rotation schedules.
- **App pattern**: App gets OIDC token → reads short-lived secret from Vault (never stored in code).

5.7 Operational Blueprint – Keycloak + HashiCorp Vault (SCMS Stack)

A) End-User Experience (What People See)

👥 Everyday Users (HR, Accounts, IT)

- Log in through **Keycloak Login Page** (with company logo).
- Use **temporary password → force change → enable MFA**.
- If locked out → **contact Helpdesk**, no email needed.
- Securely access internal portals — **no secret visible** to them.

👨‍💻 Developers / Power Users

- Same login flow, but may view **limited secrets** in Vault (if allowed).
- Apps fetch secrets from Vault — **no hardcoding passwords**.

👨‍💼 Helpdesk / Super Admin

- **Admin Console in Keycloak** for user lifecycle.
- **Vault dashboard** for policies, audits, rotations.
- No need for SMTP setup — fully admin-driven credential control.

B) Maintenance Time

⚙️ One-Time Setup (Project Start)

- 3.5–5 days total (one engineer).
- Includes Docker setup, policy mapping, testing, and training.

🔄 Steady-State Operations (Per Year)

- Helpdesk (L1): ~130 hrs/year.
- DevOps (L2): ~80 hrs/year.
- IT Manager (L3): 16 hrs/year.

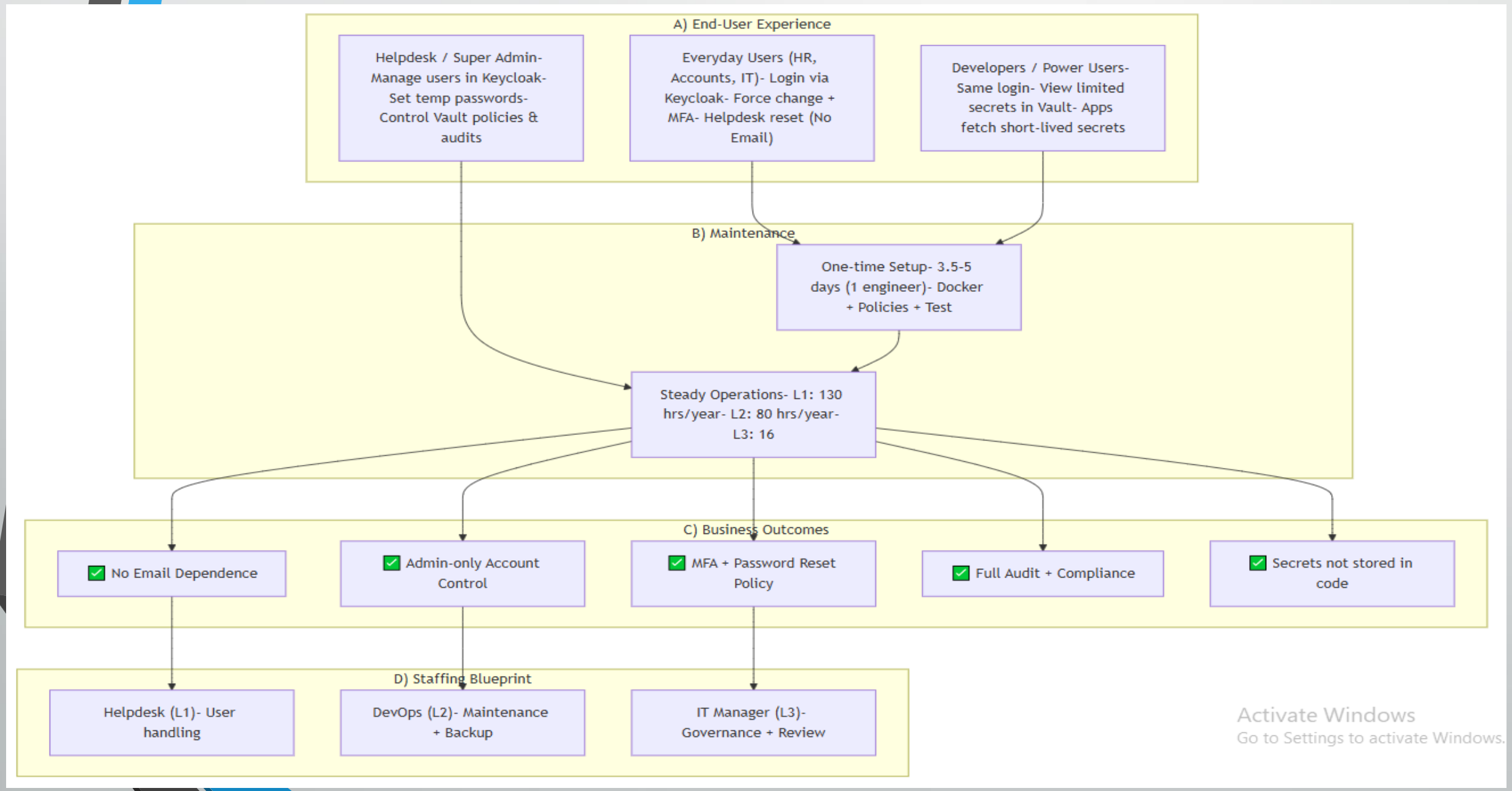
C) What This Buys You (Business Outcome)

✅ Zero dependency on email for password reset.
✅ No self-registration – total admin control.
✅ MFA + password change at first login.
✅ Least-privilege, policy-based access.
✅ Full audit trail for compliance.
✅ Secrets never exposed in code.
✅ Long-term cost efficiency.

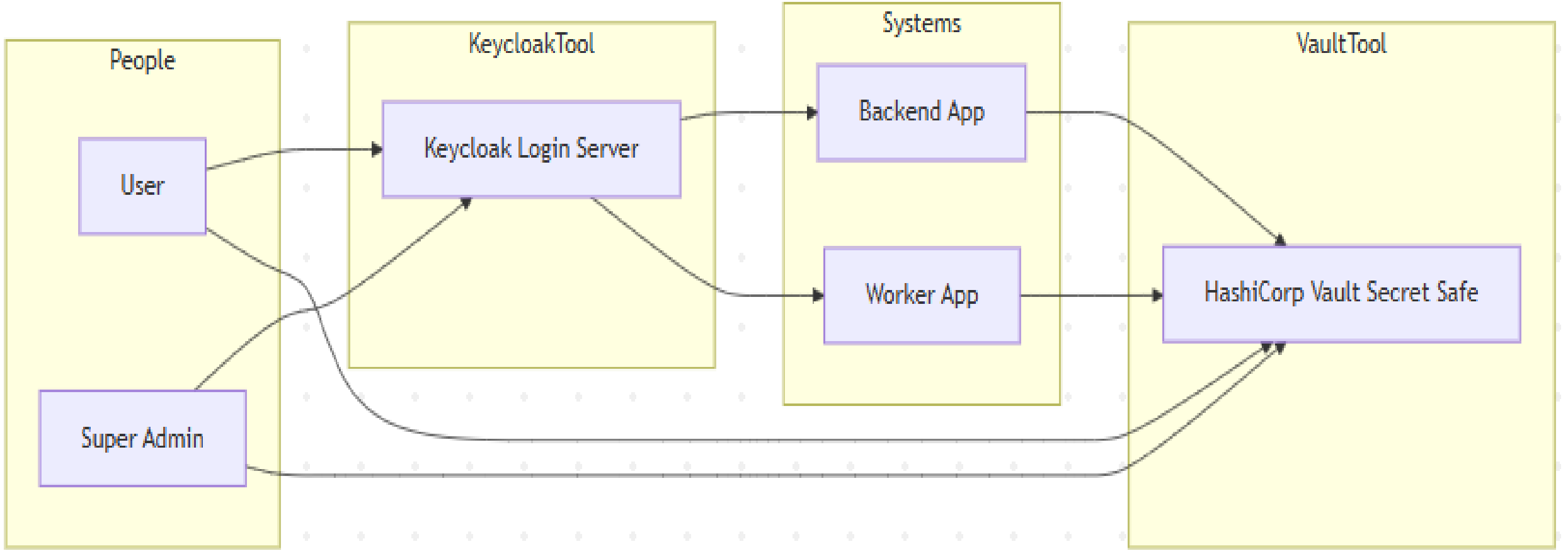
D) Quick Staffing Blueprint

Role	Responsibility	Frequency
Helpdesk (L1)	Create users, reset passwords	Daily / Weekly
DevOps (L2)	Maintain Keycloak + Vault, backups	Weekly / Monthly
IT Manager (L3)	Access reviews, governance	Half-yearly

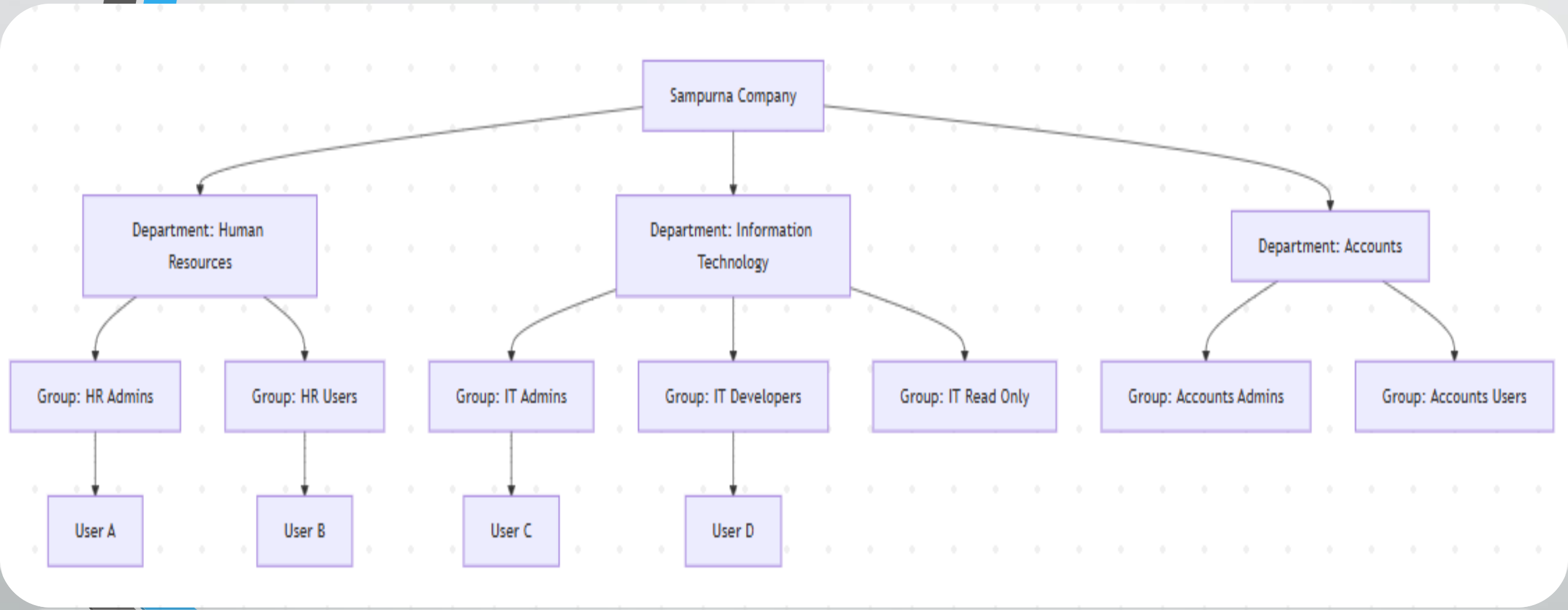
5.8 Mermaid Diagram (For the Operational Blueprint – Keycloak + HashiCorp Vault (SCMS Stack)



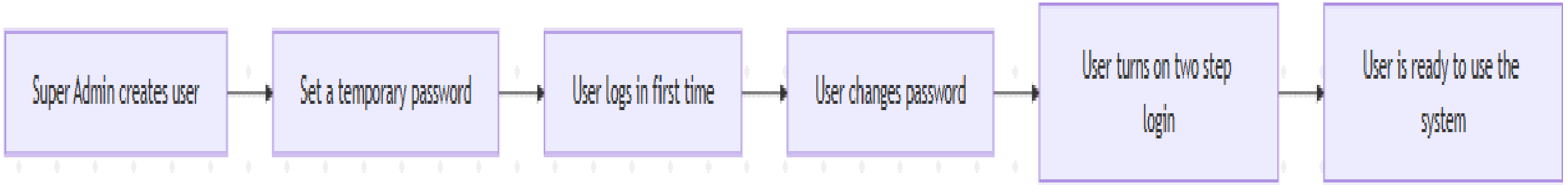
5.9 High-Level Architecture (SCMS)



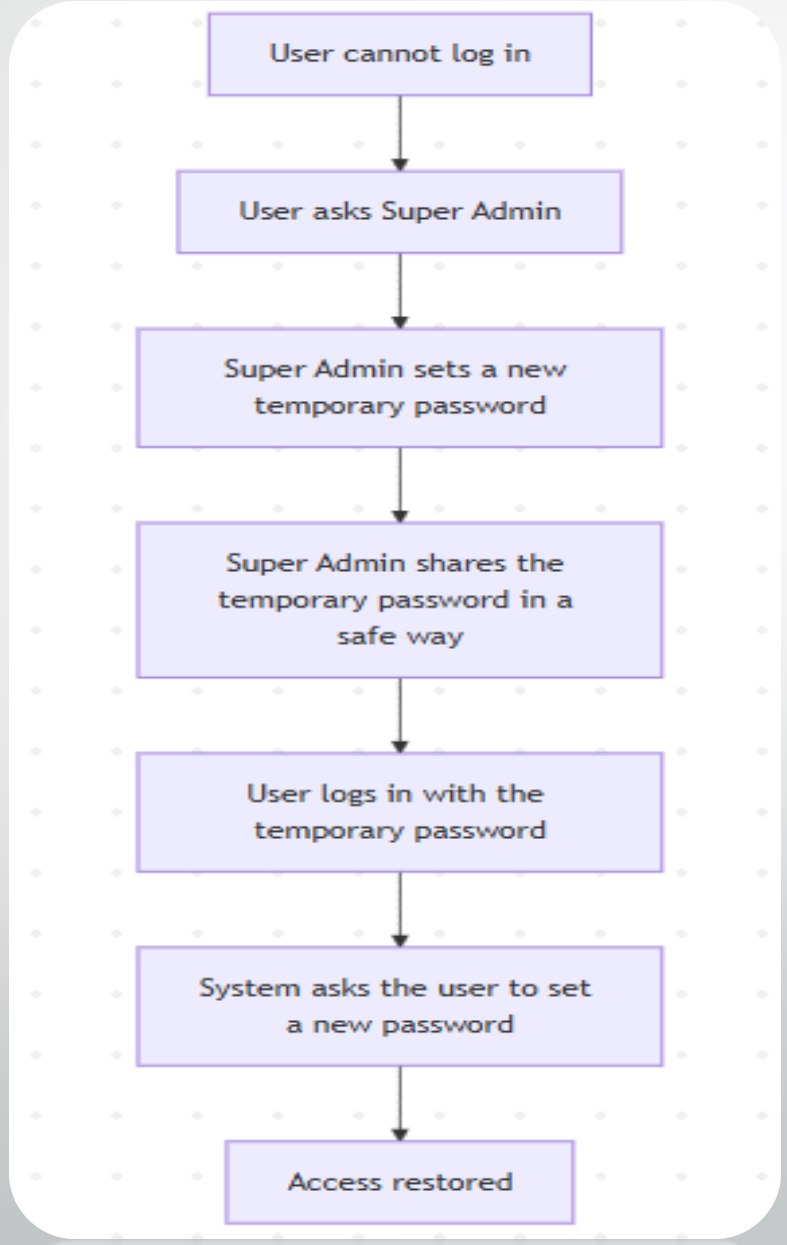
5.10 Organization → Departments → Groups (Structure)



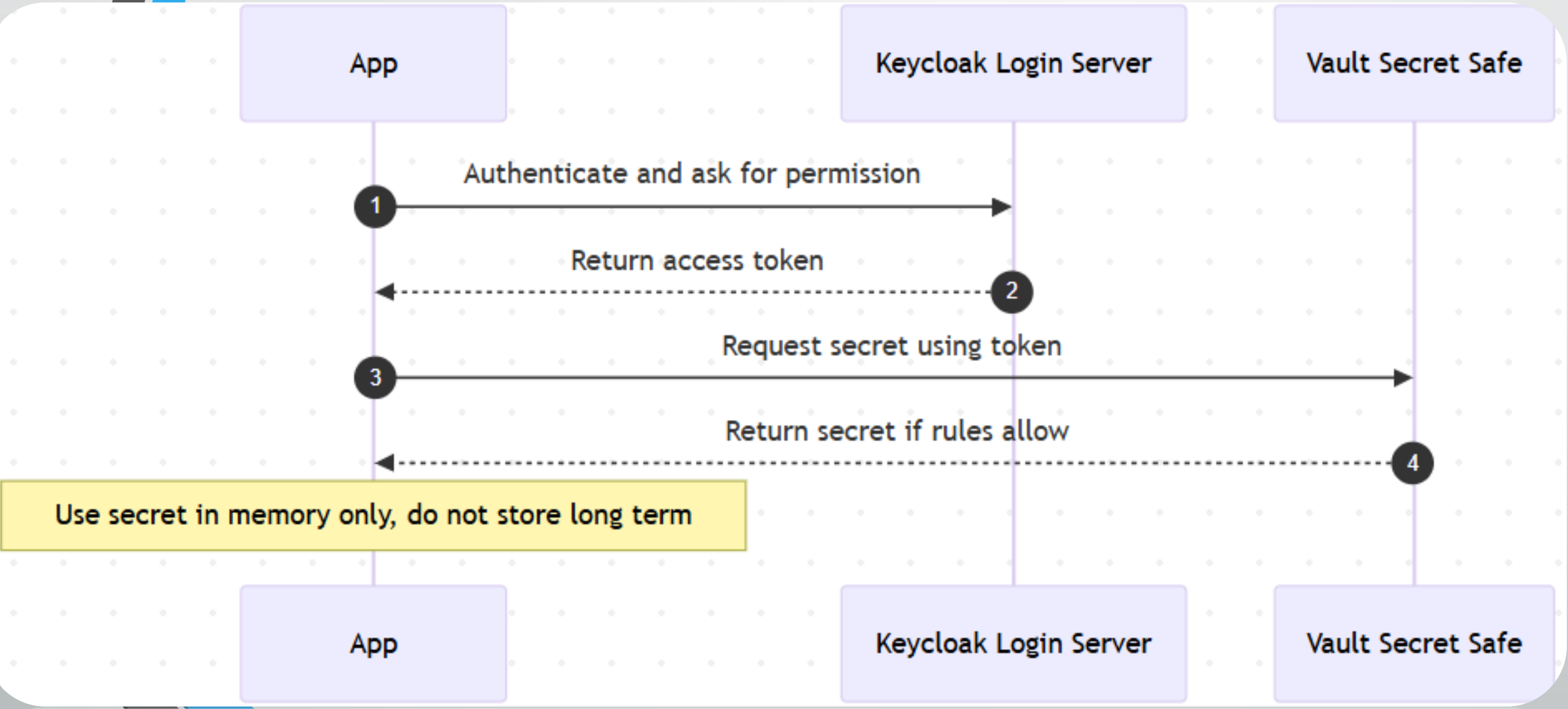
5.11 User Lifecycle (Admin-Created Accounts, First Login → Forced Change)



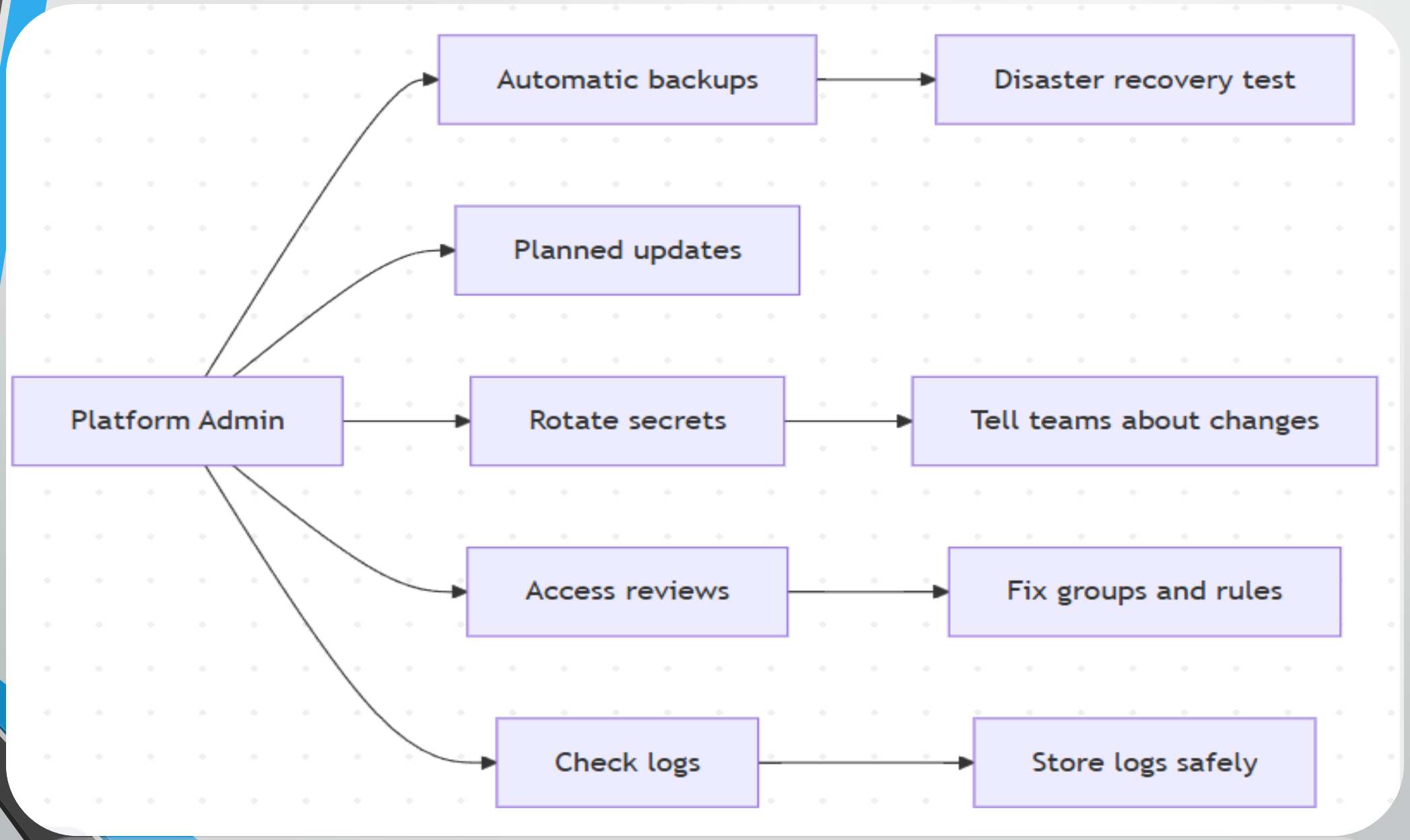
5.12 Password Reset WITHOUT SMTP (Admin Reset / Recovery Codes)



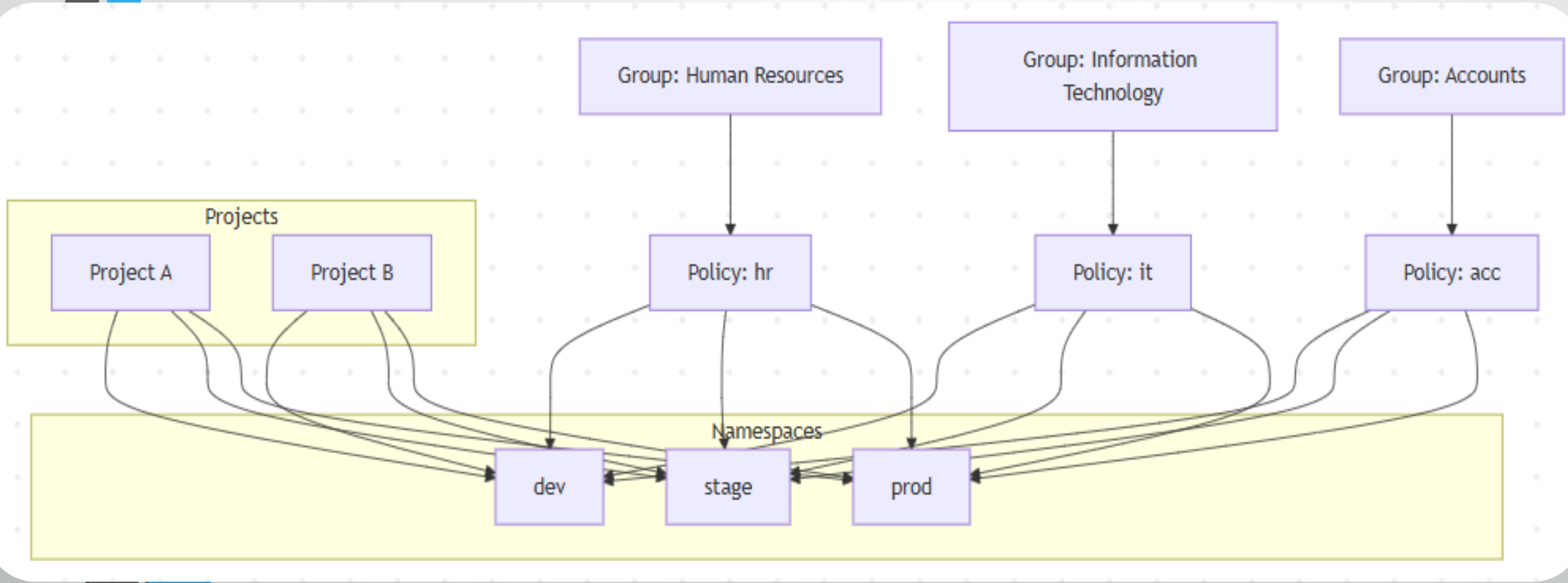
5.13 Secret Retrieval by a Service (Zero Secrets in Code)



5.14 Maintenance, Backup & Audit (Operational Runbook)



5.15 Environments and Collections (How projects map)



THANKYOU