SyncVault Tray App (Electron + GitHub + AWS Secrets Manager)

1. Purpose and Goals

1.1 What this tool is

SyncVault is a cross-platform desktop tray application (Windows and Linux first) that keeps configuration files synchronized across multiple computers, while keeping secrets out of the remote Git repository.

It is designed for the common workflow: - You have a project repository on one machine with a plaintext `.env` file (or similar config files). - You want the same file on another machine, kept in sync over time. - You want a readable Git history and line-by-line diffs for non-secret changes. - You want secret values (API keys, passwords, tokens) stored centrally and securely, not committed to Git.

SyncVault accomplishes this by: - Storing a *templated* version of each synced file (placeholders instead of secrets) in a private GitHub repository. - Storing the corresponding secret values in **AWS Secrets Manager**, using **one AWS secret per project**. - Rendering a local plaintext file by combining the template + secrets. - Monitoring for local changes and remote changes and keeping both sides synchronized.

1.2 Primary goals

1. **Cross-platform tray app** that stays running all the time.
2. **Multi-file sync** across many local project repositories.
3. **Intelligent grouping**: files belonging to the same local Git repo are grouped into one “project” and stored in one remote GitHub repo.
4. **No secrets in Git**: remote contains template files only.
5. **One AWS secret per project**: secrets are stored as a JSON object in AWS Secrets Manager.
6. **Bidirectional sync**:
7. Local changes update template and/or secret values.
8. Remote template changes pull down and regenerate local plaintext.
9. **Conflict-safe** behavior with clear resolution paths.
10. **Simple UX**: add files quickly (clipboard path or file drop), browse remote, pull down files and choose destinations.

1.3 Non-goals (initially)

- Not a full replacement for a secret manager UI.
- Not a general Git client.
- Not a cloud server product.
- Not real-time push notifications from GitHub (polling is acceptable).

---

2. Core Concepts

2.1 Project

A **Project** is defined as: - A local Git repository root containing one or more synced files. - A corresponding remote private GitHub repository that stores templates and metadata. - A corresponding AWS Secrets Manager secret containing the project's secret values.

Key: One local Git repo root == one SyncVault Project.

2.2 Template vs plaintext

- **Plaintext file**: The real file used by your project locally (e.g., `./.env`). It may contain secrets.
- **Template file**: A safe-to-sync version stored in GitHub, where selected secret values are replaced with placeholders.

Example: Plaintext:

```
DB_HOST=localhost
DB_PASSWORD=s3cr3t
STRIPE_KEY=sk_live_...
```

Template stored in GitHub:

```
DB_HOST=localhost
DB_PASSWORD={{SYNCVAULT:DB_PASSWORD}}
STRIPE_KEY={{SYNCVAULT:STRIPE_KEY}}
```

2.3 One AWS secret per project

Each project has exactly one AWS secret. - Name pattern (suggested): `syncvault/<github_owner>/<project_id>` - Stored value is JSON:

```
{
  "DB_PASSWORD": "s3cr3t",
  "STRIPE_KEY": "sk_live_..."
}
```

This improves cost/management and keeps the model simple.

2.4 Mapping metadata

SyncVault must know: - Which lines/keys in a file are “managed secrets”. - Which placeholder keys map to which JSON fields in the project secret.

This metadata lives in both: - The remote repo (so other machines can understand the template). - The local database (for fast lookup and UI state).

---

3. High-level Architecture

3.1 Components

1. **Electron main process**
2. Tray icon and menu
3. Background services: file watchers, sync scheduler, pollers
4. Credential access (OS keychain)
5. Git and AWS operations
6. **Electron renderer windows**
7. “Add file” wizard UI
8. “Remote browser / pull file” UI
9. “Project settings” UI
10. “Conflicts” UI
11. Logs and status dashboard
12. **Local data store**
13. SQLite database (recommended)
14. Stores projects, files, destinations, sync state, auth settings, conflict records
15. **Working directory**
16. A per-app “data” folder (AppData on Windows, ~/.local/share on Linux)
17. Contains clones of remote GitHub repos, and internal working copies
18. **External services**
19. GitHub: private repos for templates
20. AWS Secrets Manager: secret values per project

3.2 Technology choices

- Electron + Node.js
- File watching: `chokidar`
- Git operations: `git` CLI via `execa` (preferred) or `simple-git`
- GitHub API: `@octokit/rest` for repo creation and listing
- AWS SDK: AWS SDK for JavaScript v3 (`@aws-sdk/client-secrets-manager`)
- DB: `better-sqlite3` or `sqlite3`

---

4. User Experience and Flows

4.1 Tray menu (always available)

Tray icon context menu: - Status (Synced / Syncing / Error) - Add file from clipboard - Add file (browse) - Pull file from remote - Projects… - Conflicts… - Logs… - Pause syncing (toggle) - Quit

4.2 Add new file (clipboard)

Goal: Quickly add a file to be synced.

Steps: 1. User copies a file path to clipboard. 2. Tray menu -> “Add file from clipboard”. 3. Tool validates path exists and is a file. 4. Tool identifies project: - Run `git rev-parse --show-toplevel` in file directory. - If inside a repo, project root is returned. - If not inside a repo: treat as “Standalone Project” (optional) or reject initially. 5. Tool checks if this project already exists in SyncVault DB. - If not, create project record. 6. Tool ensures remote storage exists: - If the project has no remote GitHub repo, create one (private). - Clone the repo into working directory. 7. “Add file wizard” UI: - Show detected file content and parse result. - Allow user to select which keys/lines are secrets. - Let user define secret key names (default: env var key). - Confirm AWS region/account and secret name. 8. Tool writes: - Template file into remote repo clone. - Metadata mapping file. - Updates AWS Secrets Manager JSON for this project. 9. Tool commits and pushes changes. 10. Tool registers this file for ongoing watch + sync.

4.3 Pull file from remote

Goal: On another system, discover and retrieve a template-backed file, render plaintext, and save it to a chosen destination.

Steps: 1. Tray menu -> “Pull file from remote”. 2. UI shows list of remote projects (GitHub repos managed by SyncVault). 3. User selects a project. 4. UI shows browseable tree of template files. 5. User selects a file. 6. Tool pulls latest remote changes into local clone. 7. Tool renders plaintext version by: - Reading template. - Loading mapping. - Calling AWS Secrets Manager GetSecretValue for the project secret. - Replacing placeholders with real values. 8. Save dialog opens; user selects destination path (likely in the local project repo on that machine). 9. Tool writes file to that destination. 10. Tool registers the destination path for ongoing watch + sync.

4.4 Ongoing sync behavior

SyncVault runs continuously.

It maintains two kinds of monitoring: - **Local file change watch**: reacts quickly to local edits. - **Remote polling**: checks for remote template changes periodically (e.g., every 20 seconds).

On local change: - Re-template and update remote repo if non-secret lines changed. - Update AWS secret if a managed secret changed.

On remote change: - Pull updates. - Re-render plaintext files for each destination mapped to the template.

---

5. File Format Handling

5.1 Supported file types (MVP)

- `.env` style KEY=VALUE files

5.2 Parsing `.env`

Rules (practical subset): - Ignore blank lines. - Preserve comment lines beginning with `#`. - Preserve ordering. - Parse `KEY=VALUE` where KEY matches `[A-Za-z_][A-Za-z0-9_]*`. - Preserve raw VALUE including quotes.

Important: Many `.env` parsers treat quoting and escaping differently. For this tool, the safest approach is: - Preserve original formatting as much as possible. - Replace only the VALUE portion for selected keys.

5.3 Placeholder format

Use a placeholder that is unlikely to appear naturally. Recommended: - `{{SYNCVAULT:KEY_NAME}}`

Where `KEY_NAME` matches the secret JSON field in AWS.

5.4 Metadata mapping file

In the remote repo, store a mapping file next to the template. Example: `syncvault/mappings/<file_id>.json`

Example content:

```
{
  "fileId": "f_123...",
  "templatePath": "templates/myproj/.env.template",
  "type": "dotenv",
```

```
  "secrets": {
    "DB_PASSWORD": {"jsonKey": "DB_PASSWORD"},
    "STRIPE_KEY": {"jsonKey": "STRIPE_KEY"}
  }
}
```

The placeholders can be derived from mapping keys.

---

6. Remote Repository Layout

6.1 Repo contents

Each project remote repo contains: - `templates/` directory containing templated files - `syncvault/` directory for metadata - `README.md` describing purpose

Suggested layout:

```
templates/
  <relative-path-from-project-root>/
    .env.template
syncvault/
  project.json
  files/
    <fileId>.json
README.md
```

6.2 Naming remote repositories

Remote repo name must be unique and stable. Suggested: - `syncvault-<localRepoName>-<shortHash>`

Where shortHash is derived from: - absolute local repo root path - plus git remote origin URL if present

This prevents collisions across similarly named projects.

---

7. AWS Secrets Manager Integration

7.1 Secret naming convention

One secret per project. Suggested: - `syncvault/<githubOwner>/<projectId>`

Where projectId is stable across machines (based on GitHub repo name and/or a generated UUID stored in remote metadata).

7.2 Secret value structure

SecretString JSON object:

```
{
  "DB_PASSWORD": "...",
  "STRIPE_KEY": "..."
}
```

7.3 Read path

- On render, call `GetSecretValue`.
- Parse JSON.
- Replace placeholders.

7.4 Update path

When a secret value changes locally: - Fetch existing JSON. - Update keys. - `PutSecretValue` with new JSON.

Note: each `PutSecretValue` creates a new version. Avoid frequent updates by debouncing.

7.5 Credentials and auth

Support multiple auth options: - AWS SSO profile selection (recommended for developers) - Access keys stored in OS keychain

Store only references locally, not plaintext keys.

---

8. Sync Engine Design

8.1 Overview

Sync engine runs as a state machine per project and per file.

Two primary event sources: - Local watcher event - Remote poll event

8.2 Local watcher pipeline

For each tracked destination path: 1. Detect file change. 2. Debounce. 3. Determine file's template mapping. 4. Parse plaintext and compare with last rendered version. 5. For each managed secret key: - If value

changed, update AWS JSON. - Replace value with placeholder in template content. 6. For non-secret changes: - Update template content. 7. Write template into local clone. 8. Commit and push.

8.3 Remote poll pipeline

Per project: 1. `git fetch`. 2. Compare remote HEAD to last known. 3. If updated: - `git pull`. - For each template file changed: - Load mapping. - Render plaintext. - For each destination path mapped: - Write file (atomic write). - Mark this write as tool-originated to suppress watcher loop.

8.4 Debounce and loop prevention

When the tool writes a local destination file, it will trigger the local watcher. Prevent loops by: - Tracking “tool write timestamps” per destination path. - Ignoring watcher events for a short window (e.g., 800ms) after tool write.

8.5 Atomic writes

To prevent partial file reads by other processes: - Write to temp file then rename.

8.6 Scheduling

- Local events are immediate.
- Remote polling interval configurable (default 20s).

---

9. Conflict Detection and Resolution

9.1 When conflicts occur

Conflicts occur when: - Local plaintext changed since last sync AND - Remote template changed since last sync.

9.2 Conflict policy (MVP)

Safe default: - Do not overwrite. - Write copies: - `<file>.local` - `<file>.remote` - Notify user and mark file as “conflict state”.

9.3 Conflict resolution UI

Provide a simple UI: - Show file path - Buttons: - Keep local - Keep remote - Open diff tool

Future: built-in diff viewer.

---

10. Data Model (SQLite)

10.1 Tables

projects

- id (uuid)
- local_repo_root (text)
- display_name (text)
- github_owner (text)
- github_repo (text)
- github_clone_url (text)
- local_clone_path (text)
- aws_region (text)
- aws_secret_id (text)
- poll_interval_seconds (int)
- last_remote_head (text)
- created_at, updated_at

files

- id (uuid)
- project_id (uuid)
- source_relative_path (text) (path relative to local repo root)
- template_path (text) (path inside remote repo)
- mapping_path (text) (path inside remote repo)
- type (text) (dotenv)
- created_at, updated_at

destinations

A template may be rendered to multiple destinations (multiple machines, multiple local repos). - id (uuid) - file_id (uuid) - destination_path (text) - last_local_hash (text) - last_render_hash (text) - last_tool_write_at (int) - is_enabled (bool)

secret_keys

- id (uuid)
- project_id (uuid)
- key_name (text) (e.g., DB_PASSWORD)
- json_key (text) (usually same)
- created_at

conflicts

- id (uuid)
- destination_id (uuid)
- detected_at

- local_copy_path
- remote_copy_path
- status (open/resolved)

settings

- key
- value

---

11. GitHub Integration

11.1 Auth

MVP: GitHub fine-grained PAT. - Stored in OS keychain.

Later: OAuth device flow.

11.2 Repo creation

When a new project is added: - Create a new private repo via GitHub API. - Add README. - Clone locally.

11.3 Git operations

Prefer using git CLI: - `git clone` - `git fetch` - `git pull` - `git add/commit/push`

Reason: reduces edge cases and leverages user's Git setup.

---

12. Security Considerations

12.1 Secrets handling

- Never write secrets to remote repo.
- Never log secret values.
- Never store secret values in local DB.

12.2 Local plaintext

Local `.env` remains plaintext. - Users must still protect their workstation.

12.3 AWS permissions

Use least privilege: - Restrict to `syncvault/*` secret namespace. - Allow only required actions.

12.4 GitHub permissions

Use least privilege token: - Only repos created/managed by SyncVault.

12.5 Audit and rotation

- Support re-auth.
- Support secret rotation (updates create versions).

---

13. Packaging and Deployment

13.1 Windows

- Use electron-builder
- Auto-start option (registry run key)

13.2 Linux

- AppImage or deb/rpm
- Auto-start option (desktop autostart)

13.3 Updates

Future: - auto-updater (GitHub Releases)

---

14. Implementation Plan

Phase 1: Core sync engine (CLI prototype)

- Build Node CLI that:
- Initializes project
- Templates a file
- Stores secrets in AWS
- Pushes templates to GitHub
- Pulls templates and renders plaintext

Phase 2: Electron tray MVP

- Tray icon
- Add from clipboard
- Pull from remote
- Background watchers and poller
- SQLite persistence

Phase 3: UX polish and conflict UI

- Conflict detection + UI
- Logs window
- Settings window

Phase 4: Auth improvements

- GitHub OAuth device flow
- AWS profile selection UI

Phase 5: Multi-file and multi-destination enhancements

- Better grouping
- Better remote browser
- Better diff tooling

---

15. Open Questions (resolve during build)

1. Do we allow non-git files (standalone projects) in MVP?
2. How do we handle `.env` files with multiline values or export syntax?
3. Should templates be stored as `.env.template` or overwrite `.env` in remote?
4. How do we identify the same project across machines reliably?
5. Use remote project.json containing stable UUID.
6. What is the desired default behavior when secrets are missing in AWS?
7. Error + prompt user to supply.

---

16. Acceptance Criteria (MVP)

- Tray app runs on Windows and Linux.
- Add file from clipboard:
- Detect project grouping
- Create remote private repo
- Create/update AWS project secret JSON
- Push template and mapping
- Pull file from remote:
- Browse templates
- Fetch secrets from AWS
- Render plaintext
- Save to chosen path
- Ongoing sync:
- Local change updates template and/or AWS secret
- Remote change pulls and re-renders local plaintext
- Conflicts do not overwrite silently.

---

17. Appendix: Example project.json

Stored in `syncvault/project.json`:

```
{
  "projectId": "b3c4a9d6-...",
  "displayName": "MyProject",
  "createdAt": "2026-01-28T00:00:00Z",
  "aws": {
    "region": "us-east-1",
    "secretId": "syncvault/myuser/b3c4a9d6"
  },
  "templatesRoot": "templates"
}
```

18. Appendix: Example file mapping

`syncvault/files/<fileId>.json`:

```
{
  "fileId": "7b2f...",
  "type": "dotenv",
  "sourceRelativePath": ".env",
  "templatePath": "templates/.env.template",
  "managedKeys": ["DB_PASSWORD", "STRIPE_KEY"],
  "placeholderFormat": "{{SYNCVAULT:%KEY%}}"
}
```