Hochschule Worms
Fachbereich Informatik
Studiengang Angewandte Informatik - dual (M.Sc)

Projektbericht
Deep Dive

Entwicklung einer Lösung zur Berechtigungsverwaltung von Secrets zwischen Entwicklerinnen und Entwicklern

In der Arbeitsumgebung des Partnerunternehmens Medienagenten oHG

Version 1.0

Vorgelegt von

Leon Etienne, 676838
inf4437@hs-worms.de
Im Wintersemester 2024/25

bei Professor Dr. Heinemann
heinemann@hs-worms.de

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist noch nicht in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung vorgelegt worden.

Ludwigshafen am Rhein, 21. Februar 2025

Leon Etienne _____

Inhaltsverzeichnis

Abbildungsverzeichnis **I**

Tabellenverzeichnis **II**

Abkürzungsverzeichnis **III**

Glossar **IV**

1. Einleitung **1**

1.1. Problemstellung 1
1.2. Zielsetzung 2
1.3. Methodische Vorgehensweise 2

2. Grundlagen **3**

2.1. Die Arbeitsumgebung 3
2.2. 1Password 4
2.3. Ansible 4

3. Anforderungen **5**

3.1. Anforderungserfassung 5
3.2. Ergebnisse 5

4. Technische Umsetzung **7**

4.1. Berechtigungsverwaltung 7
4.1.1. Ausarbeitung der Herangehensweise 7
4.1.2. Kodierung 11
4.2. Integration in Ansible 15

5. Evaluation **18**

6. Fazit **19**

6.1. Ausblick 19
6.2. Offene Problemstellungen 19

Literaturverzeichnis **20**

Anhang **21**

A. Stakeholder-Interview **22**

B. Ideensammlung **23**

C. Relationsdiagramm (Überholt) **25**

Abbildungsverzeichnis

1. Relationsdiagramm: Bereitstellen von Projekten des Partnerunternehmens in einer Entwicklungsumgebung 3
2. Relationsdiagramm: Ansatz 1 | 1Password-API 8
3. Relationsdiagramm: Ansatz 2 | MASA 9
4. Relationsdiagramm: Ansatz 3 | Python-Toolbox 10
5. Struktur der Zugriffs-config.yml 12
6. Diagramm: Programmstruktur Secret-Synchronizer . . . 13
7. Relationsdiagramm: Docker-Ansible-Struktur, um 1Password (1P)-Einträge zu dereferenzieren 16
8. Ideensammlung 24
9. Relationsdiagramm: (Überholt) Relationsdiagramm . . . 26

Tabellenverzeichnis

1. Anforderungen 6

Abkürzungsverzeichnis

1P 1Password

API Application Programmer Interface

CLI Command Line Interface

GAU Größter Anzunehmender Unfall

GUI Graphical User Interface

MASA Medienagenten Secret Authority

YAML Yet Another Markup Language

UUID Universally Unique Identifier

1PSA 1Password Secrets Automation

Glossar

(1P-)Eintrag/Secret

Eine Gruppierung an Daten, die einen Login ermögliche. Z.B. (Nutzername, Passwort). Eine solche Struktur kann bein 1P aus beliebig vielen Schlüsselwertpaaren bestehen. Wird in dieser Ausarbeitung synonym zu 'Secret' verwendet.

(1P-)Vault

Eine Kollektion an Secret-Einträgen in einem Passwort-Manager (1Password).

Ansible-Playbook/s

Ansible-Playbooks sind Skripte, mit dem Ziel einen deklarierten Zustand herzustellen. [Red Hat, Inc., 2025]

Docker

Eine arrivierte Container-Engine für Anwendungsentwicklung.

Toolbox

Eine Ansammlung an Werkzeugen, wie zum Beispiel Skripte.

Öffentliche 1P-UUID

Die 1P-UUID, die zu einem originalen Eintrag gehört, und nicht zu einer Referenz. Externe Entwickler*innen haben also keinen direkten Zugriff auf einen solchen Eintrag, sondern müssen stattdessen eine Referenz auf diesen Eintrag verwenden. Eine solche UUID kann beispielsweise in Ansible-Konfigurationdateien stehen und von jedem*r Entwickler*in verwendet werden.

Private 1P-UUID

Die 1P-UUID, die zu einem Referenz-Eintrag gehört, und nicht zu einem originalen Eintrag. Diese Einträge sind emphemerisch und existieren ausschließlich in den einzelnen Developer-Vaults und sind Entwickler*innen-spezifisch.

Listenansicht / Listenaufruf

Eine Anfrage, die eine Liste von Objekten oder Einträgen zurückgibt.

Detailansicht / Detailaufruf

Eine Anfrage, *ein* Objekt oder *einen* Eintrag zurückgibt oder manipuliert.

1. Einleitung

1.1. Problemstellung

In der Arbeitsumgebung des Partnerunternehmens besteht zum Zeitpunkt der Themenfindung der hier beleuchteten Arbeit kein Management für Secrets und Logindaten zwischen Entwickler*innen. Logindaten zu den Projekten des Unternehmens liegen schlicht in einem 1P-Vault. 1P ist der vom Unternehmen verwendete Passwortmanager. Auf diesen Vault haben sämtliche internen Entwickler*innen Zugriff, jedoch keine externen Entwickler*innen. Das ist so, weil anderenfalls dLesezugriff auf sämtliche Einträge dieses Vaults gegeben werden müssten. 1P unterstützt keine Freigaben einzelner Einträge an andere Nutzer, ohne diese Einträge in einen eigenen Vault zu kopieren. Würden diese manuell in einen eigenen Vault kopiert werden, müssten diese Einträge fortan redundant gepflegt werden. Das ist eine Fehlerquelle, die zu asynchronen Einträgen führt. Außerdem ist das ein großer Arbeitsaufwand. All das gestaltet das Einbinden von externen Entwickler*innen, wie z.B. Freelancer*innen, schwer.

Ein weiteres Problem ist, dass Secrets in Konfigurationsdateien, die firmeninternen Ansible-Scripten beilegen, unverschlüsselt einsichtig sind. Das macht es zu einem großen Sicherheitsrisiko und somit inpraktikabel externen Entwickler*innen Zugriff auf dieses Ansible-Repository zu gewähren. Dieses Ansible-Repository ist jedoch zwingend erforderlich, um eine Entwicklungsungebung für Firmenprojekte auf dem lokalen Rechner zu schaffen. Auch hier sind Lösungen für externe Entwickler*innen zumeist unschöne Workarounds.

1.2. Zielsetzung

Ziel ist es, eine Umgebung zu schaffen, in der beliebigen Entwickler*innen bestimmte 1P-Einträge zugewiesen werden können. Der Pflegeaufwand sollte hierbei überschaubar bleiben. Das heisst, dass z.B. ganze Gruppen von Einträgen Entwickler*innen zugewiesen werden können. Wenn z.B. einem Projekt viele Einträge zugeordnet sind, sollten diese idealerweise mit einer einzigen Konfigurationszeile einem*r Entwickler*in zugeordnet werden können. Außerdem sollte eine Möglichkeit ausgearbeitet werden, um 1P-Einträge in Ansible auszulesen, damit keine Secrets mehr in den beiliegenden Konfigurationsdateien stehen, die das Freigeben dieser zu einem Sicherheitsproblem machen.

1.3. Methodische Vorgehensweise

Einige Anforderungen sind bereits im Voraus definiert. Weiterführende Anforderungen werden im Rahmen einer Anforderungserfassung ermittelt. Anschließend werden verschiedene Lösungsansätze betrachtet und auf Tauglichkeit geprüft. Nachdem ein akzeptabler Lösungsweg gefunden ist, wird dieser umgesetzt. Abschließend wird der Erfolg des Unterfanges evaluiert und mögliche, auf dieses Projekt aufbauende Arbeiten in Ausblick gestellt.

2. Grundlagen

2.1. Die Arbeitsumgebung

Die Arbeitsumgebung des Partnerunternehmens besteht für diese Themenstellug nennenswert aus:

- Cloudbasierten Web- und Datenbankservern
- Git-Repositories bei Bitbucket
- Der lokalen, Docker-basierten Arbeitsumgebung
- Ein Ansible-Playbook, das ein Projekt mit Daten aus der Cloudumgebung und Code aus Bitbucket in der lokalen Entwicklungsumgebung bereitstellt.

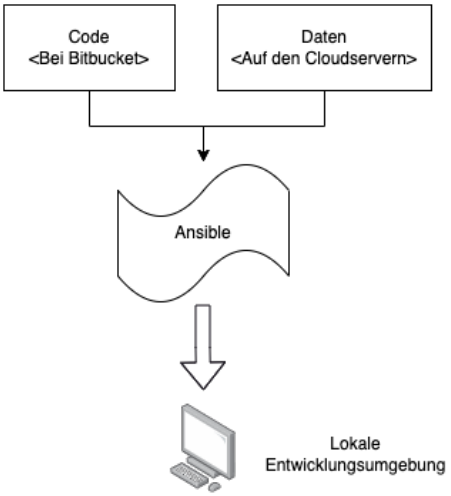


Abbildung 1.: Relationsdiagramm: Bereitstellen von Projekten des Partnerunternehmens in einer Entwicklungsumgebung

Quelle: Eigene Darstellung

Die lokalen Arbeitsumgebungen der Entwickler*innen liegen großteils außerhalb des Firmennetzwerkes, da diese Entwickler*innen oft oder

ausschließlich im mobilen- bzw, Homeoffice arbeiten. Ein Firmen-VPN-Netz existiert nicht und ist auch nicht erwünscht.

2.2. 1Password

1P ist der vom Partnerunternehmen verwendete Passwort-Manager. Bereits vor Beginn der Bearbeitung dieser Themenstellung wurde deutlich gemacht, dass es Ziel ist, 1P auch für das Verwalten von Secrets in Ansible zu verwenden.

2.3. Ansible

Ansible ist ein Automatisierungswerkzeug von Red Hat, Inc. und hat das Ziel, einen definierten Zustand im behandelten System herzustellen. [Red Hat, Inc., 2025] Ein Administrator definiert also nicht die erforderlichen Schritte, um einen Zustand z zu erreichen, sondern lediglich z selbst. Ansible kann über speziell gefertigte Python-Module um Schnittstellen erweitert werden.

3. Anforderungen

3.1. Anforderungserfassung

Obwohl bereits vor Beginn des Projektes einige Anforderungen bekannt sind, müssen manche Details nachträglich in Erfahrung gebracht werden. Hierfür wurde ein semistrukturiertes Interview mit dem Stakeholder durchgeführt. Im Rahmen dieses Interviews wurden vorbereitete Fragen gestellt, dem Stakeholder aber auch die Möglichkeit gegeben frei heraus zu sprechen und Wünsche zu äußern. Notizen zu diesem Interview befinden sich im Anhang unter *⟨⟨A Stakeholder-Interview⟩⟩*.

3.2. Ergebnisse

Das Ergenis der Anforderungserfassung ist ein Lastenheft, das in constraints, funktionale und nicht-funktioniale Anforderungen zu unterteilen ist.

Funktionale Anforderungen
Entwickler*innen erhalten verschiedene Zugang zu verschiedenen 1P-Einträgen (Zugänge), definiert in einer YAML-Datei.
Wildcard-Matching auf den 1P-Eintragstitel für zusammenhängende Einträge.
1P-Einträge sollen Entwickler*innen einzeln zuweisbar sein.
Nicht in der Konfugration gelistete Zugänge sollen bei Anwendung entfernt werden.
Ansible-Secrets müssen aus 1P dereferenziert werden können.
Einträge sollen für Entwickler*innen einsehbar sein.
Nicht-funktionale Anforderungen
Das System muss Berechtigungen von Entwickler*innen verwalten.
Das System muss benutzerfreundlich sein.
Das System darf nicht aufwändig zu pflegen sein.
Die benötigte Zeit zur Ausführung der Anwendung soll nicht sehr lange sein.
Das System muss robust gegenüber Misskonfigurationen sein, die zur Löschung der zugrunde liegenden 1P-Einträgen führen könnten.
Constraints
Nutzung von 1P ist zwingend erforderlich.
Die Übermittlung der Secrets muss über das Internet erfolgen.

Tabelle 1.: Anforderungen

4. Technische Umsetzung

4.1. Berechtigungsverwaltung

4.1.1. Ausarbeitung der Herangehensweise

Zunächst wurde gebrainstormed, welche Herangehensweisen hier möglich sind. Ein Artefakt des Brainstormings ist eine Mind-Map, die unter *⟨⟨B Ideensammlung⟩⟩* zu finden ist.

Ansatz 1

Der aus dieser Mindmap, nach individueller Meinung des Autors, vielversprechenste Ansatz ist es, die 1P-Restful-API zu verwenden. Bei diesem Ansatz würden Administrator*innen und Entwickler*innen API-Keys für 1P erhalten. Entwickler*innen hätten mit ihren Keys bestimmte Leseberechtigungen r und Administratoren die Berechtigung r zu verändern.

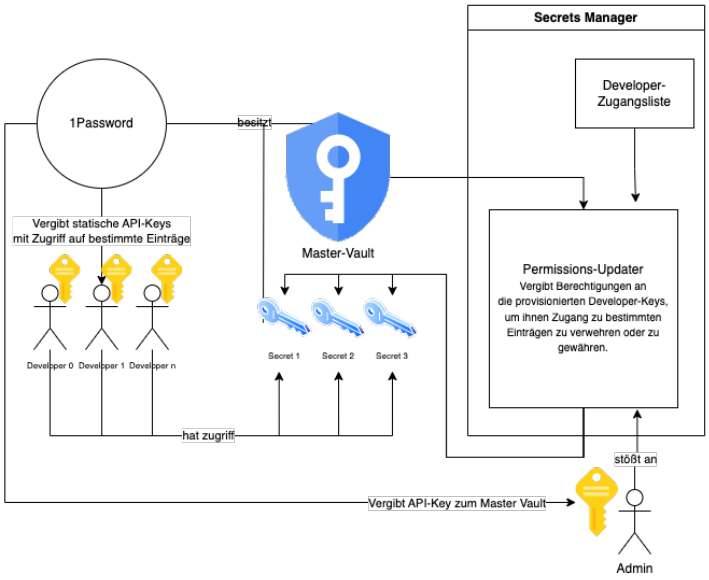


Abbildung 2.: Relationsdiagramm: Ansatz 1 | 1Password-API

Quelle: Eigene Darstellung

Dieser Ansatz wurde zeitnah als unumsetzbar erkannt und verworfen, da 1P das nachträgliche Verändern von API-Key-Berechtigungen nicht erlaubt.

Ansatz 2

Der nächste Lösungsansatz befasst sich mit einer Abstraktionsebene: Der Medienagenten Secret Authority (MASA). Hier ist die grundlegende Idee, dass es eine serverseitige Anwendung gibt, die sich MASA nennt. Diese Anwendung übernimmt die Aufgabe anhand eines hinterlegten 1P-API-Keys Secrets aus dem 1P-Vault des Partnerunternehmens abzufragen und an Entwickler*innen weiterzureichen. Die MASA provisioniert eigene API-Keys an Entwickler*innen und vermermerkt serverseitig, welcher API-Key berechtigt ist, welche 1P-Einträge abzufragen. Der API-Key könnte grundlegende Informationen wie zum Beispiel Entwickler*innennamen und Ablaufzeitpunkte des Keys einbetten. Dieser Ansatz trägt viel Sicherheitsverantwortung, da eine mögliche Ausnutzung

einer Sicherheitslücke der MASA direkt in den Firmen-Passwortmanager führen würden. Um diesem Risikofaktor entgegenzuwirken würde der 1P-Key der MASA verschlüsselt werden und die MASA würde nur einen Teil des Entschlüsselungs-Keys vorrätig halten. Der andere Teil wäre in jedem Entwickler*innen-Key eingebettet. Dadurch wäre gewährleistet, dass ein*e Angreifer*in, selbst bei sehr weitreichendem Zugriff in die MASA, nicht auf das Innere des Passwortmanagers zugreifen könne, da die MASA dazu selbstständig gar nicht im Stande wäre. Da Entwickler*innen lediglich ein Schlüsselfragment des Verschlüsselungs-Schlüssels in ihrem Key eingebettet hätten, der einen serverseitigen Schlüssel der MASA zum auslesen benötigt, bestünde auch keine Gefahr, dass ein*e Entwickler*in anhand seines bzw. ihres Keys ungeschützten Zugang zum Passwortmanager erhalten würde. Dieser Ansatz erlaubt für weitreichende Flexibilität, da sämtliche Logik, die sich mit Berechtigungen beschäftigt, anwendungsfallspezifisch geplant und umgesetzt wäre.

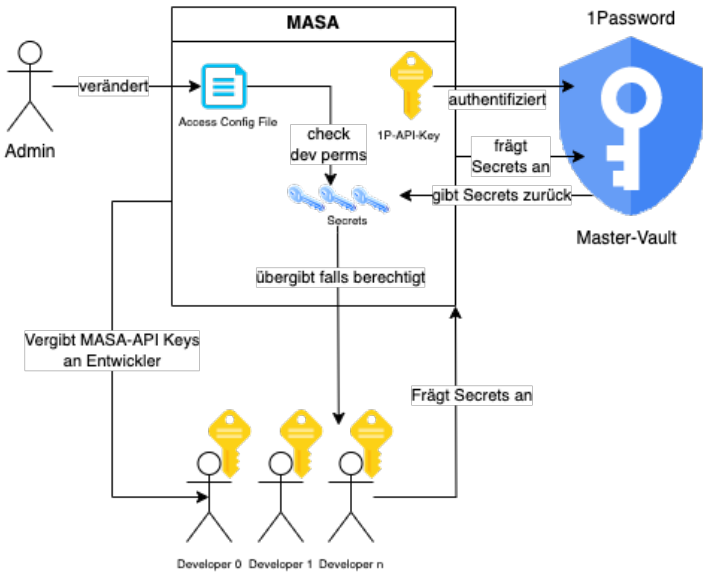


Abbildung 3.: Relationsdiagramm: Ansatz 2 | MASA

Quelle: Eigene Darstellung

Letztendlich entschied sich der Stakeholder gegen die Umsetzung der

MASA, da dieser Ansatz für zu Aufwändig betrachtet wird und für den durch sie erbrachten Vorteil zu viel Aufwand und Angriffsfläche schaffen würde.

Ansatz 3

Der letzte Lösungsansatz befasst sich mit dem Erstellen dedizierter Vaults für jede*n Entwickler*in e. Hierbei existiert eine Python-Toolbox, die anhand eine Yaml-Datei Referenzen auf diese Passwort-Einträge in Vault_e legt und von dort entfernt, wenn ein solcher Zugriff laut der Yaml-Datei nicht mehr vorgesehen ist. Diese Einträge können über feste Eintrags-IDs und über Regex bezogen auf die Eingrags-Titel einem/r Entwickler*in vorgesehen werden.

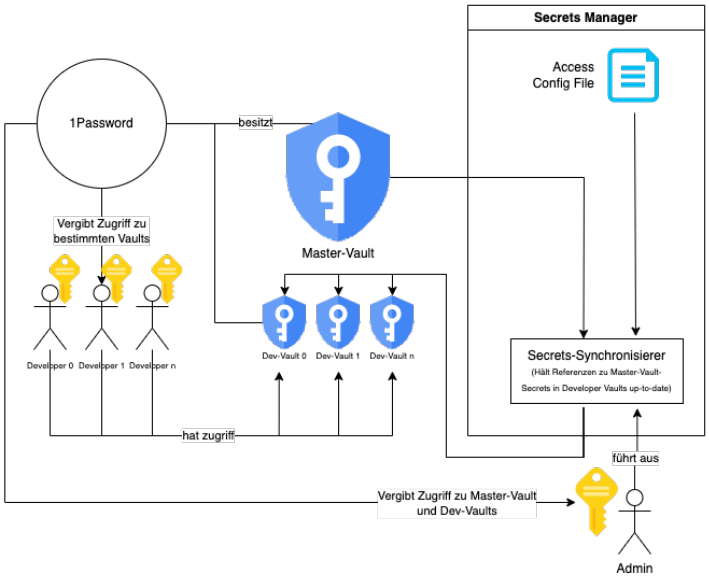


Abbildung 4.: Relationsdiagramm: Ansatz 3 | Python-Toolbox

Quelle: Eigene Darstellung

Letztendlich entschied sich der Stakeholder für Ansatz 3, da er ihm kostengünstig und aursreichend erschien.

4.1.2. Kodierung

Um den vom Stakeholder ausgewählten Ansatz 3 wie geplant umzusetzen, wurden zunächst die Dokumentationen diverser 1P-Schnittstellen konsultiert. Schnell offenbarte sich eine Alternative zu API-Keys: Die 1P-Desktop-Anwendung stellt eine CLI-API bereit. Die CLI-API der Desktop-Anwendung zu verwenden, würde drei Probleme lösen:

Kosten und hedonische Qualität

Einen API-Key zu erstellen und zu übermitteln ist kostenspielig und umständlich. Ein 1P-Konto haben dem gegenüber bereits alle Entwickler*innen des Partnerunternehmens.

Authentifizierung und Sicherheit

Anstatt einen API-Key unsicher zu speichern und in relevante Programme (=Ansible) zu laden, wird die Authentifizierung zu 1P ausgelagert.

Manuelle Einsicht

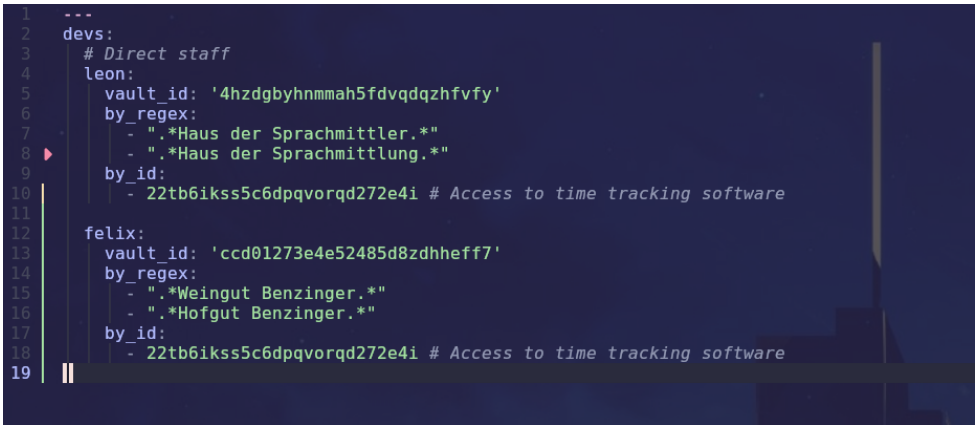
Da über die CLI-Methode der Zugriff auf die Entwickler*innen-Vaults direkt über die 1P-Desktop-App geschieht, kann diese den Vault-Inhalt auch dem Nutzer in ihrem GUI offenbaren.

Integrationsaufwand

Die Verwendung der 1P-Restful-API erscheint dem Autor nach ihrer Dokumentation sehr aufwändig und kompliziert. Eine CLI-API zu verwenden würde somit in der Umsetzung Projektressourcen sparen.

So begründet fällt die Wahl der Schnittstelle zu 1P auf ihre CLI-API. Um schnelle Softwareentwicklung mit minimalem Overhead zu gewährleisten und um für eine spätere Einbindung in Ansible bereits in Vorleistung zu treten, fällt die Wahl der Programmiersprache auf Python. Ansible-Module können mit Python geschrieben werden. [Red Hat, Inc., 2019] Es wurde eine rudimentäre Architektur entworfen, die beschreibt, welche Komponente des Werkzeuges aus welchen kleineren Komponenten

besteht. Am unteren Ende dieses Aggregatbaumes stehen atomare Operationen. Im Kontext dieses Werkzeuges sind atomare Operationen Operationen, die vom 1P-CLI ausgeführt werden. Diese Operationen implementiert also 1P selbst. Hierbei handelt es sich nur um Lese, Erstell- und Löschvorgänge. Das Erfassen, auf welchen Eintrag welche*r Entwickler*in Zugriff hat, und auf welche nicht, übernimmt *sync-dev-vault.py*. Die Funktionen der andere Skripte ergeben sich in Gänze aus ihren Dateinamen.



```
---
devs:
  # Direct staff
  leon:
    vault_id: '4hzdgbyhnmmah5fdvqdqzhfvfy'
    by_regex:
      - ".*Haus der Sprachmittler.*"
      - ".*Haus der Sprachmittlung.*"
    by_id:
      - 22tb6ikss5c6dpqvorqd272e4i # Access to time tracking software

  felix:
    vault_id: 'ccd01273e4e52485d8zdhheff7'
    by_regex:
      - ".*Weingut Benzinger.*"
      - ".*Hofgut Benzinger.*"
    by_id:
      - 22tb6ikss5c6dpqvorqd272e4i # Access to time tracking software
```

Abbildung 5.: Struktur der Zugriffs-config.yml

Quelle: Eigene Darstellung

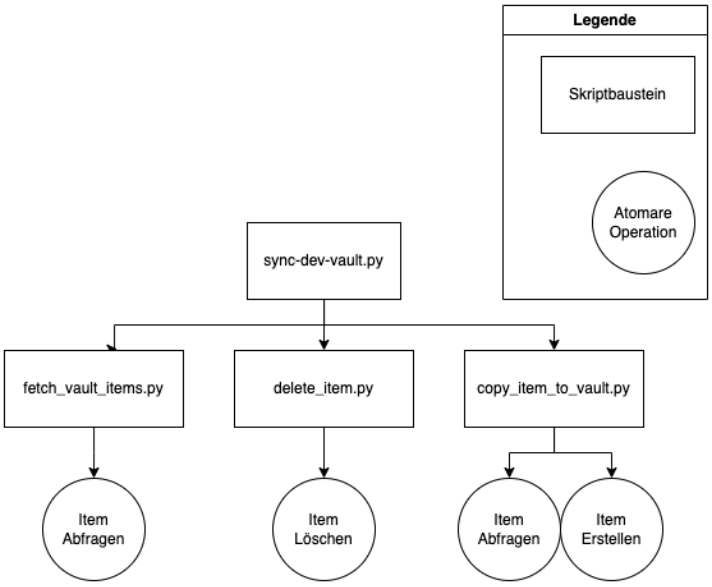


Abbildung 6.: Diagramm: Programmstruktur Secret-Synchronizer

Quelle: Eigene Darstellung

Die Funktionsweise des Programmes ist wie folgt:

1. Wende die Schritte 2..n auf alle zu synchronisierenden Entwickler*innen $d \in D$ an.
2. Lösche alle Einträge in $e_d \in d$.
3. Kopiere alle vorgesehenen Einträge $e_m \in$ Master-Vault nach d und ergänze das Feld “Originale Eintrags-ID” mit dem Universally Unique Identifier (UUID) von e_m.
4. Erstelle ein ID-Mapping-Objekt von e_d zu e_m in d.

Performanzprobleme und Optimierung

Eine Schwierigkeit, die sich im Rahmen der Umsetzung offenbart hat, ist, dass das 1P-CLI sehr langsam ist. Soll die Liste aller Einträge eines Vaults ausgelesen werden, dauert das nach Erfahrungen des Autors etwa

eine Sekunde, da es sich hierbei um *eine* Anfrage handelt. Diese Listenansicht zeigt jedoch nur begränzte Informationen der Einträge. Möchte man ein bestimmtes Feld eines Eintrages $z.B. [OriginaleEintrags-ID]$ in Erfahrug bringen, müssen alle Informationen eines Eintrages abgefragt werden. Hier ist ein CLI-API-Aufruf pro Eintrag erforderlich. Sind einem*r Entwickler*in z.B. 30 Einträge zugeordnet, so dauert das Finden eines Eintrags mit einer bestimmten originalen ID im Durchschnitt $n = 30; \frac{n}{2} = 15$ Sekunden. Im langsamsten Fall wären es $n = 30$ Sekunden. Ein Kopierforgang stellt zwei Aufrufe dar (=*lesen,erstellen*), also dauert das Kopieren von 30 Einträgen $n = 30; 2n = 60$ Sekunden. Das 1P-CLI kann zwar für Detail-Aufrufe mehrere Eintrags-IDs auf Standard-Input annehmen und bearbeiten, jedoch zeigen Versuche des Autors dies zu implementieren, dass das nicht die Zeitkomplexität von $O(n)$ verändert. Das heisst, eine Anfrage für 10 Einträge zu stellen, dauert in etwa zehn mal so lange, wie zehn Anfragen für jeweils einen Eintrag zu stellen.
Eine spätere Ergänzung, um die programmatische Auslesung der Einträge in $O(n)$ anstatt $O(n^2)$ zu gewährleisten, ist die Unterhaltung von Mapping-Objekten in Entwickler*innen-Vaults. Je Entwickler*innen-Vault wird abschließend der Synchronisierung ein Mapping-Objekt erstellt. Diese Mapping-Objekte halten die Informationen vorrätig, welche öffentliche 1P-UUID zu welcher privaten 1P-UUID gehört. Ohne diese Mapping-Objekte müssten für jeden Eintrag in Entwickler*innen-Vaults der nach einer öffentlichen UUID identifiziert wird (=Fremdschlüssel “Originale Eintrags-ID”), alle Einträge im Entwickler*innen-Vault abgefragt werden, bis ein Eintrag mit “Originaler Eintrag-ID = <x>” gefunden wird. Mit diesen Mapping-Objekten kann ein beliebiger Eintrag anhand einer öffentlichen ID binnen zwei Anfragen erfasst werden: Anfragen des Mapping-Objektes und Anfragen des privaten Objektes. Das entspricht einer Zeitkomplexität von $O(2n) = O(n)$.
Desweiteren kann 1P Eintragsdaten lokal zwischenspeichern. Diese Option lässt sich mit dem Flag *–cache* auf Leseoperationen verwenden und beschleunigt das Auslesen der angefragten Informationen, soweit die-

se im Cache existieren. Auf Unix-ähnlichen Betriebssystemen ist dieses Verhalten standardmäßig aktiviert. [1Password, 2025b]

Sicherheitsbedenken

Die Konfigurationsdatei definiert Zielvaults, nach ihren UUIDs. Anhand dieser IDs sieht ein*e Administrator*in keine Vaultnamen. Wenn nun aus etwaigen Gründen dort die ID eines Nutzvaults des Partnerunternehmems aufgeführt wäre, würde das Werkzeug alle sich dort befindlichen Zugänge löschen. Das wäre ein Super-GAU in Form von Datenverlust.

Sicherheitsvorkehrungen

Um das zu verhindern, wurde eine Liste mit wichtigen Vault-IDs fest einkodiert. Alle Erstell- oder Löschmethoden müssen einen Vault-ID-Parameter erhalten, selbst wenn dieser technisch nicht notwendig ist. Wenn diese Vault-ID nun in der Liste der fest kodierten Nutzvault-IDs vorkommt, meldet die Methode einen deskriptiven Fehler und beendet die Programmausführung. Somit ist gewährleistet, dass selbst bei einer fatalen Fehlkonfiguration kein Datenverlust entseht.

4.2. Integration in Ansible

Es ist Anforderung, dass 1P-Einträge von Entwickler*innen innerhalb von Ansible-Playbooks dereferenziert und verwendet werden können.
1P unterstützt nativ das Ersetzen von 1P-Referenzen in Dateien durch Secrets. Diese Technik nennt sich “1Password Secrets Automation (1PSA)”. Diese Technologie ist jedoch nicht für die hier vorliegende Aufgabenstellung verwertbar, da die dem zugrunde liegende Berechtigungsverwaltung auf Vault-Basis steht. Entweder hat ein*e Entwickler*in Zugriff auf einen gesamten Vault, oder er*sie keinen Zugriff auf den gesamten Vault. Eine feingranularere Steuerung ist hier nicht möglich, jedoch für die hier gegebenen Anforderungen nötig. [1Password, 2025a] 1PSA erfasst UUIDs und ist daher mit dem Konzept von Entwickler*innen-Vaults inkompatibel, da Entwickler*innen hierbei eigene Kopien der originalen Einträge

führen, die jeweils eigene UUIDs haben. Externe Entwickler*innen haben somit keinen Zugriff auf die originalen, öffentlichen UUIDs und die privaten UUIDs, die im Entwickler*innen-Vault vorhanden sind, gelten jeweils nur für ein*e Entwickler*in. Das erfordert eine maßgeschneiderte, programmatische Lösung:

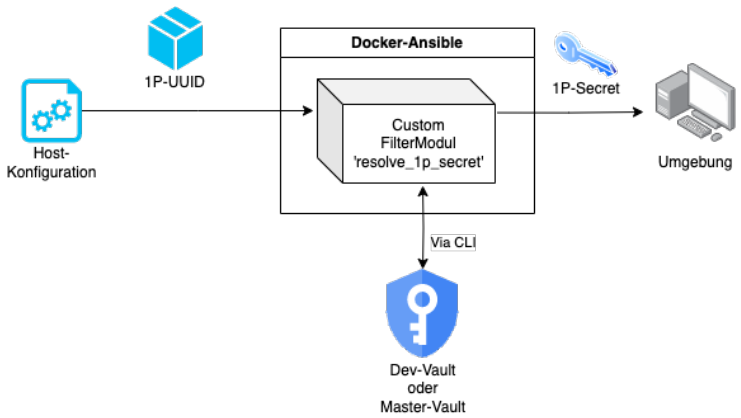


Abbildung 7.: Relationsdiagramm: Docker-Ansible-Struktur, um 1P-Einträge zu dereferenzieren

Quelle: Eigene Darstellung

Die hierfür angedachte Architektur erfordert, dass öffentliche UUIDs in Host-Konfigurationen in Docker-Ansible aufgeführt sind. Also eine UUID, die für alle Entwickler*innen greifbar ist.
Ab hier wird die Nutzergruppe “Entwickler*innen” in zwei Untergruppen strukturiert

Interne Entwickler*innen

Interne, festangestellte Entwickler*innen, haben Vollzugriff auf den 1P und somit auch Zugriff auf die notierte, öffentliche UUID eines Eintrages. Da diese Entwickler*innen keinen Entwickler*innen-Vault haben, müssen sie direkt auf die notierte, öffentliche UUID zugreifen.

Externe Entwickler*innen

Externe Entwickler*innen verfügen über einen Entwickler*innen-Vault, nicht jedoch über direkten Zugriff auf die vermerkte, öffent-

liche UUID. Falls der*die jeweilige Entwickler*in Zugriff auf einen verlinkten Eintrag hat, dann nur auf eine Kopie des Eintrages in dessen jeweiligen Entwickler*innen-Vaults. Diese hat eine andere UUID als die vermerkte. Die vermerkten, öffentlichen UUIDs müssen also zunächst in eine private, sich im Entwickler*innen-Vault befindliche, UUID übersetzt werden.

Um diese Problemstellung anzugehen, wird ein Ansible Filtermodul entworfen. Ein Filtermodul dient als Texttransformator und kann in Jinja-Templates, wie sie von Ansible verwendet werden, wie folgt verwendet werden:
`{{ "hello world" | uppercase }}`. Dieses Beispiel führt das “uppercase”-Filtermodul an. Ein Beispiel mit dem hierfür bereitgestellten Filtermodul würde so aussehen:
`{{ smtp.password | resolve_1p_secret }}`.

5. Evaluation

6. Fazit

6.1. Ausblick

6.2. Offene Problemstellungen

Literaturverzeichnis

[1Password, 2025a] 1Password (2025a). 1Password Secrets Automation . https://developer.1password.com/docs/secrets-automation/. Zugriff: Februar 2025.

[1Password, 2025b] 1Password (2025b). Cache item and vault information . https://developer.1password.com/docs/cli/reference/#cache-item-and-vault-information. Zugriff: Februar 2025.

[Red Hat, Inc., 2019] Red Hat, Inc. (2019). Ansible module development: getting started . https://cn-ansibledoc.readthedocs.io/zh-cn/latest/dev_guide/developing_modules_general.html. Zugriff: Januar 2025.

[Red Hat, Inc., 2025] Red Hat, Inc. (2025). Ansible Collaborative - What is Ansible? . https://www.redhat.com/en/ansible-collaborative. Zugriff: Januar 2025.

Anhang

A. Stakeholder-Interview

!!!TODO TODO TODO ADD APPENDIX INTERVIEW!!!

B. Ideensammlung

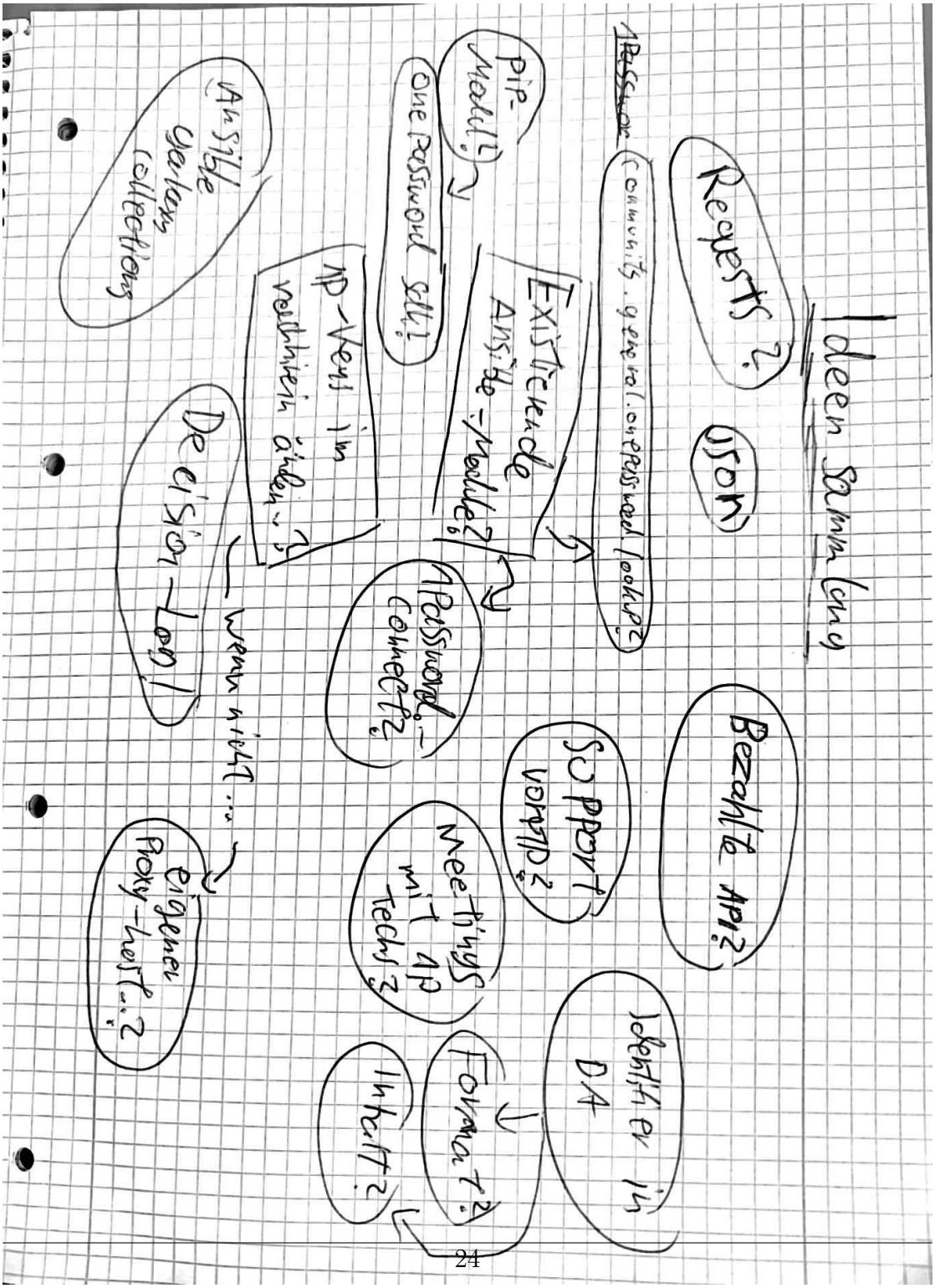


Abbildung 8.: Ideensammlung

Quelle: Eigene Darstellung

C. Relationsdiagramm (Überholt)

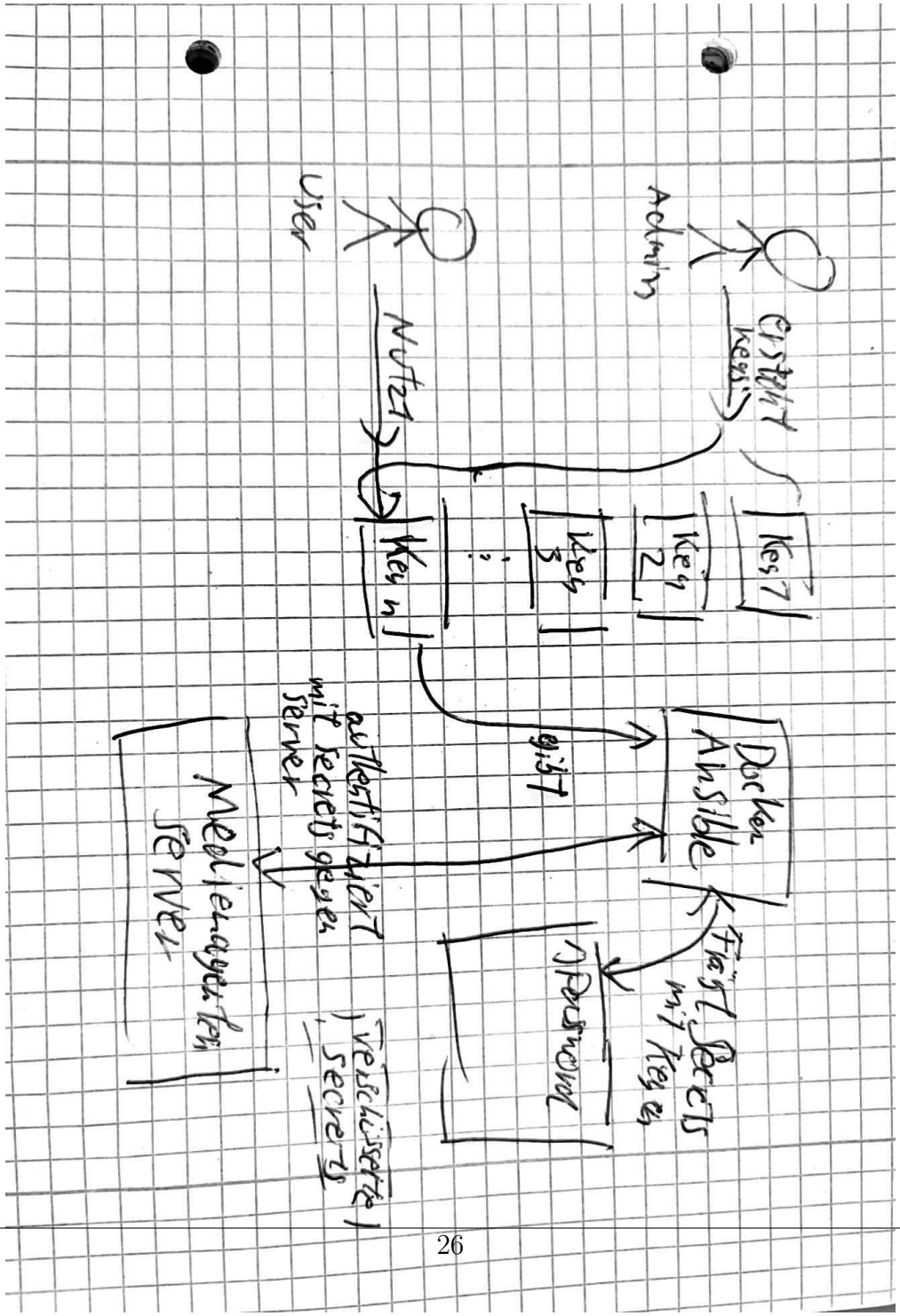


Abbildung 9.: Relationsdiagramm: (Überholt) Relationsdiagramm

Quelle: Eigene Darstellung