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

Tabellenverzeichnis

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

Glossar

(1P-)Eintrag/Secret

Eine Gruppierung an Daten, die einen Login ermögliche. Z.B. (Nutzername, Passwort). Eine solche Struktur kann bein 1Password (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.

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*innenn. 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 Configzeile 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.

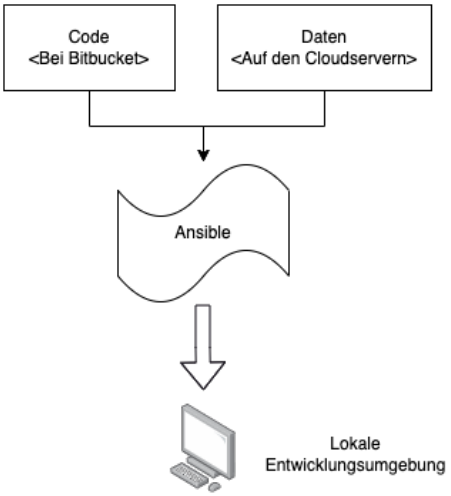


Abbildung 1.: Relationsdiagramm: Bereitstellen von Projekten des Partnerunternehmens in einer Entwicklungsumgebung

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 Zugänge, definiert in einer YAML-Datei.
Wildcard-Matching auf den 1P-Eintragstitel für zusammenhängende Einträge.
1P-Einträge sollen einzeln zuweisbar sein.
Nicht im YAML gelistete Zugänge sollen bei Anwendung entfernt werden.
Ansible Secrets müssen aus 1P dereferenziert werden können.
Einträge sollen auch manuell 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.

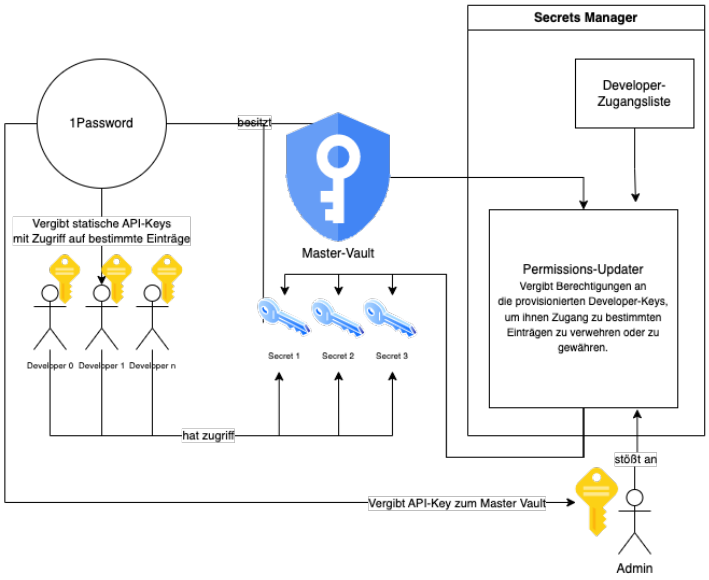


Abbildung 2.: Relationsdiagramm: Ansatz 1 | 1Password-API

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.

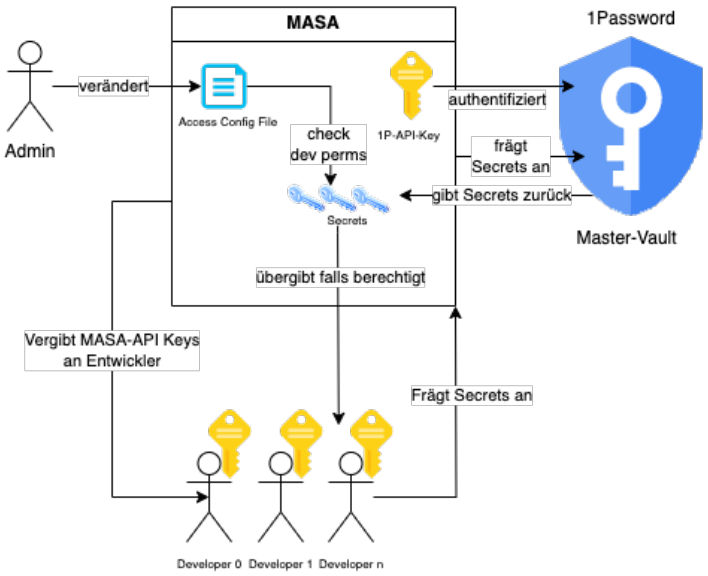


Abbildung 3.: Relationsdiagramm: Ansatz 2 | MASA

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.

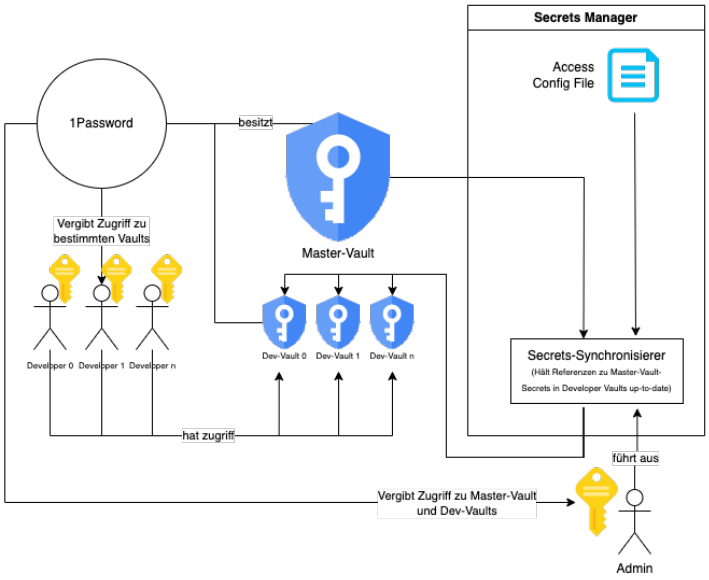


Abbildung 4.: Relationsdiagramm: Ansatz 2 | Python-Toolbox

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

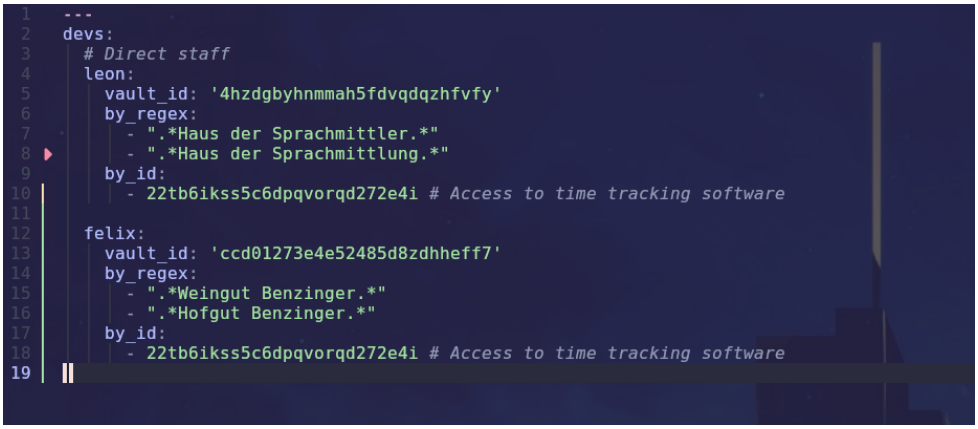
Da über die CLI-Methode der Zugriff auf die Entwickler*innen-Vaults direkt über die 1P-Desktop-App geschieht, kann diese den Vault-Inhalt auch dem Nutzer in ihrem GUI offenbaren.

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

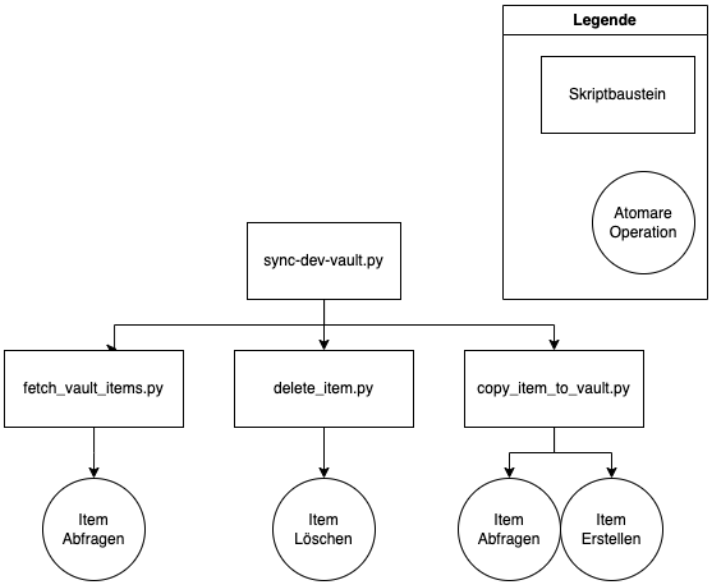


Abbildung 6.: Diagramm: Programmstruktur Secret-Synchronizer

Quelle: Eigene Darstellung

Sicherheitsbedenken

Die Konfigurationsdatei definiert Zielvaults, nach ihren kryptsichen IDs. 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

5. Evaluation

6. Fazit

6.1. Ausblick

6.2. Offene Problemstellungen

Literaturverzeichnis

[Maral et al., 1991] Maral, G., de Ridder, J.-J., Evans, B. G., und Richharia, M. (1991). Low earth orbit satellite systems for communications. *International Journal of Satellite Communications*, 9(4):209–225.

[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.

[TYPO3 Association, 2024] TYPO3 Association (2024). TYPO3 — the Professional, Flexible Content Management System . `https://typo3.org/`. Zugriff: Mai 2024.

Anhang

A. Stakeholder-Interview

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

B. Ideensammlung

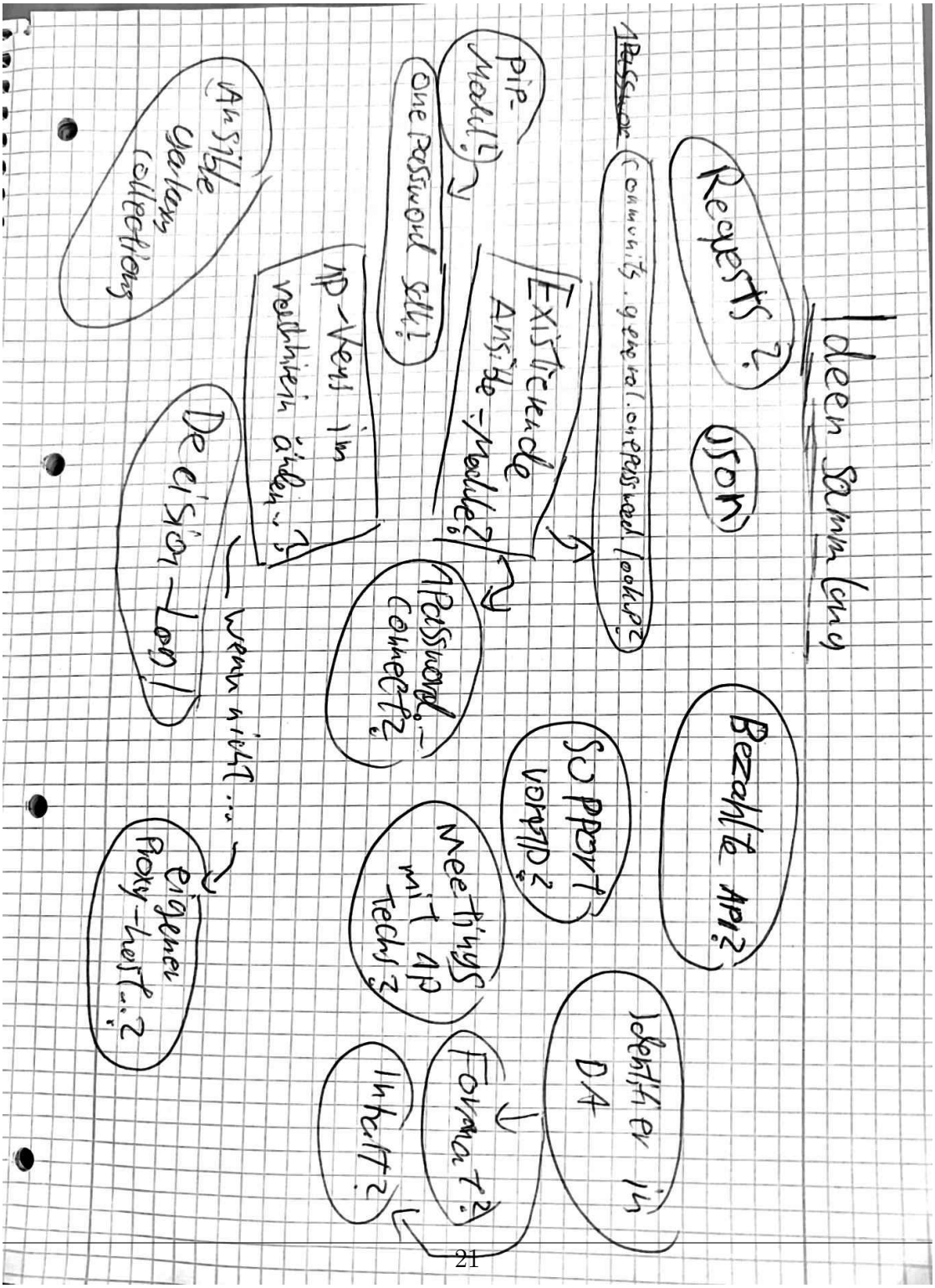


Abbildung 7.: Ideensammlung

Quelle: Eigene Darstellung

C. Relationsdiagramm (Überholt)

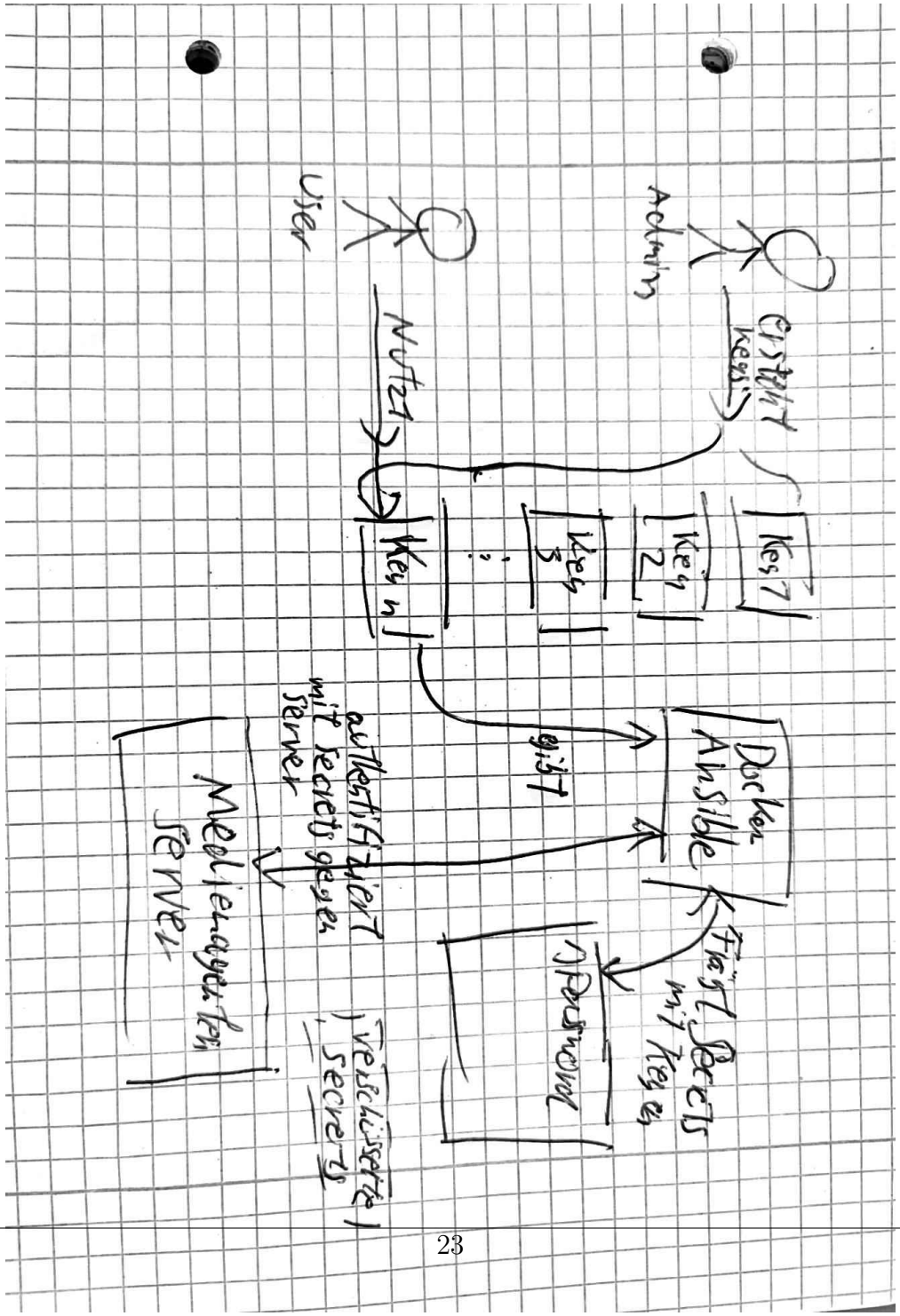


Abbildung 8.: Relationsdiagramm: (Überholt) Relationsdiagramm

Quelle: Eigene Darstellung