Kraken 2.0

Datenaggregation in der Netzwerkautomatisierung

Bachelorarbeit

Studiengang Informatik

OST – Ostschweizer Fachhochschule
Capmus Rapperswil-Jona

Herbstsemester 2021

Autoren:	Felix Kubli, Daniel Steudler
Betreuer:	Prof. Beat Stettler
Experte:	Dr. Peter Heinzmann
Gegenleser:	Prof. Oliver Augenstein

14. Januar 2022

Abstract

Einleitung

Bei der Netzwerkautomation sind die benötigten Informationen oft verstreut, so dass der Netzwerkengineer meist die einzige Person ist, welche genau weiss, was wo zu finden ist. Änderungen im Netz können deshalb zeitaufwändig werden und bei redundanten Daten in verschiedenen Quellen zu inkonsistenten Werten führen.

Deshalb besteht der Wunsch nach einer Single Source of Truth (SSoT). „Kraken“ als SSoT soll also Daten aus verschiedenen Quellen zusammenziehen, die Produkte der Firmen aber nicht ablösen, sondern diese um neue Funktionen erweitern, damit so u.a. auch die Herstellerunabhängigkeit erreicht werden kann.

Ziel

Anhand von Interviews mit verschiedenen Firmen soll ermittelt werden, welche Umsysteme vertreten sind und wofür sie verwendet werden. Wie gehen die Firmen mit ihren Daten und der Automation um?

Es besteht eine Vorgängerarbeit namens „Kraken“, die dieses Problem angeht. In dieser Arbeit soll nun die Datenstruktur flexibler implementiert werden und mehr Informationen abdecken. Die Datenstruktur soll nun auch Meta-Informationen wie Rack, Location und IP-Segmentation unterstützen. Konflikte bei Attribut-Werten sollen erkannt und aufgelöst werden.

Ergebnis

Durch die Interviews und den Fokus auf die Workflows der analysierten Firmen, konnten deren involvierte Umsysteme ermittelt werden. Die Interviews ermöglichten einen Einblick darin, was deren Workflows und wo deren Pain-Points im Hinblick auf die Automation sind. Leider haben uns die Interviews keine Workflows geliefert, welche direkt für den Test von „Kraken 2.0“ hätten verwendet werden können.

Für das Mergen von Daten wurden mehrere Konzepte in Betracht gezogen und ein auf Ähnlichkeiten basierender Algorithmus implementiert.

Die Applikation kann nun unterschiedlichere Informationen verarbeiten. Hierzu wurde aufgezeigt, wie verschiedene Informationen aus dem Netzwerk aus verschiedenen Quellen miteinander in Beziehung stehen und in welcher Art sie miteinander verbunden werden können. Mit diesem Wissen wurde ein flexibleres Datenschema entwickelt, welches schliesslich in der bestehenden Applikation implementiert wurde.

Aufgabenstellung

1. Ausgangslage

«Infrastructure-as-Code» resp. hier «Network-as-Code" geht davon aus, dass sämtliche für die Konfiguration von Infrastrukturen resp. Netzwerken benötigten Daten an einem zentralen Ort abgespeichert sind - in einer sogenannten «Single Source-of-Trouth» (SSoT). Basierend auf den Informationen dieser SSoT wird dann mit Hilfe einer «Automatisierungs Tool-Chain» die gewünschte Infrastruktur vollständig automatisiert bereitgestellt und konfiguriert. Damit entfallen manuelle Konfigurationen und Eingriffe an der Infrastruktur selbst. Werden Änderungen gewünscht, wird einfach die SSoT entsprechend angepasst – woraus die geänderte Konfiguration wiederum vollautomatisch in der Infrastruktur resp. im Netzwerk vorgenommen wird. Mit diesem (bereits aus der Software-Entwicklung bekannten Verfahren) wird die Agilität des Infrastrukturbetriebes massiv verbessert.

Die Problematik bei diesem Ansatz ist, dass es eine allumfassende «Single Source-of-Truth» in Firmen oftmals gar nicht gibt. Die Daten sind vielmehr in unterschiedlichsten Systemen (wie zum Beispiel Inventartool, Netzwerkmanagement-Anwendungen, IP-Adressmanagement usw. sowie im Netzwerk selbst) vorhanden und müssen von den Aktoren (meist von Hand) zusammengetragen und auf Richtigkeit überprüft (resp. von Inkonsistenzen befreit) werden. Die ausführenden Personen wissen meist nur dank langjähriger Erfahrung, welche Daten von welchem System benötigt werden und wo es potentielle Probleme bei der Datenqualität gibt.

In einer früheren Arbeit wurde eine Software entwickelt, welche aus verschiedenen Umsystemen Daten ausliest, diese auf Inkonsistenzen überprüft und - falls solche existieren - sie entweder automatisch korrigiert oder einem User zur Korrektur vorlegt. Erste Erfahrungen haben aber gezeigt, dass insbesondere der Aspekt des Data-Engineerings eine grosse Herausforderung darstellt und in einer weiteren Arbeit vertieft werden muss. Dazu gehören Fragen/Herausforderungen wie:

- Wie können die «richtigen» Daten in den verschiedensten Umsystemen gefunden werden, da es keinen umfassenden Suchbegriff gibt, der alle benötigten Attribute zu einem Gerät oder einem Service aus allen Umsystemen liefert? So sind z.B. Standortdaten, IP-Adressen und Gerätenamen nicht einfach auseinander ableitbar. Wie können also die dafür nötigen hierarchischen Filter (z.B. gib mir alle Switches an einem bestimmten Standort) auf die heute weitverbreiteten Umsysteme angewendet werden?
- Wie können die so gefunden Daten richtig «zusammengesetzt» werden. Wie geht man mit unterschiedlich langen oder unterschiedlich strukturierten Datensätzen um? Wie können zum Beispiel die zu einem Gerät gehörenden Parameter wie IP-Adressen, VLANs, VRFs, Topologie Informationen, Hostnamen usw. aus den verschiedenen richtig kombiniert werden?

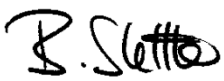
- Bei Inkonsistenzen soll ein Konfliktlösungs-Workflow angestossen werden, der einem Operator den Konflikt zur Auflösung präsentiert. Der Operator soll dann entscheiden, welchen Wert in die Source-of-Truth übernommen werden muss.
- Zur Nachvollziehbarkeit müssen solche Transaktionen gelogged und wenn möglich rückgängig gemacht werden können.
- Aus den Entscheiden der Operator könnte zudem automatisch «gelernt» werden, damit mehrfach auftretende Inkonsistenzen mit der Zeit vollautomatisch korrigiert werden können.
- Die aufgelösten Inkonsistenzen könnten zudem auch in die Ursprungssysteme zurückgeschrieben werden, damit sich die Datenqualität aller Systeme über die Zeit verbessert.

2. Aufgabe

In dieser Bachelorarbeit soll die Problematik des SSoT Data-Engineering anhand von 2-3 echten Firmenumgebungen analysiert und vertieft werden. Aus welchen Informationsquellen werden dort die Daten zur Konfiguration von Netzwerken zusammengetragen und in welcher Form? Welche der oben beschriebenen (oder weitere) Data-Engineering Probleme bestehen in diesen Firmen, wie gehen die Mitarbeiter damit um und wie könnten diese auch algorithmisch gelöst und automatisiert werden?

Aus der Analyse der 2-3 Firmenumgebungen soll ein Anforderungskatalog entstehen, der als Basis für die Weiterentwicklung der bestehenden Softwarelösung genutzt werden kann. Dies beinhaltet die häufig genutzten Umsysteme und ihre Schnittstellen sowie die Methodik, wie daraus generierten Konfigurationen und Services entstehen. Im Umsetzungsteil der Arbeit sollen konkrete Funktionen entwickelt werden, die die Data-Engineering Probleme adressieren und dem bestehenden Software-Tool hinzugefügt werden können.

Rapperswil, den 1.10.2021



Prof. B.Stettler

OST
Ostschweizer Fachhochschule

Oberseestrasse 10
8640 Rapperswil
Switzerland

T +41 58 257 41 11
info@ost.ch
ost.ch

Management Summary

Ausgangslage

Netzwerkgeräte werden heute meist noch von Hand konfiguriert. Dies kann verschiedene Gründe haben. Einer davon ist die Tatsache, dass die zur Konfiguration der Netzwerke benötigten Informationen in Firmen meist auf mehrere Tools verteilt sind. Es werden meist mehrere Tools von den Firmen eingesetzt, da diese jeweils einen spezifischen Anwendungsbereich haben. Dies kann jedoch dazu führen, dass Informationen verteilt abgelegt werden und damit mit der Zeit divergieren.

Korrekt aufbereitet könnten die verteilten Informationen dazu dienen, den Workflow für die Gerätekonfigurationen effizienter zu gestalten. Die Daten dazu sind bereits vorhanden, sie müssten nur richtig zusammengefügt und abgefragt werden.

Mit dem Ziel, diese Informationen zu vereinen und dadurch eine einfachere und wirksamere Netzwerkkonfiguration zu ermöglichen, wurde die Applikation „Kraken“ in einer Vorgängerarbeit entwickelt. Dieses Tool ermöglicht es, Quellen miteinander zu vereinen. Dabei auftretende Konflikte können vom Benutzer automatisch oder manuell gelöst werden.

Die vorliegende Arbeit beabsichtigt nun, die Datenhaltung zu vereinfachen. Es soll eine Möglichkeit gefunden werden, wie sich Informationen aus verschiedenen Quellen vereinen lassen.

Vorgehen

In dieser Arbeit wurde anhand von Interviews mit mehreren Firmen untersucht, wie Kraken deren Probleme in der Netzwerkautomation lösen könnte. Dazu sollte ermittelt werden, wie die aktuellen Konfigurations-Workflows der Firmen ablaufen. Es sollte beispielsweise festgestellt werden, welche Informationen aus welchen Quellen involviert sind und wie die Daten vom Engineer aufbereitet werden müssen. Zusätzlich wurden einige auf dem Markt befindliche Umsysteme genauer betrachtet.

Im Analyse-Teil wurde untersucht, wie Informationen aus verschiedenen Quellen miteinander verbunden werden können. Zum einen geht es um Daten, die ein Identifikationsmerkmal, wie zum Beispiel einen Namen, beinhalten. Zum andern stellt sich die Frage,

wie man Daten, die kein eindeutiges Merkmal aufweisen, identifizieren kann. Je nach vorhandenen Attributen sind bei letzterem zu wenige Informationen vorhanden, um eine Vereinigung durchführen zu können.

Im Implementationsteil wurde die Datenstruktur ausgetauscht, um die Handhabung der Daten zu vereinfachen. Mit einem Prototyp wurde das neue Konzept erprobt und anschliessend wurde die bestehende Applikation umgebaut.

Um den Umgang mit den Daten in Kraken praktisch testen zu können, wurde das in der Vorgängerarbeit entwickelte Beispiel-Netzwerk um zusätzliche Quellen erweitert. Nun war es das Ziel, die Informationen aus den fünf Beispielquellen von Kraken automatisch miteinander in Verbindung zu bringen.

Ergebnisse

Die in den Interviews erwähnten Pain-Points in der Netzwerkautomatisierung deckten sich nicht mit unseren Erwartungen. Anhand von Interviews und der Marktanalyse sollten praktische Beispiele erstellt werden, welche dann in der Umsetzung dazu dienen, die Anwendung zu verifizieren. Die in den Interviews besprochenen Worfklows liessen sich aber nicht direkt auf Kraken übertragen, weshalb für das Design eine eigene Beispielumgebung entwickelt wurde. Die Interviews haben aber einen guten Überblick über die Pain-Points eines Netzwerk-Engineers gegeben.

Beim Design ging es um die Frage, wie verschiedene Quellen vereint werden können, welche entweder kaum oder wenig Gemeinsamkeiten haben. Von den vorgeschlagenen Varianten, welche beim Sekundärvergleich angeschaut wurden, wurde eine Variante als umsetzbar eingestuft und umgesetzt. Die andere Variante, welche ein Hash-basiertes Verfahren verwendet, wurde als interessant eingestuft, hat aber auch neue Fragen aufgeworfen, welche im Rahmen dieser Arbeit nicht beantwortet werde n konnten.

Bei der Implementierung war der Kern der Frage, wie wir das vorhandene Normalisierungsformat und den Umgang von Kraken mit heterogenen Quellen verbessern. Dazu sollten die Erfahrungen aus Analyse und Design in der Software umgesetzt werden.

Die aus der Analyse ausgewählten fünf Quellen wurden zu einem sinnvollen Beispiel aufgebaut. Der Kern von Kraken wurde nach und nach für die neuen Quellen ausgerüstet. Zudem wurde der Kern so umgebaut, dass die Struktur nun für einen Programmierer einfacher zu handhaben ist, ohne dass aber die Komplexität zu stark zunimmt.

Änderungsverlauf

Datum	Version	Änderung	Autor
2021-09-20	1	Initiale Version des Dokumentes	Daniel Steudler
2020-10-23	2	Finale Version	Felix Kubli, Daniel Steudler

Inhaltsverzeichnis

1. Technischer Bericht

1.1. Einleitung und Übersicht

Mit automatisierten Abläufen sollen sich Netzwerke effizienter, schneller und günstiger konfigurieren lassen. Deswegen nehmen immer mehr Firmen die Automatisierung in Angriff. Doch das gestaltet sich oft als schwierig, da Netzwerke eine lange Zeit von Hand konfiguriert wurden und alle Werkzeuge und Prozesse darauf ausgerichtet sind.

Was dabei oft am Anfang steht, sind die Daten, die für die Konfigurierung der Geräte verwendet werden. Hier treten schon die ersten Probleme auf, denn die benötigten Informationen sind oft an verschiedenen Orten verteilt. Nur der zuständige Netzwerkengineer weiss, wo diese zu finden sind und muss sie sich in mühsamer Handarbeit zusammen kopieren.

Deshalb ist die Grundlage zur erfolgreichen Automatisierung im Netzwerk eine Single Source of Truth (SSoT)[1]. Die SSoT kennt alle Informationen und dient als einzige Wahrheit. Somit dient sie als Quelle für Konfiggeneratoren und als Basis für ein automatisiertes Netzwerk.

Doch wie kommt man nun zu einer SSoT? Das gestaltet sich häufig als schwierig, da zum Management eines Netzwerks oft verschiedene Programme und Tools verwenden, die ihren spezifischen Anwendungsbereich abdecken und die man auch behalten will.

Somit braucht es ein Tool, das die Informationen aus diesen Quellen importieren und vereinen kann.

In dieser Arbeit steht die Suche nach einer geeigneten Datenstruktur und einem geeigneten Datenmodell für eine SSoT im Vordergrund.

Erschwert wird diese Suche dadurch, dass es für jeden spezifischen Anwendungszweck und jede Domäne ein ideales Datenmodell gibt, doch ein Modell zu finden, das für alle perfekt passt, ist aufgrund der Unterschiede nahezu unmöglich.

> „The issue is that, with different technology domains and different protocols, come different data models. In order to assure cross domain use cases, the network management system and network operators must integrate all the technologies, protocols, and therefore data models. In other words, it must

> perform the difficult and time-consuming job of integrating & mapping information from different data models. Indeed, in some situations, there exist different ways to model the same type of information[2].“

Diese Arbeit beschäftigt sich mit der Schwierigkeit, Daten aus verschiedenen Quellen zu kombinieren, so dass der Informationsgehalt steigt. Es wurde der Frage nachgegangen, wie dass Netzwerkengineers das heute in der Praxis lösen. Verschiedene Quellen heisst hier auch, dass jede Quelle ihr eigenes Datenmodell hat, welches für den jeweiligen Anwendungszweck optimiert ist. Von diesem gilt es, die relevanten Daten zu extrahieren.

Es wurden Firmen verschiedener Grösse interviewt um herauszufinden wie diese im Alltag mit Daten aus verschiedenen Quellen umgehen, zum Beispiel wenn neue Geräte oder Kunden provisioniert werden. Unter anderem galt es auch herauszufinden, wo derzeit deren Pain-Points und Schwierigkeiten bei der Automatisierung sind. Kombiniert mit Recherche ist daraus eine Marktanalyse entstanden.

1.1.1. Problemstellung

Eine SSoT soll einem Netzwerkengineer helfen Netzwerkkonfiguration schneller und effizienter zu erstellen.

Die Schwierigkeit dabei ist, dass in der Praxis jede Firma ihr eigenes Toolset und Workflow einsetzt. Viele Tools bieten wiederum eine eigene API, mit einem Informationsset an. Deshalb muss die Applikation erweiterbar gestaltet werden, damit in Zukunft einfach neue Quellen angeschlossen werden können.

Die in der Vorgängerarbeit erstelle Applikation namens Kraken soll erweitert werden.

Die folgende Auflistung beinhaltet die Kernprobleme, die von Kraken angegangen werden.

- Datenqualität in den Quellen
- Konfliktlösung der Quellen
- Aufbereitung der Daten

1.1.2. Vorarbeit

Es existiert eine Vorgängerarbeit, welche an der Ostschweizer Fachhochschule (OST) von Julia Fritsche und Méline Sieber geschrieben wurde[3]. Die Vorgängerarbeit wiederum basiert auf einem *Proof of Concept*, welcher ebenfalls an der *OST* erstellt wurde. Kraken geht davon aus, dass die benötigten Informationen, um ein neues Netzwerk zu

konfigurieren, sich bereits in den Quellen/Umsystemen befinden. Kraken muss nun die Informationen aus diesen Quellen zusammenführen.

Die Vorgängerarbeit stellt eine funktionierende Plattform bereit, welche aus mehreren Quellen Informationen importiert und diese kombiniert bereitstellt.

Die Datenstruktur der Vorgängerarbeit hat jedoch noch Verbesserungspotenzial. Diese wurde damals aufgrund der Kompatibilität mit der Merge-Library *DeepDiff*[1] ausgearbeitet.

1.1.3. Methodik

Diese Arbeit ist in die Teile Analyse, Design und Umsetzung aufgeteilt.

Bei der Analyse geht es vor allem darum zu ermitteln, was es für Umsysteme gibt und was für allfällige Alternativen zu Kraken am Markt existieren. Diese wurde mithilfe von Interviews mit vier verschiedenen Schweizer Firmen erstellt.

Im Design geht es darum die interne Datenstruktur von Kraken zu verbessern. Ausserdem sollten Algorithmen entwickelt werden, welche mit widersprüchlichen Informationen in verschiedenen Quellen umgehen können.

In der Umsetzung wird die aus der Vorgängerarbeit entwickelte Applikation um diese Funktionalitäten erweitert.

1.2. Resultat aus der Marktanalyse

Die Marktanalyse ist im Detail im Kapitel 2 abgehandelt.

In der Marktanalyse wurden Produkte, welche am Markt existieren, miteinander verglichen. Der Vergleich der Tools gestaltet sich jedoch als „schwierig“, da die Tools häufig nicht in direkter Konkurrenz stehen und sie sich relativ frei miteinander kombinieren lassen. Deshalb wurden die Kategorien Tool, Plattform und Framework eingeführt.

Des Weiteren wurden die mit vier Schweizer Firmen durchgeführten Interviews analysiert. Die Firmen wurden nach ihrer Grösse ausgewählt.

Zwei Hypothesen wurden vor den Interviews aufgestellt:

1. Kleine Firmen verwenden Automatisierung, um einen Vorteil gegenüber grossen Firmen zu erlangen.

[1] `https://zepworks.com/deepdiff/current/`

2. Grosse Firmen benötigen Automatisierung, um Änderungen schnell durchzuführen.

Die Hypothesen beruhen auf Beobachtungen im Bereich der Webserver-Automation. Dort wird angestrebt, mit wenigen Mitarbeitern immer mehr und komplexere Systeme zu entwerfen und zu betreiben.

In den Interviews hat sich dann aber gezeigt, dass die Automatisierung in den verschiedenen Unternehmen einen unterschiedlichen Stellenwert hat. Die Interviews sind aber nur bedingt repräsentativ, da sie zwar verschiedene Firmen zeigen, drei davon aber im Internet Service Provider (ISP) Bereich tätig sind. Diese wurden bewusst so ausgewählt, da der Gedanke war, das im ISP-Bereich die Automatisierung mit Sicherheit ein Thema sein muss.

Bei allen interviewten Unternehmen ist die Automatisierung vor allem dort ein Thema, wo viele Änderungen geschehen und wo die Workflows definiert und standardisiert sind.

Als Beispiel sei Datapark genannt, welches Internet für Kabelnetzbetreiber im Raum Wil, SG bereitstellt. Datapark hat ein eigenes Tool namens SAM, mit welchem die Kabelnetzbetreiber die Digital Subscriber Line Access Multiplexers (DSLAMs) automatisch konfigurieren können. Dieser Workflow besitzt zwei wichtige Eigenschaften: Die DSLAMs werden immer identisch konfiguriert, und es ist ein Task, welcher mehrmals täglich ausgeführt wird.

Am anderen Ende der Grössenskala befindet sich die Migros. Die Migros ist auch spannend, da sie sowohl Wide Area Network (WAN), als auch Local Area Network (LAN) Geräte verwalten. Zudem hat die Migros eine Grösse erreicht, bei welcher die kommerziellen Produkte des Geräteherstellers an ihre Grenzen kommen.

Bei Migros ging es spezifisch darum, wie der Workflow für eine neue Site aussieht. Das Problem bei Migros sind nicht primär die Quellen an sich, sondern die Datenqualität in den Quellen. Dabei erschweren aber folgende Punkte die Automatisierung mit einer SSoT.

- Verschiedene Organisationen haben unterschiedliche Wünsche bei der Konfiguration der Geräte.
- Sehr viele Networkengineers haben Zugriff auf die Geräte.
- Die Qualität der Daten in den Systemen (zum Beispiel Attribute die sich nur menschenlesbar, aber nicht maschinenlesbar in der Beschreibung befinden)

Sie besitzen bereits einen Konfigurationsgenerator, welcher ihnen hilft, Netzwerkkonfigurationen zu erstellen. Dieser kennt pro Routertyp ein Template. Der User kann dann die benötigten Virtual Local Area Networks (VLANs) auswählen. Die VLANs sind jeweils Unternehmensweit definiert, und die benötigten Features an einem Standort definieren

dann welche VLANs benötigt werden. Eine Schwierigkeit ist, dass die Subnetze über die Zeit inkonsistent definiert wurden. Im schlimmsten Fall gibt es an einem Ort oder Standort ganz viele Subnetze, von denen aber nur wenige relevant sind.

Eine neue Version des Konfigurationsgenerators ist in Entwicklung. Der Konfigurationsgenerator soll ähnliche Funktionalitäten haben wie Kraken sie besitzt, ist aber auf die Bedürfnisse der Migros zugeschnitten.

Init7 hofft, dass sich das Problem der SSoT mit Netbox[2] lösen lässt. Init7 migriert derzeit von einem Wiki-zentrischen System zu Netbox als Source of Truth (SoT).

Netrics setzt ebenfalls auf Netbox. Es sind nach eigenen Angaben etwa 80% der Informationen in Netbox auffindbar.

Alle Interviewten Parteien bis auf Migros setzen Netbox ein. Netbox versteht sich als eine 80% Lösung[4], was es sehr flexibel macht. Darauf Services aufzubauen ist komplex, wird aber durchaus von Firmen praktiziert.

Auf die Annahme, dass sich alle benötigten Informationen bereits in den Umsystemen befinden, deuten keine der Interviewten hin. Der von Init7 bereitgestellt Workflow zeigt, wie die Informationen nach und nach in den Umsystemen eingetragen werden. Der Engineer muss dabei aufgrund der Anforderungen des Kunden Entscheidungen treffen.

1.3. Lösungsansatz

Die Forschung, welche in der Vorgängerarbeit angefangen wurde, soll weitergezogen werden.

Als Beispielanwendungsfall für Kraken als SSoT wird eine Template-Engine genannt.

Der Fokus dieser Arbeit liegt darauf, wie ein Datenmodell/Normalisierungsformat aussehen kann, welches in einer SSoT verwendet werden kann.

Da sich die Informationen und Workflows nicht mit denen von Kraken decken, welche sich auch in einer so kurzen Zeit akkurat nachbauen liesse, wurde eine eigene Quellstruktur entwickelt. In 3.3.1 wird die Quellstruktur und das dazugehörige Data-Engineering näher erläutert.

Das Datenmodell hat aus Sicht dieser Arbeit folgende Voraussetzungen:

- Einfache Erweiterbarkeit durch einen Entwickler oder im Idealfall einen Netzwerkengineer.

[2] `https://netbox.readthedocs.io/en/stable/`

- Hohe Flexibilität; es eignet sich für verschiedene Einsatzzwecke, wie die Suche und verschachteltet Abfragen.
- Effizienz; es ist auch noch bei vielen Datensätzen performant.

Ein Normalisierungsformat hat ausserdem noch folgende Voraussetzungen:

- Lesbarkeit für Netzwerkengineers
- Einfache Verwendbarkeit für Entwickler/Engineer
- Redundanzen; so viel wie nötig, so wenig wie möglich

Das Datenmodell kann sich aber je nach Applikationsschwerpunkt unterscheiden. Spielt zum Beispiel der Kontext der Daten keine Rolle, reicht ein Key-Value-Store. Haben die Daten einen Kontext, muss dieser erhalten bleiben, da sonst potenziell Informationen verloren gehen.

In Fall dieser Arbeit handelt es sich um kontextabhängige Informationen, welche zum Teil stark verschachtelt sind. Daher bietet sich für diesen Anwendungsfall entweder eine Baumstruktur oder ein Graph an.

Die Anforderungen an Kraken sind daher:

- Verbinden mit Quellen
- Importieren und Normalisieren der Daten der Quellen
- Daten zusammenführen, Konflikte finden und bereinigen
- Daten aufbereiten

Der Konfliktlösungsteil und der Aufbereitungsteil sind orthogonal zueinander zu betrachten. Dies bedeutet, dass die Probleme einzeln für sich betrachtet zu einer eigenen optimalen Lösung führen würden. Für eine Lösung, welche beide Problemfelder abdeckt, sind Kompromisse in beide Richtungen nötig.

Der Konfliktlösungsteil beschäftigt sich vor allem damit, wie verschiedene Informationen kombiniert und bereinigt werden können, während sich der Aufbereitungsteil mit der Abfrage von Information beschäftigt.

Anhand von fünf Beispielquellen wird ermittelt, wie verschiedene Informationen zusammengehängt werden können und wie sie in Beziehung zueinander stehen. Mit den Erkenntnissen dieses Teils werden dann Daten-Objekte definiert, die in Kraken implementiert werden. Um den Vergleich dieser Objekte zu verbessern, werden Ansätze erarbeitet, von denen schliesslich einer implementiert wird. Die bestehende Applikation wird dann entsprechend umgebaut und erweitert.

Damit technische Probleme und potenzielle Schwierigkeiten bei der Implementation frühzeitig erkannt werden können, wurde ein Prototyp[3] erstellt.

Es wurden damit Ansätze zum Vergleich von Objekten getestet, sowie der Umgang mit Python-Objekten anstelle von Python-Dicts.

1.3.1. Verworfene Ansätze

Um Kraken weiterzuentwickeln und das Problem des Data-Engineering anzugehen, wurden im Verlauf dieser Arbeit verschiedene Ansätze in Betracht gezogen und einige davon verworfen. In den folgenden Absätzen werden wir etwas genauer darauf eingehen.

1.3.1.1. OpenConfig als Hauptformat

In der vorgehenden Arbeit wird OpenConfig[4] als mögliches Datenformat für das Normalisierungsformat beschrieben, da es nahe am Netzwerk sei [3, S. 16]. Deshalb wurde untersucht, inwiefern es möglich ist die Daten in Kraken im OpenConfig NETCONF/YANG-Format aufzubereiten.

Dabei wurde jedoch festgestellt, dass einiges gegen eine solche Verwendung spricht. Zum einen ist das vorhandene Tooling auf die programmatische Konfiguration von Geräten ausgelegt und nicht auf die Aufbereitung von Informationen zu Geräten. Die Erweiterung der vordefinierten Datenstruktur um Metadaten ist nicht trivial und hätte ein tiefes Verständnis von OpenConfig und NETCONF/YANG vorausgesetzt. Zusätzlich hätte hier Tooling für den Umgang mit OpenConfig YANG geschrieben werden müssen. Da OpenConfig NETCONF/YANG möglichst flexibel sein will, werden dort Eltern-Kind Beziehungen in der Datenstruktur abgebildet. Dies macht es aufwändig das Ganze zu verwenden und hätte nach Einschätzung der Autoren das Zeitbudget dieser Arbeit um ein vielfaches überschritten.

Fokus auf Switch und Router Konfiguration Der Fokus von OpenConfig NETCONF/YANG liegt in der Konfiguration. Die Modelle beginnen bei Device und gehen von dort abwärts. Alles, was sonst noch an Informationen für ein Netzwerk benötigt werden könnte, ist nicht vorgesehen in OpenConfig.

[3] `https://gitlab.ost.ch/diktyo/ba-sena-adorabilis`
[4] `https://openconfig.net/projects/models/`

OpenConfig sind standardisierte Modelle Änderungen am Modell kommen vom Standardisierungsgremium her. Kraken soll aber mehrere Modelle miteinander kombinieren und muss entsprechend anpassbar sein. Mit OpenConfig wäre das nur möglich, wenn der OpenConfig Teil des Baumes nicht angerührt wird, und durch uns von oben ergänzt wird. Das ist grundsätzlich auch unsere Idee, aber ohne dass wir auf OpenConfig genauer eigehen, sondern dass wir es in unserem Modell wie eine Blackbox behandeln.

YANG Die Modellierungssprache YANG ist nur im Netzwerkbereich bekannt, worunter das Tooling etwas leidet. Die Tools gehen kaum über das Konfigurieren von Netzwerkgeräten hinaus.

Die YANG Software Development Kits (SDKs) haben einen starken Fokus auf der programmatischen Konfiguration von Netzwerkgeräten.

Zudem ist YANG zwar menschenlesbar, aber auch nur in Grenzen.

Im Umgang mit YANG werden Tools vermisst, die das Format in menschenlesbarer Form darstellen und Tools die das grafische Editieren von YANG-Models ermöglichen. Somit könnte mit diesen Models effizienter umgegangen werden.

Die effektiven Daten liegen bei der Verwendung nicht in YANG vor, sondern entweder in XML oder JSON. Das macht es schwierig, Daten zu einem selbst erstellen Modell zu generieren.

Mit einem Fokus auf OpenConfig YANG hätte diese Arbeit Tools für den Umgang mit YANG hervorbringen können, aber kein Datenmodell. Die Arbeit hätte sich mit diesem Weg zu sehr von der Aufgabenstellung entfernt.

Komplexität Alleine die UML-Repräsentation von *Interface* ist etwa zehnmal grösser, als das in dieser Arbeit entwickelte Datenmodell. Diese Komplexität kann damit erklärt werden, dass OpenConfig eine möglichst grosse Feature-Abdeckung anstrebt, um möglichst viele Geräte zu unterstützen. Im OpenConfig Datenmodell müssen alle möglichen Konfigurationsmöglichkeiten abgebildet werden und nicht wenige der Konfigurationselemente sind wiederum in eigenen YANG-Klassen enthalten. Diese Verschachtelung ist für die Grösse verantwortlich.

Erweiterung des OpenConfig Modells Ein Ansatz war, das OpenConfig Modell nach oben zu erweitern. Dieser Ansatz wurde verworfen, weil er viel mehr Wissen über die Domäne und das Modellieren von YANG vorausgesetzt hätte. Dafür bräuchte man Erfahrung in der Verwendung dieser Technologie in der Praxis. Ausserdem sollte eine Erweiterung des Models vom Standardisierungsgremium kommen. Da der Kontext von

OpenConfig die Konfiguration von Netzwerkgeräten ist und die Informationen zum Inventar nicht direkt dazu gehören ist fraglich, ob dies je kommt, denn dies wäre eine Verwässerung des Kontextes.

1.3.1.2. OpenConfig Actual vs. Desired

Ein Ansatz war es, Konfigurationen aus Routern auszulesen und diese mit einer gespeicherten Kopie zu vergleichen. Die Konfiguration sollte dabei ins OpenConfig Format übertragen und die vendorspezifischen Informationen, welche nicht in OpenConfig existieren werden, sollten separat abgelegt werden.

Dieser Ansatz hat aber das gleiche Problem wie der Ansatz von OpenConfig als Hauptformat.

1.4. Umsetzung und Resultate

Zuerst wurden mit einem Prototyp Erfahrungen im Umgang mit Objekten und deren Vergleich gewonnen, welche dann bei der Umsetzung angewandt wurden. Dann wurde ein Datenschema mit dem Device im Zentrum aufgebaut, da das Ziel der Netzwerkautomatisierung die Konfiguration von Geräten ist. Es wurden fünf Quellen definiert, deren Attribute und Informationen dann zu einem Datenschema verschmolzen wurden. Mit Daten zu den Quellen wurde das Beispiel-Netzwerk der Vorgängerarbeit erweitert um die Applikation effektiv testen zu können. Das erstellte Netzwerk ist in der Abbildung 1.1 ersichtlich.

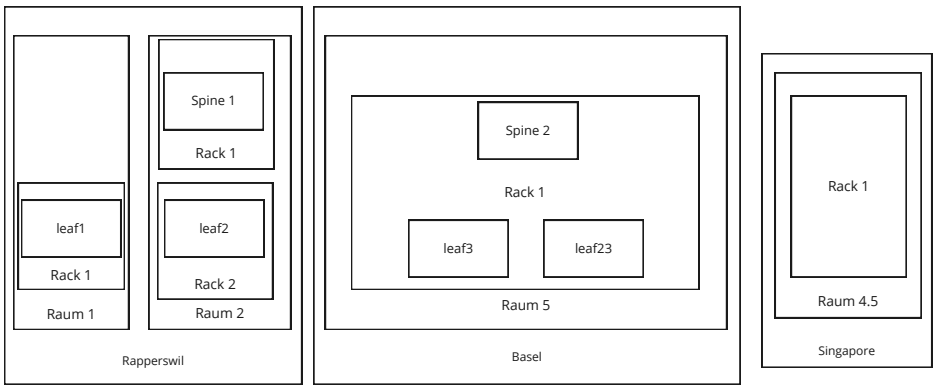


Abbildung 1.1.: Beispiel Netzwerk Geräte

Die Informationen der Quellen wurden dann entsprechend in Netbox und Excel-Sheets eingetragen.

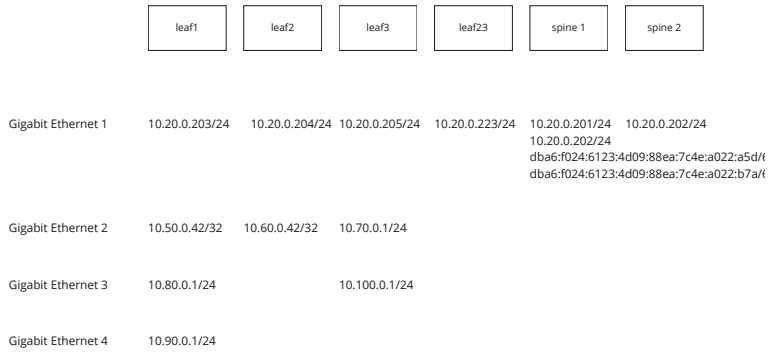


Abbildung 1.2.: Beispiel Netzwerk, IPs

Für die definierten Quellen wurden jeweils neue Konnektoren, sowie deren Mapper geschrieben, die in der folgenden Auflistung zu sehen sind.

- SourceIPPrefixConnector (JSON Netbox API)
- SourceIPConnector (JSON aus Netbox API)
- SourceInventoryConnector (CSV aus Excel)
- SourceLocationsConnector (CSV aus Excel)
- SourceSegmentationConnector (CSV aus Excel)

Um bei einem inkonsistenten Identifikationsattribut eine Entität trotzdem identifizieren zu können, wurde angeschaut die **Identifikation** verbessert werden kann, angenommen man habe mehr als ein überlappendes Attribut. Es wurden mehrere Ansätze evaluiert und ein auf Ähnlichkeiten basierender Algorithmus wurde schliesslich implementiert, der beim Vergleich der Objekte angewandt werden kann.

Um die Daten der Konnektoren vereinen zu können, wurden mehrere **Merger** implementiert. Der auf der Library *DeepDiff* basierende Merger der Vorgängerarbeit[5] wurde für die Verwendung mit Objekten umgeschrieben und dient als Basis für die spezifischeren Merger. Um die fünf Quellen zu vereinen wurde der *CombinedMerger* implementiert, der

[5] `https://gitlab.ost.ch/ins-stud/ba21-nettowel-kraken/-/blob/d8f12aaa341613f12ac3c45ee27d0020ef1bbaba/kraken/merger/merge.py`

die verschiedenen Merger kombiniert und deren Aufruf kontrolliert. Die restlichen Merger verbinden Informationen von jeweils zwei Quellen und beinhalten die dafür nötige Logik.

Dass in dieser Arbeit mehrere Merger implementiert wurden, hat den Grund, dass die Quellen heterogene Informationen bereitstellen, die teilweise nur indirekt mit dem Device zu tun haben. In der Vorgängerarbeit musste jeder Konnektor dem Merger eine vom Device ausgehende Struktur übergeben, weshalb die Implementation dort mit einem Merger funktionierte [3, S. 51, 55, 83].

Die Applikation der Vorgängerarbeit konnte zwar auch heterogene Informationen mergen, machte dies aber im Konnektoren selbst. Dort wurden alle Informationen zu Netbox mit der `unite_data` Funktion zusammengeführt [3, S. 55].

Da nun nicht alle Quellen und somit nicht alle Konnektoren einen Device-Namen kennen, Musste der Merge-Vorgang angepasst werden. Um diese `unite_data` Funktion[6] abzulösen und damit das Zusammenführen von Daten nur von den Mergern gemacht wird, wurden separate Konnektoren für die Daten aus Netbox geschrieben. Diese sind die Konnektoren `SourceIPConnector` und `SourceIPPrefixConnector`.

Um Daten aus CSV-Dateien zu filtern, wurde eine Helferfunktion geschrieben. Zudem wurden für die neuen Konnektoren neue **Filter** implementiert.

Mit den neuen Konnektoren und dem neuen Datenschema mussten die **Validatoren** angepasst werden. Zur Überprüfung der Kraken-Konfiguration musste nur das Kontrollschema aktualisiert werden, sprich die neuen Konnektoren mussten angegeben werden und die Option einen Merger zu wählen wurde hinzugefügt. Um das Merge-Template zu validieren musste jedoch eine eingene Funktion geschrieben werden. Bei inkorrekten Einstellungen werden dem Benutzer Tipps gegeben, weshalb die Datei inkorrekt ist.

Damit die Strukturierung der Ausgabe von Kraken dem Zweck der Verwendung angepasst werden kann, besteht nun die Option eine **Template**-Funktion angeben zu können. Diese Funktion wird dann auf jedes Basis-Element der gemergten Daten angwendet. So kann die Funktionalität bezüglich des Outputs erweitert werden, ohne dass die Applikation selbst verändert und neu installiert werden muss.

1.5. Ergebnisdiskussion

Am Anfang war es wichtig die Problemdomäne zu verstehen. Es war wichtig zu erkennen, wieso das Erstellen einer umfassenden Datenstruktur im Netzwerkbereich so schwierig

[6] Stand 13.01.2022 `https://gitlab.ost.ch/ins-stud/ba21-nettowel-kraken/-/blob/d8f12aaa341613f12ac3c45ee27d0020ef1bbaba/kraken/connectors/mapper.py`

ist und Teilprobleme zu identifizieren, in die das Hauptproblem aufgeteilt werden kann und wo Lösungsansätze bestehen.

Die Marktanalyse hat einerseits geholfen zu verstehen, wie Netzwerke von Firmen verwaltet werden. Dies hat uns andererseits aber auch viel Zeit gekostet, was sich am Schluss beim Design und der Umsetzung bemerkbar machte. Rückblickend erhofften wir uns von den Interviews konkretere Informationen zu den Thematiken der Aufgabenstellung. Die im Interview erwähnten Pain-Points stimmten jedoch nach der ersten Analyse nicht mit der Aufgabenstellung rund um Kraken überein. Mit besserem Verständnis der Problemdomäne wurde dieses Missverständnis schliesslich aufgeklärt. Diese Fehlinterpretation des Zieles von Kraken hat die Suche nach einem Datenmodell verlangsamt.

Die Analyse der Vorarbeit und der Interviews haben darauf hingedeutet, dass ein internes Datenformat einiges an Komplexität hat. Deshalb wollten wir mit OpenConfig ein in der Netzwerk-Industie etabliertes Datenschema verwenden und als Kern von Kraken implementieren. Bei den Versuchen mit der OpenConfig-Definition umzugehen wurde jedoch klar, dass es für unseren Anwendungsfall derzeit nicht geeignet ist. Deshalb entschieden wir uns ein eigenes Datenmodell zu entwickeln.

Mithilfe des Beispiel-Netzwerks und den Beispiel-Quellen konnte dann das Datenschema konzipiert werden und die Ansätze zum Vergleichen der Objekte erarbeitet werden.

Leider konnte die komplette Feature-Parity mit der Vorgängerarbeit nicht komplett erreicht werden, da die Zeit für die Implementation des Multilevel-Filters nicht mehr reichte. Das liegt hauptsächlich daran, dass mit der vorhandenen Zeit eine ausreichende Testung des Features nicht möglich gewesen wäre. Zudem basieren die Tests der *Collector*-Klasse noch immer auf den Konnektoren der Vorgängerarbeit, da für dessen Testung viele Testdaten geschrieben wurden, die es zu ersetzen gäbe.

Für den Merge-Vorgang war ursprünglich die Verwendung eines einzelnen Mergers vorgesehen. Dieser kann die Daten von allen Konnektoren direkt mergen kann, mit der Ausnahme der Zuweisung von IP-Addresse zu Prefix. Die angedachte Lösung hat jedoch nicht wie erwartet funktioniert und deshalb musste auf die Implementation von mehreren Mergern ausgewichen werden. Dadurch wurde die Flexibilität der Applikation etwas eingeschränkt.

Zur besseren Identifikation von Objekten konnte ein Algorithmus implementiert werden. In welchen Fällen, dass dieser gut funktioniert und in welchen er nicht angewendet werden sollte, konnte jedoch nicht überprüft werden, da dazu reale Daten vonnöten wären.

Der Entscheid die manuelle Konfliktlösung nicht weiterzuverfolgen konnte das Scope erfolgreich eingrenzen, auch wenn dadurch Abstriche bei der Konfliktlösung gemacht

wurden. Dadurch konnte auch nicht darauf eingegangen werden, wie man aus den Entscheiden der Benutzer lernen kann. Zudem können bereinigte Inkonsistenzen noch nicht in die Umsysteme zurückgeschrieben werden.

Alle anderen Features konnten jedoch erfolgreich implementiert werden, wenn auch die Implementation oft komplexer und schwieriger war. Das war zum einen der Verwendung von Objekten geschuldet und zum anderen der Verwendung von heterogenen Quellen. Dadurch konnte die Arbeit der zukünftigen Quellenanbindung auch nicht erleichtert werden.

Der Hauptanwendungszweck konnte jedoch erfolgreich erstellt werden. So können nun mehrere Quellen angegeben werden, deren Daten werden in die Objekte des Datenmodells übertragen, basierend auf deren Präzedenz zusammengeführt und können über eine Template-Funktion in die gewünschte Form gebracht werden.

1.6. Ausblick und Weiterentwicklung

1.6.1. Templates

Momentan ist die Ausgabe von Kraken immer gleich strukturiert. Es wird eine Liste von Devices mit allen verschachtelten Objekten serialisiert. In Zukunft soll es möglich sein eine Funktion z.B. als Plugin anzugeben, die für die Ausgabe verwendet werden soll und die gemergten Daten zur Verfügung hat.

1.6.2. Bereinigung

Aus Zeitgründen war es, nicht möglich allen nicht mehr verwendeten Code der Vorgängerarbeit zu entfernen, da gewisse, noch relevante Tests von diesem Code abhängen. Die Ordner `netbox` und `powerdns`, sowie die `unite_data` Funktion wurden deshalb in den Ordner `legacy_connectors` verschoben.

1.6.3. Generalisierung Merge

Im Moment braucht es mehrere Merger um die Quellen zusammenzuführen. Der Grund dafür ist, dass zum Beispiel von der einen Quelle eine Liste an Devices mit Racks und von der anderen eine Liste an Racks mit Locations zurückgegeben wird. Somit kann man nicht einfach die Devices mit den Racks vergleichen, sondern muss die verschachtelten Racks mit den anderen Racks vergleichen. Damit nur noch ein Merger für alles verwendet

werden kann, muss dieser, mit einer potenziell rekursiven Suche in den Daten der Quellen, nach dem richtigen Objekttyp für den Vergleich suchen.

1.6.4. Manuelle Konfliktlösung

Im Rahmen dieser Arbeit wurde die manuelle Konfliktlösung nicht behandelt. In der Vorgängerarbeit wurde eine Webseite geschrieben, die die manuelle Behebung von Konflikten bei Attributwerten ermöglicht. Für den Ausblick und die Weiterentwicklung der Webapplikation verweisen wir auf die Vorgängerarbeit.

1.6.4.1. Verifizierung

Im Normalfall sollte die Laufzeit von Kraken nicht sehr lange dauern. Wenn sie aber trotzdem lange dauert, kann es sein, dass die Daten in den Quellen, auf die über eine API zugegriffen wird, schon wieder geändert haben. Gerade bei der manuellen Konfliktlösung könnte das zu einem Problem werden. Deshalb soll Kraken vor der Zurückgabe der gemergten Daten verifizieren, dass sich keine Daten der Quelle verändert haben und wenn doch, sollen die von Kraken verarbeiteten Daten aktualisiert werden.

1.6.5. Konnektoren durch Konfiguration

Derzeit ziehen Änderungen am Model oder an Konnektoren einige Änderungen am Code nach sich, da ein grosser Teil des Mapping-Vorgangs programmatisch geschieht. Verschachtelte Objekte werden mit Programmcode erstellt, während auf der ersten Ebene ein Dict mit Zuweisungen von Quell-Attributen zu Kraken-Attributen verwendet werden kann. Schön wäre es, wenn die Definition des Models und die Definition der Konnektoren in Konfigurationsfiles ausgelagert werden könnten. Ist es möglich, dass eine Mapper-Definition als Zuweisung von Attributnamen definiert werden kann?

```
{
  "device_name": "Device Name",
  "rack": {
    "rack_number": "Racknummer"
  }
}
# die zu befüllende Struktur
Device(
  device_name=...
  rack=Rack(
    rack_nummer=...
  )
)
```

Listing 1.1: einfaches Beispiel Mapperdefinition

1.6.6. Filter

Momentan kann man mehrere Attribute festlegen, nach denen gesucht werden soll. Wenn nun z.B. der Filter `DeviceName` mit `device_name="leaf1"` gesetzt wird, wird jeder Konnektor, der diesen Filter unterstützt, nach *leaf1* suchen.

Wenn es nun aber eine Quelle gibt, die `DeviceName` nicht unterstützt, werden von dieser Quelle potenziell mehr Daten geladen als nötig. In der Vorgängerarbeit wurde dieses Problem mit dem Multilevel-Filter angegangen und konnte in dieser Arbeit aus Zeitgründen nicht umgesetzt werden. Es soll möglich sein, eine Suche zu definieren, die auf den unterstützten Quellen nach `DeviceName` sucht und dann basierend auf den von diesen Konnektoren zurückgegebenen Daten die anderen Quellen durchsucht. So werden nur so viele Daten wie nötig abgerufen, gerade wenn Daten von APIs über das Netzwerk gesendet werden müssen.

2. Marktanalyse

Um besser zu verstehen, wie sich Kraken im Markt einfügt und wo die Bedürfnisse des Marktes sind, wurde eine Marktanalyse durchgeführt. Zum einen geht es darum zu verstehen, was für Produkte derzeit am Markt existieren, zum anderen welche Workflows von Firmen eingesetzt werden, um neue Geräte oder Teilnetzwerke zu konfigurieren. Hiermit sollte auch ermittelt werden, auf was bei der Weiterentwicklung von Kraken geachtet und worauf fokussiert werden soll. Die Marktanalyse besteht zum einen aus Recherche und zum anderen aus Interviews, die mit Firmen geführt wurden.

2.1. Software am Markt

Von der existierenden Software lässt sich Kraken noch am ehesten mit Nautobot in Kombination mit dem Golden Configuration Plugin vergleichen. Nautobot ist dabei ein Fork von Netbox. Netbox wurde geforkt, da Nautobot und Netbox nicht dasselbe Ziel verfolgen. Mit einem Fork haben die Nautobot-Macher nun mehr Entscheidungsfreiheit un Kontrolle über die Entwicklung[5]. Mehr dazu in Abschnitt 2.1.1.3.

Es wurden keine konkreten Zahlen zu der Verbreitung der Produkte gefunden. Das liegt an folgenden Gründen.

1. Viele Open-Source-Produkte sind im Einsatz.
2. Firmen setzen die gleiche Software für verschiedene Zwecke ein.
3. Grosse Controller-Anwendungen sind auf Hersteller-Produkte begrenzt.
4. Kommerzielle Applikationen, welche mehrere Hersteller unterstützen, veröffentlichen keine Zahlen.

Um trotzdem die Popularität der Open-Source-Anbieter vergleichen zu können, wurden die Stars auf Github verglichen. Github Stars als Metrik sind nicht wirklich aussagekräftig was die Verbreitung des Produktes angeht, aber sie können einen Richtwert über die Bekanntheit des Produktes geben. Für den Verleich ist als mehr die Grössenordnung der Anzahl Stars interessant.

2.1.1. Alternativen/Umsysteme zu Kraken

Es ist schwierig die Software am Markt direkt miteinander zu vergleichen, da sich je nach Software der Fokus und die Philosophie, wie mit der Software gearbeitet wird, unterscheiden. Die Grenze zwischen Alternative und Umsystem zu Kraken ist fliessend. Die untersuchten Produkte haben nicht zwingend den Funktionsumfang von Kraken, können aber über Plugins oder im Zusammenspiel mit anderer Software auf ein ähnliches Niveau gebracht werden.

Wir unterscheiden dabei zwischen folgenden Kategorien:

Tool Das Tool sieht sich als kleines Zahnrad im Gesamtbild. Es deckt nur einen Bereich ab.

Plattform Die Plattform sieht sich als Komplettlösung für (fast) alles. Meist setzt dies aber einen bestimmten Hersteller voraus. Die Plattformlösung kann mit weiterer Software des Herstellers interagieren und kombiniert werden.

Framework Das Framework bietet dem Anwender die meiste Freiheit.

Die Alternativen / potenziellen Umsysteme zu Kraken werden wie folgt zugeordnet.

- Tool
  - Netbox (IP Adress Management (IPAM), Data Center Infrastructure Management (DCIM))
  - Bluecat (IPAM, DNS, DHCP)
  - Really Awesome New Cisco config Differ (RANCID) (Backup)
  - NSOT (IPAM, DCIM)
- Plattform
  - Netbox (durch Plugins)
  - Cisco Prime[1] (WAN)
  - Cisco DNA Center[2] (LAN)
  - Nautobot mit *Golden Configuration* Plugin (IPAM, DCIM, Automation)
- Framework

[1] `https://www.cisco.com/c/en/us/products/cloud-systems-management/prime-infrastructure/index.html`

[2] `https://www.cisco.com/c/de_de/products/cloud-systems-management/dna-center/index.html`

– Ansible (Automatisierungsframework)

2.1.1.1. Ansible

Ansibles[3] Einsatzgebiet beschränkt sich nicht auf Netzwerke. Im Netzwerkbereich wird Ansible vor allem für die programmatische Konfiguration von Geräten verwendet, auch im Zusammenspiel mit anderen Tools. Im Interview mit Datapark wurde erwähnt, dass es für die Konfigurationsgenerierung eingesetzt wird. Die Möglichkeiten sind hier nahezu grenzenlos. *Red Hats* Ansible hat 50k Stars auf GitHub[4].

2.1.1.2. Netbox

Versteht sich als 80% Lösung[5] im Bereich IPAM und DCIM. Der Fokus der Funktionalitäten liegt auf IPAM und DCIM. Netbox baut auf Python und Django auf und bietet eine gute Plugin-Infrastruktur. Der Aufbau der Informationen erinnert an Excel, da die Daten meist tabellarisch angezeigt werden. Das einfache Design der Architektur vereinfacht es die Applikation zu erweitern, weshalb das Projekt gut unterhalten ist[4]. Dies ist als Open-Source-Projekt ein Vorteil. Netbox hat 9.2k Stars auf Github[6].

2.1.1.3. Nautobot mit Golden Configuration

Nautobot[7] ist ein Produkt von *Network to Code*[8]. Es baut auf Netbox auf, setzt aber den Fokus stärker auf die Automatisierung. Das Produkt steckt noch in den Kinderschuhen, könnte aber interessant sein. Es soll grundsätzlich dieselben Probleme wie Kraken lösen:

1. Configuration Backups - Is a Nornir process to connect to devices, optionally parse out lines/secrets, backup the configuration, and save to a Git repository.
2. Intended Configuration - Is a Nornir process to generate configuration based on a Git repo of Jinja files to combine with a GraphQL generated data and a Git repo to store the intended configuration.

[3] `https://www.ansible.com/`
[4] `https://github.com/ansible/ansible`
[5] `https://netbox.readthedocs.io/en/stable/#keep-it-simple`
[6] Stand 12.01.2022 `https://github.com/netbox-community/netbox`
[7] Stand 12.01.2022 `https://www.networktocode.com/nautobot/`
[8] Stand 12.01.2022 `https://www.networktocode.com/`

3. Source of Truth Aggregation - Is a GraphQL query per device that creates a data structure used in the generation of configuration.
4. Configuration Compliance - Is a process to run comparison of the actual (via backups) and intended (via Jinja file creation) Command Line Interface (CLI) configurations upon saving the actual and intended configuration. This is started by either a Nornir process for cli-like configurations or calling the API for json-like configurations

[6]

Aber im Gegensatz zu *Kraken* holt es sich die Informationen nicht aus anderen Quellen, sondern Nautobot soll die Quelle der Wahrheit sein. *Network to Codes* Golden Configuration hat derzeit 41 Stars auf GitHub[9].

2.1.1.4. NSot von Dropbox

Network Source of Truth (NSoT) ist ein API-first IPAM.[10] Es wurde von Dropbox entwickelt und unterstützt IP-Adressen, Netzwerkgeräte und Netzwerk-Interfaces. *Dropboxs* NSoT hat 358 Stars auf Github[11].

2.1.1.5. StableNet von Infosim

SableNet ist laut der eigenen Webseite eine Lösung für alles, was Netzwerke betrifft [7]. Preise sind aber nur auf Anfrage erhältlich.

2.1.2. Umsysteme aus den Interviews

2.1.2.1. Cisco Prime

Cisco Prime wird für das Management von WAN verwendet und ist bei Migros im Gebrauch.

2.1.2.2. Cisco DNA Center

Das Cisco DNA Center wird für das Management von LAN und WLAN verwendet und ist ebenfalls bei Migros im Gebrauch.

[9] Stand 12.01.2022 `https://github.com/nautobot/nautobot-plugin-golden-config`
[10] `https://github.com/dropbox/nsot`
[11] Stand 13.01.2022 `https://github.com/dropbox/nsot`

2.1.2.3. Cisco ISE

Cisco ISE[12] wird von der Migros als RADIUS verwendet.

2.1.2.4. Bluecat IPAM

Bluecat[13] wird von der Migros als IPAM verwendet und um Domain Host Configuration Protocol (DHCP) zu managen.

2.1.2.5. Wiki

Ein Wiki wird für die Wissensverwaltung und manchmal auch für die Konfigurationsverwaltung verwendet.

2.1.2.6. CRM

Im Customer Relation Management (CRM) befinden sich zum Teil kundenspezifische Informationen.

2.1.3. Vergleich

Bei Nautobot bietet sich ein direkter Vergleich mit Kraken an. Da die Ziele der beiden Projekte sich ziemlich ähnlich sind.

Folgende Kriterien werden zum Vergleich verwendet:

Normalisierung Die Daten, welche aus verschiedenen Quellen stammen, werden normalisiert.

Aufbereitung externer Quelle Externe Quellen werden für die Daten in der SoT miteinbezogen.

Konfliktlösung Konfliktbereinigung von Informationen

Automatische Konfliktlösung Die Konfliktlösung kann auch automatisiert vollzogen werden.

Open-Source Das Produkt ist Open-Source.

[12]Stand 12.01.2022 `https://www.cisco.com/c/de_de/products/security/identity-services-engine/index.html`
[13]Stand 12.01.2022 `https://bluecatnetworks.com/de/`

	Nautobot mit Golden Configuration	Kraken
Normalisierung	Ja	Ja
Aufbereitung aus Externen Quellen	Nein	Ja
Konfliktlösung (Compliance)	Ja	Ja
Automatische Konfliktlösung	Nein	Ja
Open-Source	Ja	Nein
Quelle für Templates	Ja	Ja
Assurance	Nein	Nein

Quelle für Template Das Tool versteht sich als Quelle zur Netzwerkautomatisierung.

Assurance Das Produkt kann helfen, Fehler auf dem Netzwerk zu debuggen.

2.2. Interviews

Im Zuge dieser Arbeit wurden vier Schweizer Firmen zur Netzwerkautomatisierung befragt. Konkret wurden die Firmen nach ihren Workflows beim Erfassen und Ändern von Netzwerkgeräten gefragt.

2.2.1. Methodik

Es wurde nach dem User Centered Design Thinking vorgegangen. Das ermöglicht es herauszufinden, was die Probleme der Netzwerkadmins aus Sicht des Users sind.

Zuerst wurde eine Vision von Kraken entwickelt, siehe Anhang F.2, so wie die Autoren Kraken in der Zukunft sehen. Mithilfe dieser Vision wurde dann ein Fragebogen erstellt. Ziel war es möglichst wenige hypothetische Fragen zu stellen, sondern den Fokus daraufzulegen viel über die derzeit involvierten Prozess, ihre Pain-Points, die Umsystem, und den derzeitigen Stand der Automatisierung zu erfahren. Die Fragen sollten ein Gemisch aus offenen und geschlossenen Fragen sein. Grundlage waren hier das Wissen aus dem Modul *Human Centered Interface Design* an der OST.

Mit den Antworten der Interviewten sollte es dann möglich sein, ein generelles Verständnis für die Bedürfnisse eines Engineers und einer Firma zu bekommen. Zudem soll aus den Antworten ermittelt werden, welchen Fokus diese Arbeit bei der Weiterentwicklung von Kraken setzt und welche Anforderungen die Firmen an eine Applikation wie Kraken haben.

	Migros	Netrics	Datapark	Init7
Network Devices	20k bis 40k	200 bis 300	10 bis 20	500 bis 1000
WAN	Ja	Ja	Ja	Ja
LAN	Ja	Nein	Nein	Nein
Haupt-Hersteller	Cisco	-	Juniper	Juniper

Die Firmen wurden ausgewählt, da sie verschiedene Grössen von Netzwerk-affinen Firmen repräsentieren.

Die vier Interviewten repräsentieren jedoch nur einen kleinen Teil des Marktes und sind somit nicht repräsentativ für den gesamten Markt.

2.2.2. Firmenspezifische Analyse

Die vier Firmen im Vergleich:

Die Analyse der Migros ist am ausführlichsten, da die Infrastruktur dort auch die grösste Menge an Geräten, involvierten Umsystemen und Workflows hat. Der Fall der Migros kann ausserdem als Inspiration für Kraken als solches angesehen werden.

2.2.3. Migros

Migros in Kürze: Die Migros ist einer der grössten Detailhändler in der Schweiz, mit einer grossen Menge an Filialen und Shoppingcentern. Sie ist als Genossenschaft organisiert und als solche auch noch in Regionen unterteilt. Die Genossenschaften sind dem Migros-Genossenschafts-Bund untergeordnet, die einzelnen Regionen sind jedoch relativ unabhängig. Die Anzahl der betreuten Netzwerkgeräte summiert sich auf ca. 20'000 bis 30'000[14]. Es werden sowohl WAN, als auch LAN Geräte verwaltet. Sie verwalten Router und Switches aber auch WLAN Access Points in den einzelnen Läden.

Die Migros befindet sich gerade im Umbau zu einem Netzwerk mit mehr Automatisierung. Sie unterscheiden zwischen dem Migros-Netz 2.0 und dem Migros-Netz 3.0. Für das neue Netzwerk sind gewisse Dinge automatisiert. Zum Beispiel hat die Migros ein neues Web-Portal, welches sie derzeit testen. Das Portal baut auf Django auf und kann verschiedene Daten mit externen Skripts heranziehen und daraus eine Konfiguration generieren.

[14]Stand 12.01.2022 `SieheTranskriptionimAnhang`

Es wurden verschiedene Workflows im Interview beschrieben, welche hier kurz zusammengefasst werden.

Workflow Geräte Konfigurieren Mittels DHCP Option 43 werden die Geräte erkannt, und so wird ermittelt auf welcher Site sich das Gerät befindet. Danach werden IPs, VLANs, Hostnamen etc. zusammengezogen. Dann wird eine Initial-Konfiguration über das DNA Center gemacht, welches IP-Adresse und Hostnamen weitergibt, damit das Gerät danach erreicht werden kann. Diese Konfiguration wird mit RESTCONF übermittelt und basiert somit auf einem YANG-Model. Somit existiert eine einfach strukturierte Basis für die Konfiguration in einem JSON-File. Diese Konfiguration wird dann mit ihrem Tool, welches die Initial-Konfiguration herunterlädt, mit einer *on-the-fly* erstelltem Soll-Konfiguration verglichen. Dabei wird die Soll-Konfiguration komplett übernommen und auf das Gerät geschrieben. Diese Soll-Konfiguration wird in einem Git-Repository gespeichert, damit die Versionierung sichergestellt ist. So kann auch später überprüft werden, ob das Gerät immer noch so konfiguriert ist, wie es erwartet ist.

Interessant ist, dass WLAN in einem neuen Laden benötigt wird, bevor dessen Abdeckung ausgemessen wird. Die Abdeckung wird erst gemessen, wenn der Laden schon eingerichtet ist, da sie sich mit der Einrichtung verändern kann.

Verwendete Netzwerk Technologien und Daten Aus dem Interview entnommene Stichworte zu Technologien und Daten.

- Access Points
- Switches
- Router
- Operation System (OS)
  - Subito Router - von der Migros verwaltet
  - Swisscom Router - von der Swisscom verwaltet
- DHCP
- IP-Adressen
- IP-Range
- VLAN
- Virtual Rooting and Forewarding (VRF)
- Domain Name System (DNS)

- Firewalls (in sehr keinen Mengen)
- Hostnamen
  - Sind im Moment generiert nach Unternehmen, Stadt, Strasse.
- Standort
  - Ortschaft
  - Adresse/Strasse
  - Gebäude
  - Stockwerk
- Tenant/Genossenschaftsfirma

Verwendete Umsysteme:

- Cisco DNA Center
- Selbst geschriebenes Web-Portal zur Konfiguration von Geräten
- Git Repository für die Speicherung der Konfiguration
- Filial-DB
- Configuration Management Data Base (CMDB) Helpline für Assets
- Monitoring (PRTG)
- Bluecat (IPAM)
- Cisco ISE (RADIUS)

Die IP-Ranges werden derzeit im Bluecat[15] verwaltet und von Hand nach einem gewissen Schema vergeben. Mit dem *Filialenfinder* besitzt die Migros ein CMDB/Filialen-Repository/Inventory. Die Problematik mit diesem Filialen-Repository ist, dass zum Beispiel ein Tivoli[16] keine Adresse besitzt, sondern dort einfach *Tivoli* steht.

Da die Migros eine Genossenschaft ist, verwalten sie zum Beispiel auch die Klubschule und Fitnesszentren. Dabei kann diese Site z.B. spezifische Namen und SSIDs als Anpassung angeben. Die Firmen können auch eigene IT-Lösungen beschaffen, die sie dann auf dem Migros-Netz laufen lassen. Dabei wird ein Netz gebaut und mit einer Firewall geschützt. Die Hardware ist standardisiert, aber die Konfigurationen widerspiegeln die Unternehmens-DNA des Ladenkonstrukts.

[15] Stand 12.01.2022 `https://bluecatnetworks.com/`
[16] Shoppi Tivoli in Spreitenbach AG, grösstes Einkaufszentrum der Schweiz

Manueller Konfigurationsgenerator Migros besitzt derzeit einen Konfigurationsgenerator. Dieser ermöglicht es On-Site Mitarbeitern Router und Switches zu konfigurieren. Der Konfigurationsgenerator besitzt eine Maske, bei der das OS für den Switch und das Unternehmen eingetragen werden müssen. Die VLANs werden anhand des ausgewählten Unternehmens generiert. Der On-Site Mitarbeiter muss dann aber noch spezifizieren, welche VLANs dieses Unternehmens er gerne möchte. Je nach Laden sind das andere, da die VLANs auch auf den Features des Ladens basieren. Die IP-Range des VLANs wird nur als Bezeichner verwendet.

Der Input sieht folgendermassen aus:

- OS
- Switch
- Unternehmen
- Ortschaft
- Strasse
- Gebäude
- Stockwerk
- Angabe wo sich das Management befindet

Als Output des Konfigurationsgenerators bekommt man eine *.rar*-Datei mit der Konfiguration, welche dann von einem Engineer auf die Switch kopiert wird.

Switch Umsysteme, die verwendet werden müssen im Migros Netz 2.0:

- Bluecat (IPAM, DNS, DHCP)
- Seite zur Konfiggeneration
- CISCO ISE

Access Point Das Gerät muss in Umsystemen eingetragen werden. Es muss im RADIUS-Server eingetragen werden, damit es RADIUS-Requests abschicken darf. Zudem muss es in einem Management System wie Cisco Prime eingetragen werden. Den DNS-Namen muss man auch noch im IPAM eintragen. Sie haben das Eintragen in den Umsystemen noch nicht automatisiert, weil man dann *add*, *remove* und *change* Operationen implementieren müsste, die mit den Umsystemen interagieren. Das heisst beispielsweise, dass wenn ein Access-Point entfernt wird, muss der aus all diesen Systemen wieder gelöscht werden, oder wenn einer eingetragen wird muss der DNS-Name im IPAM eingetragen werden.

Inkonsistenzen Es kann vorkommen, dass Geräte nicht in allen Umsystemen eingetragen sind. Genannt wurden da die Umsysteme Monitoring, CMDB und DNS/IPAM. Der Grund weswegen hier noch nichts automatisiert worden ist, ist, dass bei einem Update der Software der Umsysteme häufig die Schnittstellen ändern. Zum Beispiel ändern Spalten in einer CSV-Datei die Namen. Aus diesem Grund müsste das Automatisierungsprogramm häufig angepasst werden und sie hatten bis jetzt nicht die Ressourcen dieses Problem anzugehen. Das Bluecat können sie seit 1.5 Jahren sauber verwenden und bei Prime verwenden sie den CSV Import und den CSV Export, welcher häufiger ändern kann.

Inkonsistenzen, welche noch erwähnt wurden, sind, dass ein altes Gerät in der falschen Domäne ist. Ausserdem kann es zu inkonsistenten Namen kommen, falls mehrere Unternehmen an einem Standort vertreten sind.

Derzeit behandelt Migros die Konfigurationen im Git als die einzige Wahrheit. Das soll aber ändern. Wie genau die Lösung in Zukunft aussehen soll, da gehen die Ideen bei der Migros auseinander. Es wurde bestimmt, dass Cisco ISE als Quelle der Wahrheit für Geräte dient. Der Grund ist, wenn das Gerät nicht im ISE eingetragen ist, darf es sich gar nicht mit dem Netzwerk verbinden und kann gar nicht angesprochen werden. Ein weiterer Grund ist, dass Cisco Prime nur Enterprise-Geräte unterstützt. Ausserdem kam es bei Cisco Prime wegen der Anzahl Geräte von Migros zu einem Performanzproblem, welches auf der anderen Seite auch mit einem Oracle Lizenzproblem begründet werden kann. Grundsätzlich wird jene Quelle als autoritativ definiert, wo die Konsequenz eines Fehlers am grössten ist.

Im Moment arbeiten die automatisierten Tasks mit dem Hostnamen, der wie aufgeführt, aus Unternehmen, Stadt, und Strasse besteht. Weil aber manchmal Unternehmen dazugekommen und manche wieder gegangen sind, stimmen die Hostnamen im Prime nicht immer, oder es fehlen Geräte. Aus diesem Grund wollen sie in Zukunft nur noch den Standort und nicht mehr das Unternehmen in den Hostnamen integrieren. Am besten wäre jedoch eine ID oder eine Seriennummer zur Identifikation. Für die On-Site-Leute sind

aber menschenlesbare Attributes wichtig, damit sie vom Namen Informationen herleiten können. Im Moment bemerken sie nur bei der Verrechnung oder beim Troubleshooting, wenn ein Gerät in einem Umsystem fehlt. Dort ist das aufwendigste das Beheben der Fehler, da jedes Gerät einzeln angeschaut werden muss.

Um bei der Konfiguration in Zukunft Inkonsistenzen zu vermeiden, haben sie sich auch schon überlegt, das Gerätekonfiguration jeden Abend mit der gespeicherten Konfiguration zu überschreiben.

Fazit Migros verfolgt mit Ihren zwei Konfigurationsgeneratoren ein ähnliches Ziel wie *Karken*. Ein Unterschied ist, dass die Konfigurationsgeneratoren der Migros sehr auf die Bedürfnisse und Umstände der Migros ausgerichtet sind. Ein anderer ist, dass Kraken davon ausgeht, dass Informationen in den Quellen korrekt erfasst wurden und nur noch miteinander verknüpft werden müssen. Bei der Migros ist es aber so, dass die Daten zwar in den Umsystemen vorhanden sind, aber zum Beispiel nur Menschenlesbar, oder es wird mehr Kontext benötigt. Immer dann, wenn es Verwechslungen geben kann, wird mehr Kontext benötigt, zum Beispiel, weil es an einem Standort mehrere Filialen gibt.

Die Migros ist auch laut eigenen Angaben eine Sammlung von Ausnahmefällen. Die Automatisierungsbestrebungen werden durch den Föderalismus und die Inkonsistenz erfasster Daten verlangsamt. Durch den Zusammenschluss aus mehreren Unternehmen haben einerseits viele Engineers direkt Zugriff auf die Geräte und andererseits hat jedes Unternehmen wieder eigene Konfigurationswünsche. Verantwortlich dafür, dass VLANs nicht einfach aus Bluecat geholt werden können, ist, dass es kein eigenes Feld zur Identifikation gibt. Die Identifikation über den Namen ist für Menschen einfach, aber durch die Inkonsistenz nicht durch Maschinen machbar. Ein Namens-Schema, welches maschinenlesbar wäre und in solchen Situationen helfen könnte, existiert häufig nicht.

Damit die Automatisierung weiter vorangetrieben werden kann, müssen zuerst die Daten in den Quellen bereinigt werden.

2.2.4. Init7

Init7[17] in Kürze: Init7 ist ein ISP aus der Schweiz. Init7 bietet sowohl Business to Client (B2C) Internet Service als auch Business to Business (B2B) Internet Service an. Ausserdem betreibt Init7 einen internationalen IP-Backbone.

Init7 besitzt derzeit noch ein sehr auf Textfiles ausgerichtetes Informationsmanagement-System. Dies ist derzeit im Umbruch und soll durch Netbox als SoT ersetzt werden. Konkret solle die Konfiguration in Zukunft via Templates geschehen, die generierten

[17] Stand 12.01.2022 `https://www.init7.net/en/`

Konfigurationen sollen kurzfristig via Copy-and-paste auf die Geräte kopiert werden und langfristig via TFTP.

Eingesetzte Technologien

- Ansible
- CRM
- Überwachungstool
- Wiki
- Netbox
- CheckMK[18]

Workflow Von Init7 wurde ein Workflow zur Verfügung gestellt. Das ist der Workflow für den Fiber7 Business-Anschluss:

1. Routername definieren: %routername%
2. Art des Services:
3. Standard-Access
   a) LAN IPv4-Subnet heraussuchen und im DNS den PTR für Router-IP hinterlegen und im CRM hinterlegen
   b) LAN IPv6-Prefix heraussuchen und im CRM hinterlegen
   c) Dem Kunden zugewiesene IPv4 & IPv6 Netze bei RIPE eintragen Wiki-Anleitung
   d) Nächste freie IPv4-WAN-IP heraussuchen und im DNS den PTR hinterlegen und im CRM hinterlegen
   e) Nächste freie IPv6-WAN-IP heraussuchen und im DNS den PTR hinterlegen und im CRM hinterlegen
   f) Forward-Einträge (A & AAAA) auf WAN-IPs / bei Backup-DSL auf LAN-IP eintragen
   g) Switchport/Interface auf dem Fiber7-Switch konfigurieren
   h) Routing-Einträge auf dem Fiber7-Switch für IPv4&IPv6 erstellen

[18]Stand 12.01.2022 `https://checkmk.com/`

i) BGP-Config auf Core-Routern r2zrh1.edge (bei DSL-Backup)

4. eBGP: %kunden-asn%
   a) Neues VLAN durch Fiber7-Ring zum nächsten Core-Router durchtaggen.
   b) VLAN auf Fiber7-Switch auf Kunden-Interface untaggen
   c) Perimeter-IPv4&IPv6-Netz heraussuchen und im Netbox sowie CRM hinterlegen
   d) Perimeter-IPs im DNS eintragen
   e) Config des Virtual-Interface auf dem Core-Router erstellen
   f) Prefix-Listen (IPv4&IPv6) definieren
   g) BGP-Config erstellen inkl. „no shut“
   h) AS-SET updaten
5. Cacti-User erstellen und im CRM hinterlegen
6. Gerät im Netbox erfassen
7. CPE konfigurieren
   a) Config-Script durchspielen
      i. PPPoE und BGP-Config einfügen (bei DSL-Backup)
      ii. VLAN-Anpassungen (bei eBGP)
   b) CPE mit P-Touch anschreiben (Routername und Eigentum Init7)
   c) CPE durch Mitarbeiter kontrollieren lassen
   d) CPE wieder einpacken
      i. Glasfaserkabel
      ii. SFP-Modul
      iii. Rackschraubenset
      iv. 2 x Stromkabel
      v. Datenblatt (aus CRM)
      vi. Installationsanleitung (für richtigen Plug)
      vii. DSL-Bridge (falls DSL-Backup)
   e) Sales Order im Odoo bearbeiten

f) Dem Versand übergeben: %datum%

8. Router im Icinga erstellen und gerade acknowledgen (Comment: „new“)
9. DSL-Backup (Routername.bkp) im Icinga erstellen und gerade acknowledgen (Comment: „new“)
10. Datenblatt Datenblatt und Meldung, dass der Router verschickt wird per Mail an Kunde
11. YT-Issue an Sales assignen (bis Leitungen parat sind)

NetEng Inbetriebnahme (Wiki-Anleitung) Router ist online: %datum%

1. Device im Cacti erfassen: WAN-IP (LAN bei DSL-Backup) & Router-Name
2. Device im Icinga checken, ob UP
3. PPPoE-Client und BGP-Config überprüfen (bei DSL-Backup)
4. RFS im CRM und Odoo setzen

Im Workflow ist zu sehen, dass die Informationen durch den ausführenden Engineer in die einzelnen Quellen eingetragen werden.

2.2.5. Netrics

Netrics in Kürze: Netrics ist ein B2B Backbone Betreiber in der Schweiz. Sie verwalten vor allem die eigene Infrastruktur.

Netrics verwendet Netbox für die Verwaltung der Geräte im Netzwerk. Derzeit sind laut Angaben aus dem Interview 80% der benötigten Informationen in Netbox gespeichert. Ausgenommen davon sind zum Beispiel Crossconnects zwischen Datacenters und Verbindungen zwischen Geräten. Von Netrics werden vor allem Subnetze in den privaten Bereichen verwaltet.

Automatisierung ist derzeit kein explizites Thema bei Netrics. Es wird Ansible verwendet, wenn zum Beispiel eine Änderung auf vielen Geräten ausgerollt werden soll. Die Konfiguration dazu wird aber manuell erstellt. Der Automatisierungsdruck äussert sich durch das Bedürfnis weniger Zeit für Wünsche von Kunden zu benötigen.

Für die Konfiguration von neuen Geräten wird entweder ein Referenzgerät oder ein Template verwendet. Unter Template versteht Netrics Textstücke welche vom Engineer angepasst werden müssen.

Als Pain-Point wurde genannt, dass der derzeitige Workflow zu statisch sei. Es sei mühsam die Informationen zusammenzusuchen, bis man weiss, wo man die Änderungen durchführen kann.

Die Informationen kommen via selbstgeschriebenes Inventar-Tool ins Monitoring.

Die Versionierung der Konfiguration der Geräte geschieht mit RANCID.

2.2.6. Datapark

Datapark in Kürze: Datapark bietet Internet-Service für Kabelnetzbetreiber in der Region Wil, SG an. Datapark ist im B2B Bereich tätig.

Das Spezielle an Datapark ist, sie besitzen ein Tool namens Service Access Manager (SAM). SAM wurde von Datapark selber entwickelt und übernimmt die automatische Konfiguration von DSLAMs. SAM bietet den Kunden von Datapark ein Interface, mit dem sie die DSLAMs selbst konfigurieren können. SAM arbeitet derzeit noch mit Simple Network Management Protocoll (SNMP), dies hat den Nachteil, dass ein SNMP-Walk etwas dauern kann. Deshalb wird derzeit eine Ablösung von SAM durch ein NETCONF/YANG basiertes Tool in Betracht gezogen.

Im Backbone selbst geschehen nicht viele Änderungen. Mit Ansible wurde letztes Jahr ein Inventar der Geräte im Backbone erstellt. Dieses ist jedoch statisch und muss manuell aktualisiert werden. Es existiert eine YAML-Datei pro Gerät. Derzeit arbeitet nur ein Engineer mit Ansible, dadurch ist es dann auch seine Aufgabe die Änderungen im Ansible nachzutragen.

Im Access-Layer soll in Zukunft die Konfiguration auch via NETCONF/glsYANG geschehen.

Netbox wird für die IP-Verwaltung verwendet.

Management-IPs besitzen derzeit keinen DNS-Forward-Eintrag.

Es existiert ein Inventar basieren auf Excel Sheets, welches sich auf die Abschreibungen fokussiert, aber nicht bei der Konfiguration von Geräten involviert ist.

2.2.7. Use Cases aus den Interviews

Aus den Interviews haben sich einige für Kraken interessante Use Cases ergeben, welche zum Teil über die bekannten Funktionalitäten hinausgehen. Die Use Cases sind nachfolgend aufgeführt sind.

2.2.7.1. Akteure

- Migros
- Init7
- Netrics
- Datapark

2.2.7.2. Usecases

UC1 Fehlende Devices Als Migros will ich wissen, welche Devices nicht in allen Umsystemen vorhanden sind. Ich will diese Inkonsistenz dann beheben können.

UC2 Add Device Als Migros will ich ein neues Device im Netzwerk erfassen. Damit ich es nicht in jedes Umsystem von Hand eintragen muss, will ich, dass Kraken das für mich übernimmt.

UC3 Change Device Als Migros will ich das Changes die ich in einem Umsystem mache in die Anderen Umsysteme übernommen werden.

UC4 Remove Device Als Migros will ich ein Device in allen Umsystemen löschen können.

UC5 Basis für Konfigurationsgenerator Als Migros will ich meinen Konfigurationsgenerator mit Kraken befüllen, da Kraken alle nötigen Daten für mich sammelt und ich somit im Generator nur eine Schnittstelle programmieren muss, und somit ändernde Schnittstellen nur an einem Ort anpassen muss. Um auf die Daten von Kraken zugreifen zu können verwende ich Kraken als Python Modul. Als Datapark möchte in Zukunft die DSLAMs mit NETCONF/YANG konfigurieren. Als Netrics möchte ich, dass Konfigurationen effizienter durchgeführt werden.

UC6 Zero Touch Provisioning Als Init 7 möchte ich meine Geräte mit möglichst wenig Aufwand provisionieren.

2.3. Erkenntnisse aus der Marktanalyse

Die Interviews und die Marktanalyse haben gezeigt, dass das Bedürfnis für ein Tool wie Kraken existiert. Mit *Nautobot* und dem *Golden Configuration* Plugin existiert auch bereits ein Tool mit ähnlichen Zielen. Jedoch sieht sich *Nautobot* im Gegensatz zu *Kraken* als einzige Quelle im Netzwerk. Das heisst, es ist nicht angedacht, dass die Informationen aus Dritt-Quellen stammen.

Die Interviews haben aber auch gezeigt wie unterschiedlich die Workflows sind und wie schwierig es ist einen gemeinsamen Nenner zu finden, was Features angeht. Dort ist auch die Stärke von *Netbox* zu erkennen. In dem Netbox sich als 80% Lösung positioniert und die restlichen 20% dem Kunden überlässt, kann die Komplexität der Applikation selbst übersichtlich gehalten werden.

Es hat sich auch gezeigt, dass die Annahme von Kraken, dass die Daten in den Quellen existieren und nur noch zusammengesetzt werden müssen, Workflowmässig derzeit am ehesten auf die Migros zutrifft.

Die Firmen erhoffen sich, durch den Einsatz eines Tools wie Kraken, einen Effizienzgewinn. Es muss von den Engineers aber auch angenommen und akzeptiert werden. Deshalb ist es wichtig, dass egal was Kraken ausgibt, dies vom Engineer unabhängig verifiziert werden kann.

Der Knackpunkt von Kraken wird sein, möglichst viele Anwendungsfälle in Firmen anzusprechen aber trotzdem die Komplexität im Zaum zu halten. Ausserdem gibt es unzählige Umsysteme welche angebunden werden wollen. Dort ist die Gefahr, dass die Entwicklung immer etwas hinterherhinkt. Ausserdem gibt es unzählige Umsysteme welche angebunden werden wollen. Dort ist die Gefahr, dass die Entwicklung immer etwas hinterherhinkt, denn jedes dieser Umsysteme hat seine Eigenheiten. Sei es, dass nur lesend darauf zugegriffen werden kann, wie zum Beispiel im DNA Center.

2.3.1. Prozesse in Firmen unterscheiden sich

Netzwerke sind historisch gewachsen. Mit ihnen sind die Tools und Prozesse rundherum mitgewachsen. Netzwerke bauen im Unterschied zu Servern auf Protokollen und nicht auf Produkten auf. In Bereichen, in denen sich Protokolle durchgesetzt haben, ist die Weiterentwicklung meist langsamer, es fehlt Innovationsdruck, da die bestehende Lösung schlicht gut genug für die Bedürfnisse der Marktteilnehmer ist.

In Firmen in welchen das Netzwerk sich kaum verändert, ist der Automatisierungsdruck entsprechend kleiner. Es muss höchstens hie und da überprüft werden, ob die Konfiguration noch konsistent ist. Es muss auch noch berücksichtigt werden, warum ein Netzwerk

wächst. Ein ISP der mehr Racks hinzufügt ist nicht zu vergleichen mit einem ISP der Router bei Kunden installiert. Dort spielt es auch wieder eine Rolle wie viele Drittparteien involviert sind und nach welchem Standard die Konfiguration ist. Wie viele Geräte befinden sich physisch am selben Ort, und gehören zur selben Einheit?

Bestehende kleinster gemeinsamer Nenner oder volles Potenzial Setzt eine Firma eine Drittsoftware ein muss sie sich die Frage stellen, ob sie nur ein Subset der Software nutzen will oder der, ob sie das volle Potenzial der Software ausschöpfen will. Nutzt sie das volle Potenzial, dann ist es schwierig das Programm gegen einen Mitbewerber zu tauschen. Nutzen sie nur ein Teil der Software, kann die Software leichter ausgetauscht werden, dafür liegt aber wiederum viel Potenzial brach.

Der Netzwerk Markt ist fragmentiert/Jedes Netzwerk ist eine Schneeflocke Netzwerkgerätehersteller wollen alle das Rundumsorglospaket bieten. Der Netzwerkgeräte Markt ist ein Markt mit für IT Verhältnisse vielen Playern. Heterogene Netze mit mehreren Herstellern sind also keine Seltenheit. Einerseits müssen die Marktteilnehmer vergleichbare Features bieten, damit ein Wechsel in Betracht gezogen werden kann, auf der anderen Seite wird mit Alleinstellungsmerkmalen geworben um einen Wechsel zu einem Konkurrenten zu verhindern. Das Resultat ist, dass gerade im Automatisierungsbereich eine grosse Landschaft an Tools besteht, die wiederum nur mit einer kleinen Anzahl an Geräten kompatibel ist. Auch Thomas Fritz erwähnt dies kurz im Interview im Anhang. Dies macht es schwieriger auf dieser Ebene zu Automatisieren. Im Umkehrschluss erklärt dies auch wieso Overlay-Netze so starken Rückenwind erhalten hat. Es abstrahiert die verschiedenen Hardware-Limitationen weg, auf eine gemeinsame Software. Firmen welche hier nur auf Open-Source setzen sind hier klar im Vorteil.

Jedes Firmennetzwerk sieht wiederum anders aus, besteht aus anderen Komponenten und es werden andere Konzepte angewandt. Lösungen welche für Netzwerk A entwickelt wurden helfen nicht unbedingt Netzwerk B[8].

Firmenkultur Eine Firmenkultur muss willig sein zu automatisieren. Dabei spielt es durchaus eine Rolle, ob dies nur ein Lippenbekenntnis ist oder auch so gelebt wird. Es ist wichtig, dass die Philosophie sowohl von den Mitarbeitern als auch vom Management getragen wird.

Etwas zu automatisieren bedeutet auch, einen Schritt zurück zu machen und das Gesamtbild anzuschauen. Es bedeutet Iterativ vorhandene Prozesse verbessern. Grössere Firmen sind meist sehr prozessorientiert. Das ist wünschenswert, kann aber auch dazu führen, dass Prozesse einfach blind ausgeführt werden.

In dem Sinne sind Firmenprozesse wie Algorithmen. Wie Software-Code müssen Prozesse gepflegt werden. Im Software-Engineering würde man das Refactoring nennen. Teile oder des Gesamte müssen infrage gestellt werden und es muss versucht werden den Prozess zu vereinfachen. Vereinfachen heisst in diesem Zusammenhang: Schritte sind in sich abgeschlossen, lassen sich ohne viel Kontextwechsel ausführen und die einzelnen Schritte lassen sich verifizieren.

Improving what is already there Eine weitere Erklärung wieso das NETCONF/YANG eine eher langsame Adoption hat. Netzwerkteams verfeinern, was sie bereits einsetzen und wollen nicht etwas einsetzen, was eine komplett andere Denkweise erfordert. Sich komplett auf die neue Denkweise zu einzulassen, können sich Teams und Firmen schlicht nicht leisten. Das führt dazu, dass Software, welche sich an Netzwerkteams richtet, von Software Engineers mit wenig Bezug zur Arbeit als Netzwerk-Engineer geschrieben wird.

3. Softwaredokumentation

3.1. Projektspezifische Begriffe

Quelle Ein Ort, an dem Informationen zu einem Objekt gespeichert sind.

Inventar Mit Inventar wird eine Sammlung von Informationen zu einem oder mehreren Objekten bezeichnet. In diesem Kontext zählen auch auf dem Gerät vorhandene Informationen als Inventar.

Service Ein Service ist etwas was auf einem Netzwerkgerät konfiguriert werden kann. Ein Service kann auf anderen Services aufbauen.

Workflow Ein Workflow ist ein Prozess welcher durchgemacht werden muss um einen Service aufzuschalten.

Kraken Workflow Ein Prozess innerhalb von Kraken, welcher angestossen wird und je nachdem eine Benutzerinteraktion verlangt. Ein Beispiel ist der Merge von zwei Quellen.

Template In Kraken ist ein Template eine definierte Selektion von Attributen, die auf dem Normalisierungsformat aufbauen.

Inkonsistenz Eine Inkonsistenz besteht, wenn beim Vergleich von Quellen ein Attribut auf einer Entität bei verschiedenen Quellen einen anderen Wert hat.

3.2. Anforderungsspezifikation

Kraken soll Daten aus verschiedenen Quellen mit unterschiedlichen Daten miteinander verknüpfen und bei redundanten Attributen den Wert mit Vorrang übernehmen. Diese zusammengeführten Daten sollen dann zur Weiterverarbeitung durch zum Beispiel einen Konfiggenerator zur Verfügung gestellt werden.

Des Weiteren soll Kraken in der Lage sein, diese aktualisierten Werte, wenn möglich wieder zurück in die Quellen zu schreiben.

Kraken hat somit, wie in Abbildung 3.1 ersichtlich, zwei Ziele. Das Eine ist die Aufbereitung und das Sammeln der Daten aus den Quellen, damit sie für die Netzwerkautomatisierung verwendet werden können. Das Andere ist die Korrektur von Inkonsistenzen und die Verbesserung der Datenqualität in den Quellen.

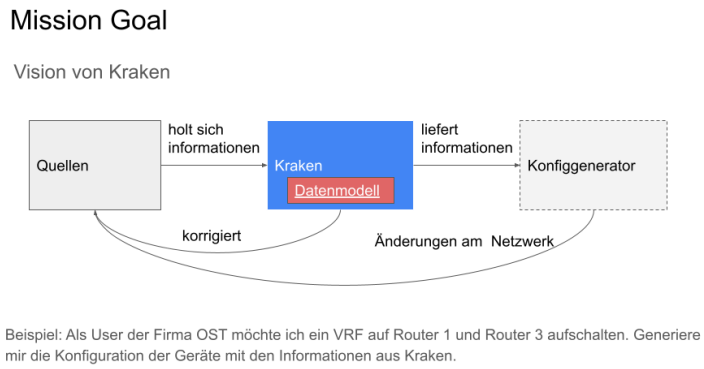


Abbildung 3.1.: Mission Goal von Kraken

3.2.1. Use Cases

In dieser Sektion werden generelle Use Cases erfasst. Firmenspezifische Use Cases werden im Teil Marktanalyse 2 identifiziert und erfasst. Die Use Cases lehnen sich an die Use Cases der Vorgängerarbeit an[3, S. 25]. Zudem werden nur Use Cases behandelt, die für die CLI relevant sind, da die Webapplikation in dieser Arbeit nicht behandelt wird.

Die Use Cases sind in der Abbildung 3.2 ersichtlich.

3.2.2. Akteure

- User

3.2.3. Brief Usecases

UC1 Einstellungen Als User will ich erfassen können, welche Quellen ich zusammenführen will und soll deren Präzedenz festlegen können. Zudem sollen weitere Einstellungen mit weiteren Use Cases hinzufügt werden.

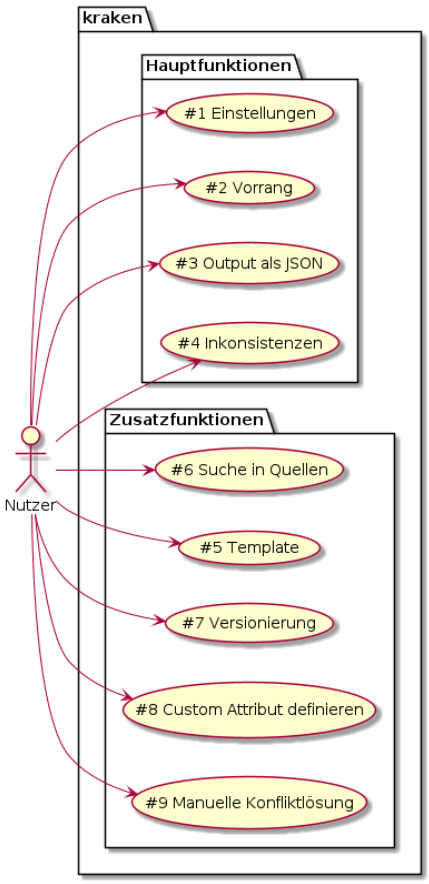


Abbildung 3.2.: Use Case Diagramm von Kraken

UC2 Vorrang Als User will ich den unterschiedlichen Quellen einen unterschiedlichen Vorrang geben, damit Kraken bei Attributen, die in mehreren Quellen vorkommen, weiss welchen Wert übernommen werden soll. Der Vorrang kann auch auf Level Attribut vergeben werden. Der Vorrang kann als Präzedenz in einem Merge-Template definiert werden.

UC3 Output als JSON Als User will ich eine Datei angeben können, in die der Output als JSON-Format geschrieben wird. Wenn ich keine Datei angebe, will ich den Output auf der Konsole sehen.

UC4 Inkonsistenzen Bei Attributen die in mehreren Quellen vorkommen, will ich beim Zusammenführen eine Überprüfung machen, ob der Wert der Attribute der Gleiche ist. Wenn sich der Wert unterscheidet, will ich, dass das in einer Logdatei aufgeführt wird und der Wert mit Vorrang übernommen wird.

Erweiterung Ich will einen Konfliktlösungs-Kraken-Workflow anstossen um die Inkonsistenzen zu beheben und wenn von der Quelle aus möglich, die Daten in den Umsystemen zu aktualisieren.

UC5 Suche in Quellen Ich will Suchbegriffe definieren können, nach denen, wenn möglich, in den Quellen gesucht wird.

Erweiterung Als User will ich filtern für welche Geräte Daten gesammelt werden. Basierend darauf will ich, dass Kraken diese Filter intelligent an die Umsysteme weiterleitet, damit nur so viele Daten von den Umsystemen abgerufen werden müssen wie nötig. Dabei ist die Suche je nach Attribut komplett anders in der Implementation, da je nach Attribut die Quellen in einer bestimmten Reihenfolge abgefragt und gefiltert werden müssen.

UC6 Template Als User will ich eine vordefinierte Abfrage ausführen, um Daten für eine Gerätekonfiguration abzurufen. Die Daten sollen von verschiedenen heterogenen Quellen abgerufen werden und miteinander verbunden werden. Die Templates sollen als Plugin erfassbar sein.

UC7 Versionierung Ich will nachverfolgen können, welche Änderungen an den Daten vorgenommen worden sind. Der Ouput von Kraken wird zur Konfiguration von Gräten verwendet, deshalb soll dieser Output in Form einer History versioniert werden. Alternativ können anstelle des Outputs auch die normalisierten Daten gespeichert werden.

UC8 Custom Attribut definieren Als User will ich ein Attribut auf einer Klasse des Normalisierungsformats definieren können und will angeben auf welchen Quellen es wie vorhanden ist.

UC9 Manuelle Konfliktlösung Als Alternative zur automatischen Konfliktlösung sollen Inkonsistenzen manuell lösbar sein. Für die manuelle Konfliktlösung gelten die Use Cases der Vorgängerarbeit.

3.2.4. Umsetzungsumfang

Ein neues Datenschema wurde definiert und das soll nun eingebaut werden. Mit diesen neuen Anforderungen an die Applikation muss der Grossteil der bestehenden Logik ausgetauscht werden. Das Klassenmodell soll implementiert werden, das Importieren von verschiedenen Quellen, das Verbinden der Daten, das Mergen der Daten und das Exportieren der gesammelten Daten als JSON-Datei.

Konkret sind da folgende Use Cases involviert:

- UC1 Einstellungen
- UC2 Vorrang
- UC3 Output als JSON
- UC4 Inkonsistenzen

3.2.5. Nicht-funktionale Anforderungen

Bei den Nicht-funktionalen Anforderungen wird auf die der Vorgängerarbeit verwiesen, welche hier, wenn nötig angepasst und ergänzt wurden[3, S. 32].

3.2.5.1. Funktionelle Stabilität

1. **Funktionelle Vollständigkeit** Kraken lässt sich konfigurieren, kann Informationen importieren, mergen und abfragen.
2. **Funktionelle Korrektheit**: Die Informationen sollen von Kraken so gespeichert werden, wie vom Anwender erwartet.
3. **Funktionelle Angemessenheit** Kraken verfügt über die Use Cases, welche es benötigt um zu funktionieren.

3.2.5.2. Performance

1. **Zeitverhalten** Sind die Quellen eingelesen, soll die Abfrage unter 30 Sekunden erfolgen, egal wie gross das Netzwerk ist.

Im Gegensatz zur Vorgängerarbeit wird der Abschnitt „Ressourcenverwendung“ entfernt. Das damals definierte Ziel ist nicht genug messbar.

3.2.5.3. Kompatibilität

1. **Interoperabilität**
   a) Kraken läuft Betriebssystem-Agnostisch
   b) Kraken läuft optional in einem Docker-Container.
   c) Das Normalisierungsformat kann von anderen Applikationen verwendet werden.
   d) Das Normalisierungsformat entspricht einem in der Netzwerkdomäne anerkannten Datenformat.

3.2.5.4. Verwendbarkeit

1. **Erlernbare Angemessenheit** Eine Anleitung zu Kraken erläutert, in welchem Umfeld Kraken eingesetzt werden kann und welchen Zweck die Applikation verfolgt.

2. **Erlernbarkeit** Es soll eine Anleitung zur Verfügung gestellt werden, damit Netzwerkadmins und Entwickler die Anwendung in drei Stunden installiern, konfiurieren und verwenden können. Ein vorkonfigurierter Kraken soll innerhalb von 45 Minuten verwendet werden können.
3. **Bedienbarkeit**
   a) Der Anwender muss wissen, wie man YAML- und JSON-Files schreibt.
   b) Der Anwender muss keine Programmierkenntnisse haben.
4. **Schutz vor Nutzer-induzierten Fehlern**
   a) Kraken Informiert den Anwender über falsche Kraken Einstellungen.
   b) Kraken Informiert den Anwender bei Fehlern im Merge-Template.
   c) Beim Zusammenführen wird der Nutzer über den Ablauf informiert.
   d) Kraken erlaubt eine manuelle Konfliktlösung bei quellsystemübergreifenden Konflikten.

3.2.5.5. Zuverlässigkeit

1. **Fehlertoleranz**
   a) Kraken akzeptiert keine fehlerbehaftete System-Konfiguration.
   b) Kraken kann Daten zusammenführen, selbst wenn kein individuelles Merge-Template definiert ist.

3.2.5.6. Security

Die Webinterfaces sollen über HTTPS gesichert werden können.

3.2.5.7. Wartbarkeit

1. **Modularität**
   a) Kraken soll eine beliebige Anzahl an Quellsystemen anbinden können.
   b) Neue Quellen sollen von einem Nutzer hinzugefügt werden ohne, dass die Architektur von Kraken verstanden werden muss.
   c) Der Normalisierungvorgang ist austauschbar und erweiterbar implementiert.

   d) Bei einem automatischen Aufbereiten von Daten können vom User Präferenzen festgelegt werden, von welchem Sytstem welche Information kommt.

2. **Analysierbarkeit** Kraken ist Closed Source.
3. **Modifizierbarkeit** Ein Anwender soll neue Systeme am Kraken konfigurieren können.
4. **Testbarkeit** Kraken stellt Beispiele zu Verfügung und nutzt Unittests.

3.2.5.8. Übertragbarkeit

1. Installierbarkeit
   a) Kraken lässt sich auf einem System installieren.
   b) Kraken lässt sich auf einem System in einem Docker Container nutzen.

3.3. Design

3.3.1. Data-Engineering

In dieser Arbeit wird die Frage gestellt, wie voneinander abhängige Informationen, die aus mehreren Quellen stammen, verbunden werden müssen, um daraus ein Netzwerk zu konfigurieren.

In den Interviews und der Analyse wurde drei verschiedene Arten von Konflikten erkannt.

Typos In einer Quelle steht *leef1* statt *leaf1*

Divergierende Informationen In Quelle A ist der Host als leaf1 drin, in der Quelle B als leaf12

Daten sind verunreinigt Der Hostname von einem Gerät ist leaf1.a, sollte jedoch leaf1.b sein. Dieser ist aber in allen Umsystemen als leaf1.a eingetragen.

In der Vorgängerarbeit war der Fokus auf *Typos* gelegt. Die bei der Migros angetroffenen Konflikte fallen in die Kategorie Daten sind verunreinigt.

In dieser Arbeit ist der Fokus nun auf Typos und den divergierenden Informationen.

Wir gehen zuerst davon aus das die Informationen so wie sie in den Quellen vorhanden sind stimmen. Wie können diese nun vereinigt werden. Als Beispiel sind hier die

Informationen für ein Device-Template angeführt, welches aus folgenden Informationen besteht:

- Device Name
- Location Name
- Raum-Nummer
- Rack-Nummer
- Zuständiger Verantwortlicher
- Statische Management IP Adresse
- DNS Name der Management IP
- Interface Name der Management IP
- for Interface 1 bis n
  - IP Adresse/Subnetzmaske
  - Adress-Typ
  - VLAN Nr
  - VRF Nr
  - Routing Protokoll
  - Routing Area

Ausserdem seien folgende Quellen gegeben: Source 1: IP Adress Management (z.B. Netbox)

- IP Prefix (Netz/Maske)
- Status. (frei, belegt, reserviert usw.)
- VLAN Nr
- VRF Nr
- Routing Protokoll
- Routing Area

Source 2: Zugewiesene IP Adressen (z.B. Netbox)

- IP Adresse (host)
- Adresstyp: z.B. Host, Primary, Secondary, VRRP, Management Interface etc.

- Interface Name
- Art der Zuweisung (statisch, DHCP, SLAC etc.)
- DNS Name
- Gerätename

Source 3: Segmentierungs-Management (z.B. Netbox)

- Segmentbeschreibung (Text)
- Segment-Typ (VLAN, VRF, MPLS-VPN)
- Segment-Nummer (VLAN Nr, VRF Nr etc.)
- Status (frei, belegt, reserviert)

Source 4: Locations (z.B. Excel)

- Location Name
- Location Adress
- Location Einheit/Firma
- Location Region
- Raum-Nummer
- Rack-Nummer
- Zuständiger Verantwortlicher

Source 5: Inventar

- Devicetyp (z.B: Router / Switches)
- Beschreibung
- Konfig-Template Name
- Inventarnummer
- Gerätename
- Seriennummer
- Raumnummer
- Racknummer
- Management IP

- Anzahl Ports/Interfaces

Es soll die Frage geklärt werden, wie die Quellen kombiniert werden müssen, um an die Informationen zu kommen. Um die Beziehungen der Quellen zu visualisieren, wurde ein Graph gewählt. Informationen die doppelt verwiesen werden, werden jeweils von beiden Quellen verwiesen.

Der Graph ermöglicht uns dann die Beziehungen zwischen den Attributen zu erkennen. Vorausgesetzt ist, dass die verbindenden Attribute einzigartige Werte enthalten.

Die Datenstruktur die in Kraken implementiert wird sieht nicht wie der Graph aus, aber der Graph ermöglicht es uns eine Datenstruktur davon abzuleiten. In Abbildung 3.3

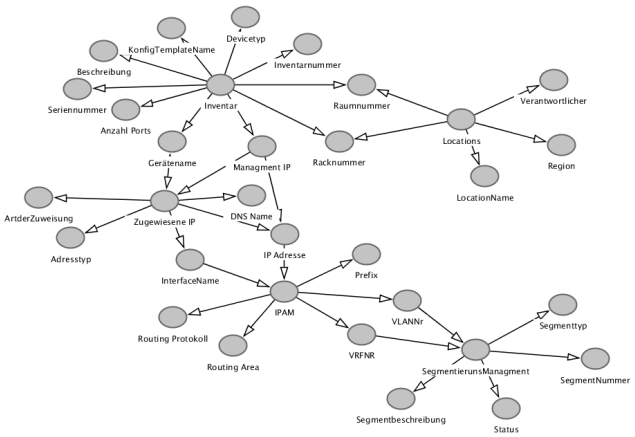


Abbildung 3.3.: Die verschiedenen Quellen als Graph

wurde der Graph visuell aufgearbeitet. Pfeile führen von den Quellen zu den Attributen. Identische Attribute haben mehrere Pfeile auf sich gerichtet.

Device Die Information zum Device-Namen kommen aus der Inventar-Quelle.

Location-Informationen Die Location-Informationen kommen aus dem Inventar.

Um ein Rack eindeutig zu identifizieren, müssen folgende Bedingungen übereinstimmen:

- Location Name
- Location Region
- Raum Nummer
- Rack Nummer

Da Racknummer, Raumnummer, und Location Name keine Eindeutigkeit für sich besitzen, kann es also einen Raum „3“ in verschiedenen Locations geben.

3.3.1.1. Device Information

Die Geräte Information ist bei diesem Template der Einstiegspunkt.

Zuständiger Verantwortlicher Diese Information kommt aus der Location-Source.

Path: Inventar.Gerätenamen » (Locations.LocationName && Inventar.Raumnummer && Inventar.Racknummer) » Locations.Verantwortlicher

Einschränkung: Die Location kann nicht eindeutig zugeordnet werden, da der zum eindeutigen Identifizieren des Racks notwendige Standort in der Location-Liste fehlt.

Statische Management IP Diese Information kommt aus dem Inventar und lässt sich dem Gerät zuordnen.

Path: Inventar.Gerätename » Inventar.ManagementIP

DNS Name der Management IP Für diese Information muss die Management IP in der Quelle der zugewiesenen IPs gesucht werden. Stimmt auch der Gerätenamen überein, enthält der gefundene Eintrag den DNS-Namen.

Path: Inventar.Gerätename » Inventar.ManagementIP » ZugewieseneIPs.DNSName Einschränkung: ZugewieseneIPs.Gerätename muss mit dem Gerätenamen im Inventar übereinstimmen.

Interface Name der Management IP Siehe 3.3.1.1 *DNS Name der Management IP.*

Path: Inventar.Gerätename » Inventar.ManagementIP » ZugewieseneIPs.InterfaceName Einschränkung: ZugewieseneIPs.Gerätename

3.3.1.2. Interface Konfiguration

Mit der Info des Gerätenamens erhalten wir alle konfigurierten Interfaces aus der Liste der zugewiesenen IPs.

Path: Inventar.Gerätename » ZugewieseneIPs.InterfaceName

IP Adresse/Subnetzmaske Da die IP-Adresse bekannt ist, muss noch die Subnetzmaske gefunden werden. Diese lässt sich über die Prefixes finden. Das kleinste vergebene Prefix, in dem unsere IP vergeben ist, ist unsere Subnetzmaske.

Path: Inventar.Gerätename » ZugewieseneIPs.InterfaceName » ZugewieseneIPs.IPAdresse » IPAM.Prefix

Beachten: Mit VRF können IPs und Prefixe mehrfach vergeben werden.

Adress-Typ Der Adresstyp kann über die Zugewiesenen IPs gefunden werden.

Path: Inventar.Gerätename » ZugewieseneIPs.InterfaceName » ZugewieseneIPs.Adresstyp

VLAN Nr Die VLAN-Nummer kann über die Zugewiesenen IPs im IP Adressmanagement gefunden werden.

Path: Inventar.Gerätename » ZugewieseneIPs.InterfaceName » VLANNr

VRF Nr Die VRF-Nummer kann über die zugewiesenen IPs im IP Adressmanagement gefunden werden.

Path: Inventar.Gerätename » ZugewieseneIPs.InterfaceName » ZugewieseneIPs.IPAdresse » IPAM.IPPrefix » VRFNr

Routing Protokoll Das Routing Protokoll kann über die zugewiesenen IPs im IP Adressmanagement gefunden werden.

Path: Inventar.Gerätename » ZugewieseneIPs.InterfaceName » ZugewieseneIPs.IPAdresse » IPAM.IPPrefix » RoutingProtokoll

Routing Area Das Routing Protokoll kann über die zugewiesenen IPs im IP Adressmanagement gefunden werden.

Path: Inventar.Gerätename › ZugewieseneIPs.InterfaceName › ZugewieseneIPs.IPAdresse › IPAM.IPPrefix › RoutingArea

3.3.2. Datenstruktur

3.3.2.1. Datenformat

Um die verschiedenen Quellen und Daten miteinander verknüpfen zu können, wird ein Normalisierungsformat benötigt. So ein Format bringt den Vorteil, dass die Logik und Verwertung der Daten auf nur einem Datenschema basieren kann und bei einer Änderung in der Quelle die Programmlogik nicht angepasst werden muss. Im Kraken 1.0 wurde dieses Normalisierungsformat in einer YAML-Datei definiert[1]. Es umfasst das Device, Interfaces und Subinterfaces.

```
DEVICE_NAME:
  name:
  model:
  manufacturer:
  device_role:
  interfaces:
    INTERFACE_NAME:
      dns_name:
      config:
        description:
        enabled:
        name:
      subinterfaces:
        0:
          index:
          config:
            index:
            name:
            description:
            enabled:
          ip_type:
          IP_TYPE:
            addresses:
              IP_ADDRESS:
                ip:
                config:
                  ip:
```

[1] Stand 12.01.2022 https://gitlab.ost.ch/ins-stud/ba21-nettowel-kraken/-/blob/d8f12aaa341613f12ac3c45ee27d0020ef1bbaba/kraken/config/sample_normalized_structure.yml

```
            prefix-length:
      record_type:
      ip_address:
      sub_interface:
      ethernet:
        config:
          auto-negotiate:
```

Listing 3.1: sample_normalized_structure.yml

Basierend auf dieser Struktur normalisieren die Konnektoren die Daten. Dies erlaubt einen einfachen Umgang mit der Vergleichs-Library DeepDiff. Zudem ist eine Dict-Struktur sehr dynamisch und deshalb schnell in der Implementation. Auf der anderen Seite ist eine Definition via YAML wegen der verschachtelten Struktur schwieriger anzupassen, da es sich dabei eigentlich um ein einziges grosses Objekt handelt. Um die Applikation einfacher erweiterbar zu machen, haben wir uns entschieden das Normalisierungsformat mit einer Struktur aus Python-Klassen und Objekten umzusetzen. Dies hat neben der einfacheren Erweiterbarkeit der Datenstruktur auch den Vorteil, dass man wieder besser objektorientiert programmieren kann. Zum Beispiel könnten auch die Daten mit einem Object-Relational Mapping (ORM) in einer Datenbank speichern. Somit profitiert man auch von einer besseren funktionellen Erweiterbarkeit. Zudem ist es bei einem Klassenschema einfacher, Redundanzen zu vermeiden.

3.3.2.2. Datenschema

Aus der Analyse der verschiedenen Quellen wurde ein Klassendiagramm erstellt, das alle Informationen in ihren Beziehungen zueinander abbildet.

Abbildung 3.4, Objektdiagramm des Datenschemas

Bei der Erstellung des Schemas wurde darauf geachtet, dass eine Balance in der Komplexität gefunden wird. Einerseits sollen alle Information abgebildet werden, andererseits es soll immer noch gut verständlich sein. Das heisst, es soll nachvollziehbar dargestellt sein, wie Daten für ein Template zusammengezogen werden können und wie Daten aus den Quellen in das Normalisierungsformat gelangen können.

3.3.3. Identifizieren von Entitäten mit wenig Informationen

Dieser Teil beschäftigt sich mit Problemen der Art *Informationen divergieren.*

In der Praxis wurden der Simmilarty Score und der Locality Sensitive Hash evaluiert.

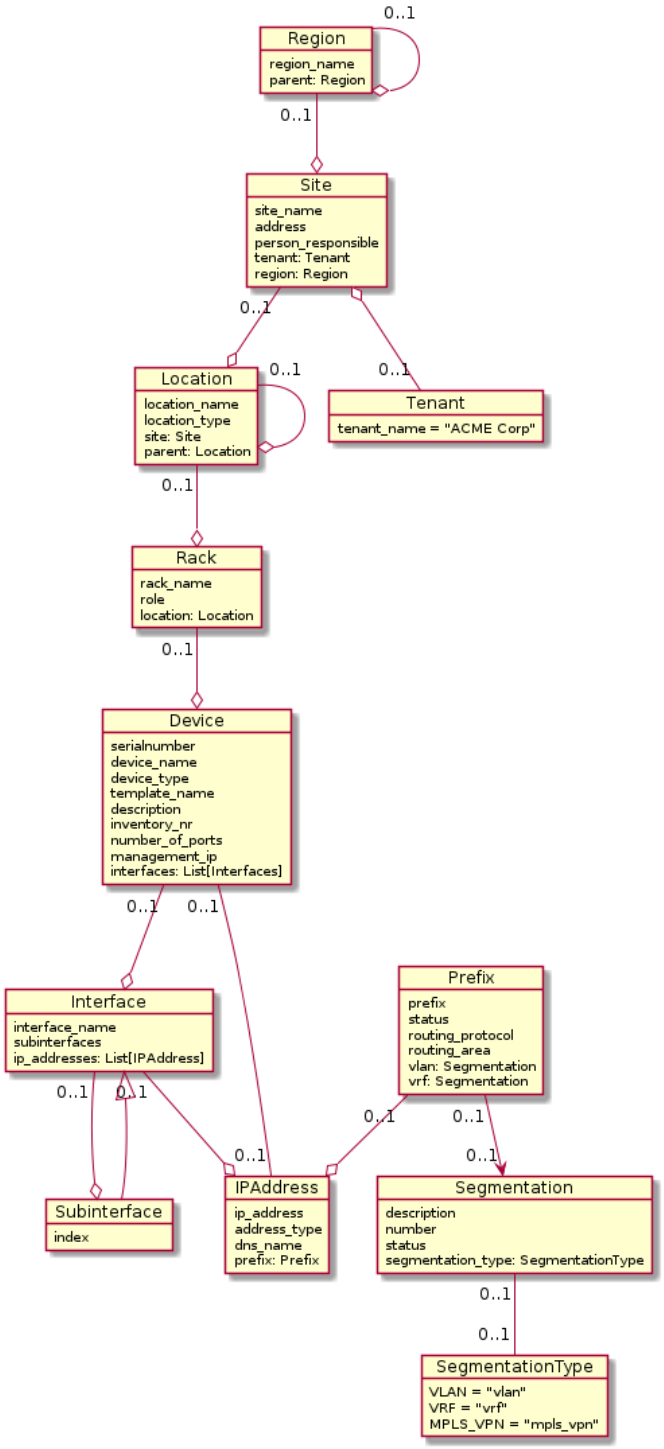


Abbildung 3.4.: Objektdiagramm des Datenschemas

3.3.3.1. Problem

Derzeit wird ein Key-Attribut verwendet, um die Daten miteinander zu verbinden und um so die Identität von Objekten festzustellen.

Das Problem dabei ist, dass diese Daten in den einzelnen Quellen nicht vorhanden, nur teilweise vorhanden oder fehlerhaft sein können.

3.3.3.2. Ziel

Es wäre wünschenswert, dass die Identifizierung von Entitäten verlässlicher implementiert werden könnte. So könnten zwei Objekte immer noch in Relation gesetzt werden, auch wenn nicht alle Attribute übereinstimmten. Hierbei muss beachtet werden, dass viele Attribute nicht in jeder Quellen vorkommen und dementsprechend auch nicht für die Identifikation verwendet werden können.

Im Zweifelsfall ist jedoch das manuelle Eingreifen eines Menschen notwendig, der die Daten in den Quellen selbst anpassen muss. Hierfür bietet es sich an, eine manuelle Verlinkung der Objekte in Kraken zu ermöglichen. Dies wird in dieser Arbeit jedoch nicht behandelt.

Nützlich wäre das auch, falls Kraken einen permanenten Datastore bekäme, zu erkennen, ob ein Gerät nur ausgetauscht wurde, oder den Platz gewechselt hat.

3.3.3.3. Lösungsansätze

Im Moment funktioniert die Identifikation in Kraken über primäre Attribute. Pro Entität wird ein Attribut festgelegt, das zur eindeutigen Identifizierung verwendet werden kann. Ist dieses Attribut unvollständig oder nicht vorhanden, ist es Kraken nicht möglich die Daten zu mergen. Wurde das primäre Attribut in einer Quelle geändert, ist das ebenfalls nicht mehr möglich. Aus diesem Grund wurden verschiedene Ansätze erarbeitet, welche die Identifikation verbessern. Diese werden im Folgenden aufgezeigt.

Cache für Items Bei diesem Ansatz werden die Daten des letzten vergangenen Kraken-Runs gespeichert. Wenn nun zum Beispiel der Wert eines Key-Attributs in einer Quelle ändert, kann das Objekt immer noch identifiziert werden. Diese Methode funktioniert entsprechend erst ab dem zweiten Kraken-Run. Sie funktioniert nach dem Prinzip, dass alle Objekte und Relationen gehasht werden (z.B. mit einem Locality Sensitivity Hash). Kraken müsste dafür so weit umgebaut werden, dass Caching möglich ist und Daten bei jedem Aufruf gespeichert werden. Zudem muss man eine geeignete Hash-Funktion finden.

Der Haken an diesem Ansatz ist, dass die Daten bei der ersten Ausführung alle korrekt sein müssen. Dafür ist diese Vorgehensweise er am besten im Umgang mit Änderungen in den Quellen.

Sekundäre Merkmale mit gewichteten Attributen Diese Methode basiert auf der Umstand, dass es neben den primären Merkmalen noch sekundäre Merkmale gibt. Sind diese genug eindeutig genug, lässt sich eine Identifizierung über sie durchführen. Da bei der Identifikation nicht alle Attribute gleich wichtig sind, ist eine Gewichtung der Attribute sinnvoll. Dazu müssen genug sekundäre Attribute mit ausreichender Einzigartigkeit vorhanden sein, um die Identifizierung zu ermöglichen, was der Schwachpunkt der Methode ist. Existiert ein Attribut, dessen Wert bei allen Objekten identisch ist, ist die Aussagekraft wieder reduziert.

Similarity Diese Methode funktioniert über den Vergleich von überlappenden Attributen. Die Idee dabei ist, dass man gewisse Attribute der verglichenen Objekte anschaut und einen Similarity-Score berechnet. Je mehr Attribute sich zuordnen lassen, desto höher ist der Similarity-Score. Für diesen kann ein Minimum festgelegt werden, ab dem die verglichenen Objekte als gleich gelten. Der Score sollte idealerweise eine Prozentzahl sein, die die Übereinstimmung zweier Objekte beschreibt. Auf diese Weise kann man zum Beispiel sagen, dass ein Objekt vom Typ Device als gleich anerkannt wird, wenn ein Similarity-Score von mindestens 70% erreicht wird.

Der Score wird dabei folgendermassen berechnet:

$$\text{Score} = \frac{\text{Anzahl Gleiche Attribute}}{\text{Anzahl Attribute Total}}$$

Es gibt verschiedene Möglichkeiten, die Similarity zu verwenden. Zum Beispiel könnte der Ansatz der sekundären Attribute mit einer Similarity kombiniert werden. Wenn eine Quelle mehrere Attribute hat, die einzigartig sind, kann man eine Similarity einsetzen, damit bei der Änderung eines einzelnen Werts nicht die gesamten Objekte als verschieden betrachtet werden.

Diese Methode hat jedoch einen Schwachpunkt bei der Überprüfbarkeit. Wie kann hier sichergestellt werden, dass die Methode zuverlässig funktioniert und nur eine kleine Fehlerquote hat? Wie kann die Stabilität dieser Funktion gewährleistet werden?

Locality Sensitiv Hash Die Methode funktioniert ähnlich wie der Similarity-Score, mit dem Unterschied, dass für den Vergleich eine etablierte Hashing-Methode genommen wird. Bei dieser Methode werden Hashes über alle oder spezifische Elemente der Objekte gebildet. Diese werden dann mit anderen Objekten oder anderen Teilen von Objekten verglichen. Liegt die Differenz nicht zu hoch, kann davon ausgegangen werden, dass es sich um das gleiche Objekt handelt. Der Schwachpunkt bei dieser Methode ist, dass sie eine weitere externe Abhängigkeit einführt, da aufgrund der Komplexität eine Library verwendet werden müsste. Bis diese verstanden und verwendet werden kann, bedarf es einer Einarbeitungszeit. Es besteht die Gefahr, dass die Lösung des Problems durch den Funktionsumfang der Library beeinträchtig wird.

Praxistest Localised Hashing In Python scheint es zwei gepflegte Libraries mit LSH zu geben. TrendMicros Locality-Sensitivity-Hashing[2] und DataSketch[3]. Datasketch ist eine Library, welche "Probabilistic-Data-Structures" zur Verfügung stellt, um grosse Datensätze zu analysieren. LSH ist nicht direkt ein Feature von Datasketch. Die andere Library ist TLSH - Trend Micro Locality Sensitive Hash. TLSH erstellt einen Hash-Wert aus einem Byte-Stream. Der Nachteil von TLSH ist, dass erst ab 50 Bytes Input der Hash erstellt wird.

Für den Test wurden drei Byte Streams erstellt. Ein Device, wie es in der Datenquelle vorkommt, wurde zum Testen verwendet.

1. leaf1 mit korrekten Infos
2. leaf2 mit Fehlern
3. leaf3 mit korrekten Infos

Zum Testen wurde Hash1 mit Hash2 und Hash1 mit Hash 3 verglichen.

```
  # Defining Devices
>>> device1 = b'leaf1,1,CH,12,10.20.0.203,default,Router,4,OST-42,Cisco␣CSR␣100v'
>>> device2 = b'leaf2,1,CH,12,10.20.0.203,default,Router,4,OST-42,Cisco␣CSR␣100v'
>>> device3 = b'leaf3,1,CH,12,10.20.0.204,default,Router,4,OST-100,Cisco␣CSR␣100v'

# Hashing Devices
>>> h1 = tlsh.hash(device1)
 T172A022B02000E2200E80C3C800C03803E3BC8A3A00BCA20202CCC820332232C2030A20
>>> h2 = tlsh.hash(device2)
 T16AA022B02000A2200E80C3C800C02803E3BC0A3A00BCA30220CCCC2033233282030A20
>>> h3 = tlsh.hash(device3)
 T158A022382000AF800FC0F3A802C0E803C3B8083800FCA20A00CCC00033200002820300
```

[2] https://github.com/trendmicro/tlsh
[3] http://ekzhu.com/datasketch/documentation.html

```
# Compare the Scores
>>> tlsh.diff(h1, h2)
 13
>>> tlsh.diff(h1, h3)
 79
```

Listing 3.2: Output des Beispiel Programms

In diesem Test wurden die Daten als Byte String der Hash Methode übergeben.

Aufgrund der geringen Differenz zwischen Hash1 und Hash2 können wir sagen, dass es sich dabei mit hoher Wahrscheinlichkeit um denselben Datensatz handelt.

Eigentlich wäre es wünschenswert, dass nur die Attribute verglichen würden, welche auch wirklich eindeutig sind. Der Schwachpunkt der Methode ist deshalb, dass es schwierig ist die 50 Bytes zu erreichen, ohne alle Felder im Datensatz zu verwenden.

Falls nur ein Teil der Attribute vorhanden ist, kann die Zuverlässigkeit dieser Methode nicht gewährleistet werden. Da wir vor allem Datensätze mit wenigen Attributen haben, taugt diese Methode für diesen Anwendungsfall nicht.

Ausserdem müsste evaluiert werden, wie die Daten optimal gehasht werden, damit der Hash aussagekräftig ist.

3.3.3.4. Schlussfolgerungen

Das Problem, welches alle Methoden haben ist, dass sie genug Daten benötigen um überhaupt zu funktionieren.

Die von uns verwendeten Datensätze der Quellen haben wenige Attribute und haben wenig bis gar keine Überlappungen. Gäbe es mehr Redundanzen in den Quellen wären die alternativen Methoden um einiges effektiver.

Der Ansatz eines Caches ist interessant, da Änderungen an Objekten verfolgt werden können. Er ist aber in der Umsetzung mit Abstand am aufwendigsten, da Kraken noch keine Daten persistiert. Zudem baut er zwingend auf einer Objektvergleichsmethode wie dem Localised-Hash oder der Similarity-Berechnung auf, weswegen eine dieser beiden Varianten onehin dafür implementiert werden muss.

Aufgrund der Einschränkung, dass mindestens 50 Bytes an Daten gehasht werden müssen, könnte das Locality-Hashing von uns nur eingeschränkt eingesetzt werden. Die im Hash eingebundenen Attribute könnten nicht frei gewählt werden, da es in vielen Fällen alle braucht um auf 50 Bytes an Informationen zu kommen. Auch müsste evaluiert werden, welches Datenformat sich am besten zum Hashen eignet und wie sich der Hash mit fehlenden Attributen verändert.

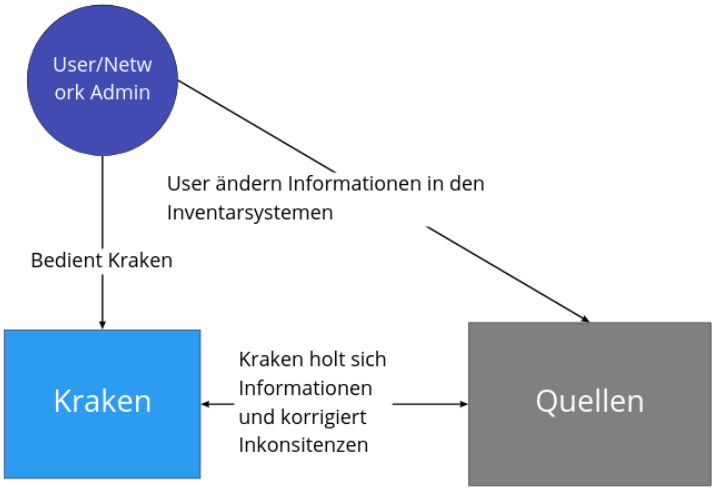


Abbildung 3.5.: C4 Context Diagram

Schlussendlich bleiben noch die Verwendung von sekundären Attributen und die Berechnung einer Similarity übrig. Diese beiden Ansätze eignen sich am besten, da sie keine direkten Einschränkungen haben und werden in einer Kombination implementiert. Zudem werden beim Vergleich von zwei Objekten nur die Attribute verglichen, die von beiden Quellen befüllt werden.

3.3.4. Architektur

Da die Webapplikation im Rahmen dieser Arbeit nicht weiter entwickelt wurde, hat sich die Architektur vereinfacht, da keine Datenbank und kein Webserver mehr mit Kraken verbunden werden müssen. Somit erfolgt die Benutzerinteraktion nur über die CLI. In den nachfolgenden Abbildungen wird das auf mehreren Granularitätsebenen dargestellt.

Das Context Diagramm 3.5 zeigt die Interaktion des Benutzers mit Kraken auf.

Wie in Abbildung 3.6 Container Diagramm ersichtlich, wird die Kraken-Funktionalität von der CLI verwendet um die Aktionen des Benutzers auszuführen.

Das in Abbildung 3.7 ersichtliche Component Diagram lässt sich die Interaktion der CLI mit den jeweiligen Programmteilen erkennen. Die CLI kontrolliert den Programmablauf, holt über den Collector die aufbereiteten Daten der Konnektoren und leitet diese an die Merger weiter.

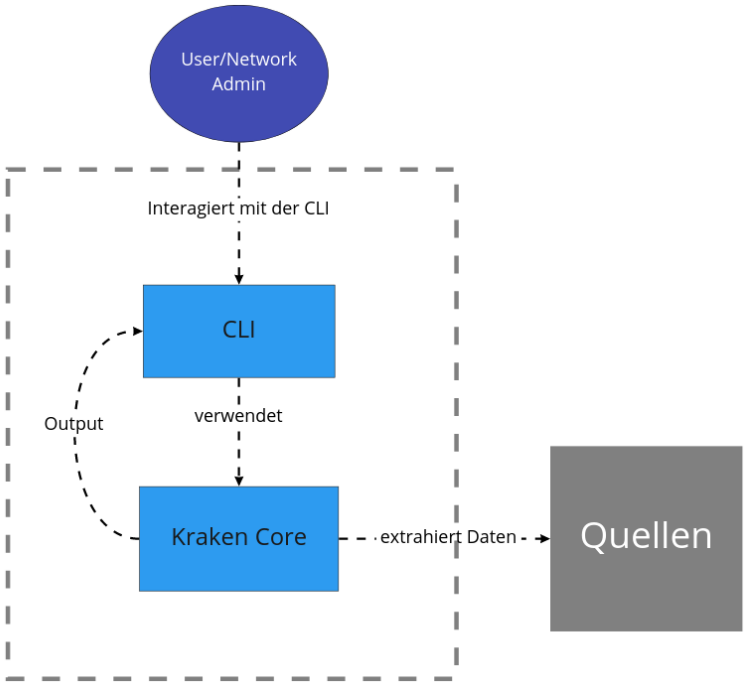


Abbildung 3.6.: C4 Container Diagram

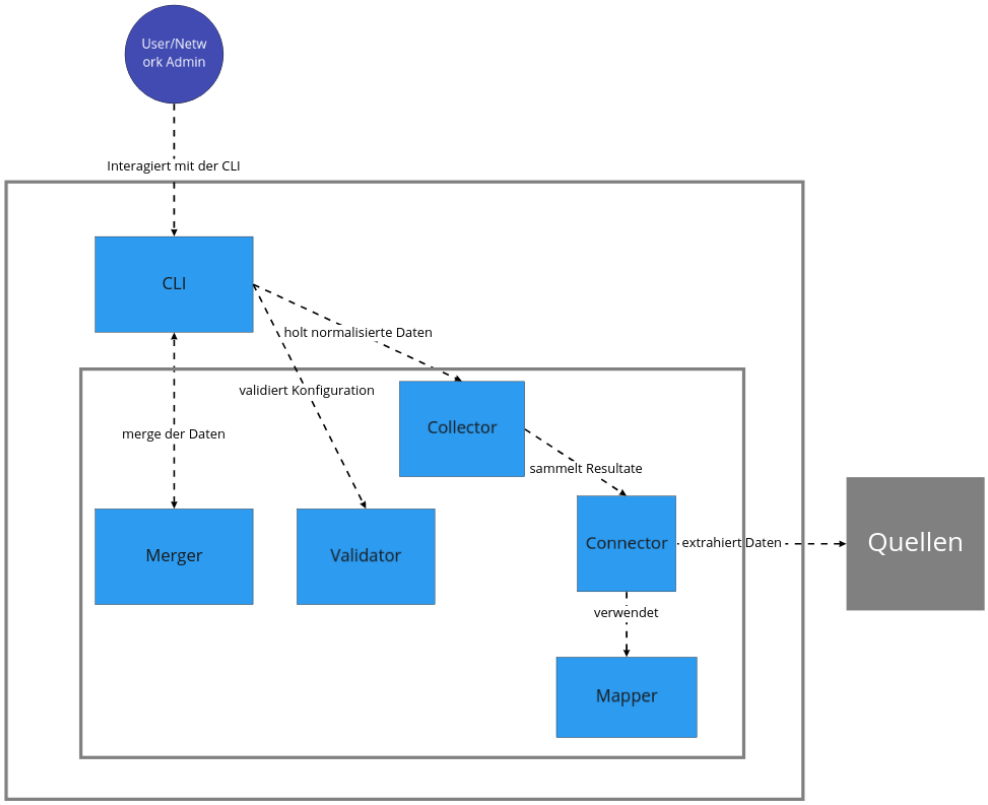


Abbildung 3.7.: C4 Component Diagram

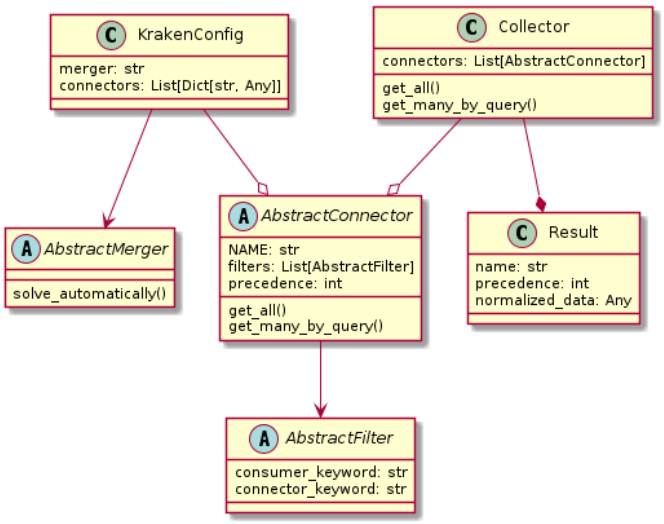


Abbildung 3.8.: C4 Component Diagram

Im nachfolgenden Klassendiagramm in Abbildung 3.8 sind einige Klassen in Beziehung zu einander zu sehen. Bei den Mergern, den Connectors und den Filter wurde der Übersicht halber nur die abstrakte Basisklasse ins Diagramm aufgenommen. Diese Klassenstruktur hat schon in Kraken 1.0 so ausgesehen[3, S. 40]. Die einzige Änderung daran ist die Einführung der *AbstractMerger* Klasse, an deren Stelle vorher die *Merger* Klasse war. Die Implementationen des *AbstractConnector* verwenden jeweils noch eine Mapper Klasse die von *Mapper* erbt und für den jeweiligen Konnektor implementiert ist.

Zudem werden die Implementationen der existierenden Klassen angepasst.

Im Kapitel 3.4 Implementation wird auf die von den Basisklassen erbenden Klassen genauer eingegangen.

3.4. Implementation

3.4.1. Konnektoren

Mit dem Ansatz eines Objekt basierten Datenmodells besteht der Vorteil darin, dass es weniger Redundanzen gibt. Eine eindeutige Dateninstanz kann einmal existieren und mehrfach referenziert werden. Wie können diese Dateninstanzen nun gefunden werden, um sie zu referenzieren, wenn sie von einer Quelle importiert werden?

Oft wird das mit einer Datenbank und einem ORM gelöst. Das hat allerdings den Nachteil, dass es die Komplexität der Applikation erhöht und eine neue Abhängigkeit hinzufügt. Zudem wird das Testing erschwert, da auch noch eine Test-Datenbank zu verwaltet werden muss. Des Weiteren gestaltet sich so die Installation für den Nutzer als schwieriger. Für die Versionierung ist eine Datenbank jedoch wieder interessant, da dafür Daten persistiert werden müssen. In diesem Fall ist eine Datenbank auch nicht das Erste, an das man denkt, denn Kraken verarbeitet Daten nur weiter. Würde Kraken eine Datenbank verwenden und Daten zwischen Aufrufen persistieren, wären diese Daten schon nach einem Applikations-Aufruf nicht mehr aktuell. Aus diesen Gründen fällt diese Variante weg.

Eine Alternative wäre, eine Objekt-Registration zu erstellen, welche die Objekte nicht persistiert. Das heisst, es werden die Referenzen zu allen existieren Objekten erfasst. Im Falle, dass ein neues Objekt, basierend auf den Daten der Quelle, erstellt wird, wird überprüft, ob das Objekt schon in der Datenstruktur existiert. Somit werden Redundanzen beim Importieren von Daten vermieden.

3.4.1.1. Existierender Kraken

Im Kraken 1.0 wird ein Dict-Baum pro Quelle generiert[3, S. 53]. Der Baum hat immer die gleiche Struktur, aber nicht die gleichen Informationen. Diese generierten Bäume werden dann DeepDiff übergeben, welcher die Bäume miteinander vergleicht. DeepDiff erkennt Änderungen im Baum und listet sie auf. Anhand von diesen Änderungen werden dann die Daten der beiden Quellen zusammengeführt. Der Dict-Baum fängt mit Device-Informationen an und beinhaltet Interfaces, welche wiederum Subinterfaces beinhalten. Beim Mergen werden mehrere solcher Strukturen vereinigt.

Jede Struktur entspricht dabei dem Normalisierungsformat. Das heisst auch, dass jede Struktur beim Device anfängt. Beim Vergleich der Dicts wird auf jeder Ebene ein sogenanntes Key-Attribut verglichen, was beim Device der Device-Name ist.

Die Struktur vom Device aus Entspricht auch dem loose von OpenConfig definierten Baum[3, S. 53]. Dies hat jedoch den Nachteil, das Metainformationen, wie zum Beispiel die Angabe des Racks, nicht in dieser Struktur gespeichert werden können. Da es aber vorkommen kann, das gewisse Merkmale mehrfach erfasst werden, sind Metainformationen nötig um eine Eindeutigkeit zu gewährleisten. Darum haben wir uns entschieden, uns mindestens bei der internen Datenstruktur nicht an OpenConfig zu halten.

OpenConfig besitzt auch noch ein Parent-Child Beziehung in jedem Node[3, S. 53]. Dies ist angenehm für den Benutzer, aber hätte in diesem Fall die Komplexität der Arbeit gesprengt.

Die Konnektoren der Vorgängerarbeit wurden in den Ordner `legacy\_connectors` verschoben, da sie mit der neuen Datenstruktur nicht mehr funktionieren. Dennoch wurden die Dateien im Projekt gelassen, da einige Tests der Vorgängerarbeit, zum Beispiel die zum Collector, noch von diesen Konnektorenklassen abhängen. Aus Zeitgründen haben wir uns entschieden diese Tests nicht umzubauen, da sie der neuen Funktionalität nicht in die Quere kommen.

3.4.1.2. Neuer Kraken

Der Kraken soll nun neu eine Python Objektstruktur verwenden. Dies soll die interne Verwaltung und Abfrage der Objekte vereinfachen. Was wiederum das Manipulieren von Informationen vereinfachen soll, sowohl für die Applikation als auch den Programmierer.

Der bestehende Code soll nun angepasst werden, damit die Vorteile einer Objektstruktur genutzt werden können.

Um dies zu ermöglichen, müssen folglich alle Quellen eine Liste von Objekten einer bestimmten Klasse zurückgeben, damit DeepDiff einen Vergleich machen kann. Wir gehen dafür auch vom Device aus, da aus den Daten von Kraken schlussendlich ja Konfigurationen für Devices gemacht werden sollen. Folglich muss neu jeder Konnektor eine Liste an Devices zurückgeben, um mit DeepDiff verwendet werden zu können.

Das ist aber bei der derzeitigen Implementation nur bei der Inventarquelle und der IP-Adress-Quelle der Fall, da nur diese Informationen zum Device enthalten. Deswegen müssen die anderen Quellen separat behandelt werden.

Die neuen Quellen werden jeweils mit dem Präfix `Source` benannt um einen klaren Unterschied zu den in der Vorgängerarbeit geschriebenen Konnektoren und zukünftigen selbst geschriebenen Konnektoren zu schaffen.

Somit haben wir fünf neue Konnektoren:

- SourceInventoryConnector
- SourceIPConnector
- SourceLocationsConnector
- SourcePrefixConnector
- SourceSegmentationConnector

3.4.2. Filters

Ein Filter wird in Kraken dazu verwendet, die Abfrage von Quellen so weit zu limitieren, dass nur die relevanten Objekte zurückgegeben werden.

Für die Implementation der Filter musste auf den jeweiligen Konnektoren die Methode `get_many_by_query` implementiert werden, die vom `AbstractConnector` vorgegeben ist. Zudem wurden für die neuen Konnektoren neue Filter-Klassen erstellt. Um die Implementation von Filter in den Konnektoren zu vereinfach wurde eine Funktion in der `AbstractConnector`-Klasse implementiert, die basierend auf dem Query, der an die `get_many_by_query` Methode übergeben wird, die erste Ebene eines Dicts nach den in den Filtern definierten Attributen durchsucht. Das heisst, bei CSV-Dateien und der ersten Ebene einer Dict-Struktur einer JSON-Datei kann direkt diese Funktion verwendet werden.

Der Multilevel-Filter konnte aus Zeitgründen nicht implementiert werden, weshalb diese Funktion deaktiviert wurde.

Folgendes Problem soll mit dem Multilevel-Filter gelöst werden:

> „Die Suche nach `dns="hallo.kraken.ch"` ist beispielsweise nur in System A möglich, da nur dieses überhaupt DNS-Informationen speichert. Trotzdem muss Kraken auch von allen anderen Systemen B, C, D, E alle Daten holen, die mit dem Suchergebnis `dns="hallo.kraken.ch"` assoziiert sind. Der Grund: Würde Kraken nur Daten von System A holen (da nur dort die Suche nach `dns="hallo.kraken.ch"` möglich ist), wären im Endresultat nur Daten von System A enthalten – alle anderen Systeme (B, C, D, E), da sie nicht über DNS-Informationen verfügen bzw. die entsprechende Suche, fehlen dann in der End-Konfiguration. Darum ist der Multilevel-Filter nötig: Er sorgt dafür, dass `dns="hallo.kraken.ch"` nicht nur das Resultat von System A liefert, sondern auch die entsprechenden Einträge von Systemen B, C, D, E [3, S. 51].“

Dieser Filter basiert darauf, dass bei der Implementation mit Dicts immer ein Dict-Key mit einem Schlüsselwert befüllt ist[3, S. 51]. So werden dann die Konnektoren mit diesen Schlüsselwerten gefiltert. Bei Objekten ist das aber nicht mehr der Fall. Ein Ansatz für die Umsetzung eines Multilevel-Filters wäre, auf jedem Model eine Methode zu definieren, die eine Liste mit suchbaren Attributen zurückgibt.

3.4.3. Merger

Der Merger das Herzstück von Kraken. Er fügt die verschiedenen Informationen aus den verschiedenen Quellen zu einem umfassenden Baum zusammen.

Beim Mergen soll möglichst viel des bestehenden Kraken wiederverwendet werden und um die nötige Funktionalität ergänzt werden. Der bisherige Merger funktioniert mit DeepDiff[3, S. 60]. Nun muss er für die Objektstruktur angepasst werden. Zudem muss ein neuer Merger geschrieben werden, damit die Zuweisung von IP-Adresse zu IP-Prefix funktioniert. Des Weiteren müssen neue Merger geschrieben werden, die mit der Segmentations-Quelle, der Prefix-Quelle und der Locations-Quelle umgehen können.

Um ein stabiles Interface für die Mergers zu haben wurde die Klasse `AbstractMerger` implementiert.

```
class AbstractMerger(ABC):
  @abstractmethod
  def solve_automatically(self, results: List[Result], as_result=False) -> Any:
      pass
```

Die Funktion `solve_automatically` hat in der ersten Version schon auf dem damaligen Merger existiert und wurde deshalb im `AbstractMerger` übernommen. Sie nimmt eine Resultats-Liste entgegen und gibt je nach Wert des Parameters `as_result` ein `Result`-Objekt oder die direkte Liste der Daten. In der ersten Version von Kraken wurde im Merger direkt die Liste der Daten zurückgegeben. Da wir aber nun mehrere Merger haben die zum Teil aufeinander aufbauen, gibt es nun die Option des Resultat-Objekts da darin auch der Name der Quelle und deren Präzedenz enthalten sind.

Zudem wurde der vorherige `Merger` in `DeepDiffMerger` umbenannt, da es sich hier nun um einen spezialisierten Merger handelt.

3.4.3.1. DeepDiff Merge

DeepDiff[4] ist eine Diffing-Library, die ähnlich wie der Diff bei einer Versionsverwaltung funktioniert. Zwei hierarchische Strukturen können miteinander verglichen werden und die Änderungen von der ersten zur zweiten Struktur werden aufgeführt. Bei diesen gibt DeepDiff für alle gefundenen Änderungen, je nach Typ der Änderung, gruppiert eine Liste zurück, die aus einem Zugriffspfad und dem geänderten Wert besteht. Dieser Zugriffspfad sieht bei Kraken 1.0 immer ungefähr so aus[3, S. 60]:

```
1 {
2   "values_changed": {
3     "root['leaf1']['interfaces']['GigabitEthernet1']['dns_name']": {
4       "new_value": "hallo.kraken.ch",
5       "old_value": "hallo.kr4k3n.ch"
6     }
```

[4]Stand 12.01.2022 `https://deepdiff.readthedocs.io/en/latest/diff.html`

```
7   }
8 }
```

Listing 3.3: Pfad beim Vergleichen von Objekten mit DeepDiff

Der Pfad zum korrekten Wert wird hier als String-Wert angegeben und die Key-Namen werden mithilfe einer Regular-Expression extrahiert.

Da wir nun Objekte verwenden, sieht dieser Pfad ein wenig anders aus. Ein Beispiel um dis zu verdeutlichen:

```
1 {
2   "values_changed": {
3     "root.rack.location.location_name": {
4       "new_value": "OST",
5       "old_value": "HSR"
6     }
7   }
8 }
```

Listing 3.4: Neuer Pfad des Objektbasierten Vergleichs.

Die Attributnamen kann man hier einfach mit `.split('.')` extrahieren, mit mehrfachem Aufruf der *getattr* Funktion das zu ändernde Objekt finden und mit der *setattr* Funktion den neuen Wert setzen.

Mit Objekten ist dies nicht trivial, da an gewissen Orten Auflistungen verwendet werden. Konkret haben in unserem Datenschema zum Beispiel Devices mehrere Interfaces und diese können wiederum mehrere IP-Adressen haben. Zudem müssen anstelle von Listen Dictionaries verwendet werden, da DeepDiff nicht gut mit Listen umgehen kann. Das hat zur Folge, dass die Interfaces und IP-Adressen von DeepDiff nur mit einem Schlüsselattribut verglichen werden, was bei denen aber gut funktioniert.

Standardmässig wird beim Vergleich von Listen das erste Element der ersten Liste mit dem ersten Element der zweiten verglichen. Das heisst, der Inhalt der Elemente wird gar nicht beachtet. Dies passiert aus dem Grund, dass DeepDiff verhindern will, dass die Listen mit einer Laufzeit von $O(n^2)$ verglichen werden. Man kann das Flag `inorder=False` setzen, aber damit hat der Vergleich der Objekte nicht zuverlässig funktioniert.

Aus den oben genannten Gründen ist die Verwendung von Dicts in Kombination mit DeepDiff am besten.

Dementsprechend kann nun ein solcher Pfad folgendermassen aussehen:

```
"root.interfaces["GigabitEthernet1"].dns_name"
```

Da man Dicts nicht mit `getattr` und `setattr` bedienen kann, muss man hier eine andere Lösung finden. Es müsste dafür ein eigener Interpreter geschrieben werden, welcher erkennt, ob ein Objekt- oder ein Dict-Zugriff erfolgt, damit die richtigen Operationen ausgeführt werden. Dies ist aber zeitaufwändig, kann sehr komplex werden und würde den Rahmen dieser Arbeit sprengen. Deshalb müssen Alternativen dazu angeschaut werden.

Da so ein Pfad, wie oben beschrieben, eigentlich valider Python-Code ist, können Funktionen wie `eval` und `exec` in Betracht gezogen werden. Der Code kann aber nicht wissen, wie auf das letzte Attribut im Pfad zugegriffen wird, deshalb können wir hier nur die `exec` Funktion verwenden.

Der effektiv ausgeführte Code soll im Endeffekt zum Beispiel folgendermassen aussehen:

```
root.interfaces["GigabitEthernet1"].dns_name = "ost.ch"
```

Der Code des Aufrufs der Exec-Funktion sieht so aus:

```
exec(f"{path} = new_value")
```

Die `path` Variable ist hierbei der oben aufgeführte Pfad als String und new_value ist eine Variable, die den Inhalt des neuen Werts enthält.

Die Verwendung der `exec` Funktion bringt ein gewisses Sicherheitsrisiko mit sich. Deshalb stellt sich hier die Frage, wie die Verwendung so sicher wie möglich gemacht werden kann, oder ob es nicht sicherere Alternativen gibt.

In diesem Fall ist der Angriffsvektor die `path` Variable, weil sie in den String interpoliert wird. Von sicherheitsrelevanter Bedeutung ist, ob in dieser Variable arbiträrer Code versteckt werden kann, welcher dann ausgeführt werden würde. Die Variable selbst enthält keinen User-Input, sondern nur eine Zusammensetzung aus Attributnamen aus Krakens Datenschema. Der eigentliche Wert in der *new_value* Variable wird nicht in den String interpoliert und wird deshalb auch nicht ausgeführt. Aus diesem Grund besteht keine realistische Gefahr einer Code-Injection. Trotzdem sollte diese Funktion entsprechend abgesichert werden.

Es wird nur der benötigte Kontext mitgegeben.

```
exec(f"{path} = new_value", dict(), {"root": root, "path": path, "new_value":
  new_value})
```

Listing 3.5: Exec für die Pfad Interpolation

Somit gibt es keine global zugreifbare Variablen und die lokalen Variablen sind als drittes Argument der `exec` Funktion definiert. Zudem wird die `path` Variable nach runden Klammern und `import` Statements durchsucht, damit keine Funktionen oder Skripts ausgeführt werden können.

Fazit zu DeepDiff Was bei DeepDiff fehlt ist eine umfangreiche Dokumentation, die die verschiedenen Konflikt-Typen erklärt, beschreibt was genau die Unterschiede beim Vergleich von Objekt- und Dict-Strukturen sind und welche Bedingungen ein Objekt erfüllen muss um erfolgreich verglichen zu werden. Gerade im Umgang mit Objekten und Listen ist das Verhalten der Library manchmal schwer berechenbar und es kann nur durch Versuche herausgefunden werden, was genau funktioniert. Dadurch gibt es viele unbekannte Variablen mit denen umgegangen werden muss, was die Implementation erschwert und zu einem falschen Verständnis führen kann. Dazu sei erwähnt, dass DeepDiff normalerweise nicht für den Zweck der Datenaggregation verwendet wird, wodurch der Umgang mit der Library zu diesem Zweck unintuitiver wird.

DeepDiff wurde von uns hauptsächlich verwendet, weil der Merger in der Vorgängerarbeit damit implementiert wurde, welche von uns angepasst werden soll. Rückblickend würden wir für den Merger jedoch die Implementation eines eigenen Algorithmus bevorzugen, da dieser auch einfach zu testen wäre.

Ursprünglich war auch angedacht nur den `DeepDiffMerger` umzubauen, um so neben dem `PrefixIPMerger` nur einen Merger zu haben, doch durch Tests mit der Library wurde klar, dass der Merger nur funktioniert, wenn die Datenstrukturen von allen Quellen beim gleichen Objekt beginnen. So können zum Beispiel keine *Devices* mit *Racks* verglichen werden. Ein Ansatz um dieses Problem zu umgehen war, dass zum Beispiel diese *Racks* jeweils in ein leeres *Device*-Objekt gegeben werden, mit der Idee, dass der Vergleich dann ab dem ersten, nicht leeren Attribut passiert, sprich dem *rack*-Attribut im *Device*. So würden dann wieder *Racks* mit *Racks* verglichen werden.

Dies hat jedoch nicht das erwartete Resultat geliefert, weshalb dieser Ansatz verworfen werden musste.

3.4.3.2. Prefix-IP-Merger

Der Fall von Prefix und IP-Adresse ist beim Merger speziell, da durch Berechnungen festgestellt werden kann, ob eine IP-Adresse in einem Prefix enthalten ist. Es sollen aber nur IP-Address Objekte erstellt werden, für IPs welche auch tatsächlich in den Prefixen vergeben sind. Um die Daten der zugewiesenen IP-Adressen und der Prefixe zu mergen, braucht es hierbei eine Speziallösung, da DeepDiff mit dieser Art von Beziehung nicht umgehen kann.

Aus diesem Grund müssen zwei neue Merger implementiert werden. Der erste Merger kann dabei die Quelle der zugewiesenen IP-Adressen mit der Quelle der Prefixes mergen und der zweite ruft den Prefix-IP-Merger und den DeepDiff-Merger auf, um weitere Quellen mit diesen beiden zu mergen.

Um festzustellen, ob eine IP-Adresse in einem Prefix enthalten ist, wir das Python-Modul `ipaddress`[5] verwendet. Nachfolgend Beispielcode welcher die Funktionsweise von `ipadress` zeigt:

```
prefix = ipaddress.ip_network("192.168.5.0/24")
ip_address = ipaddress.ip_address("192.168.5.120")
if ip_address in prefix:
  ...
```

Findet hier eine Übereinstimmung statt, wird der Prefix der IP-Addresse zugewiesen.

3.4.3.3. Weitere Merger

Nicht alle in unserem Beispiel definierten Quellen können direkt eine Liste von Devices zurückgeben.

Die Konnektoren der IP-Adressen und des Inventars geben beiden eine Liste Devices zurück, somit können sie direkt verglichen werden. Für den IP-Konnektor wurde ein JSON der Netbox-API verwendet. Dieser besitzt im Gegensatz zum CSV Export von Netbox mehr Informationen.

Bei den anderen Konnektoren sieht das etwas anders aus. Die Location-Source gibt eine Liste von Racks zurück und die Segmentation-Source gibt eine Listen von Segmentations zurück. Deshalb müssen sie jeweils mit der Quelle gemerged werden, mit der sie eine Überschneidung haben.

Die Prefixe enthalten ebenfalls Segmentations-Informationen, in Form der Nummer und des Segmentations-Typs. Somit können deren Daten direkt verknüpft werden.

Die Devices der Inventar-Source haben eine Referenz zum Rack, weshalb sie mit den Racks der Location-Source gemerged werden können.

Der Vergleich von zwei Racks sieht dann im Merger beispielsweise so aus:

```
device.rack == rack
```

[5]Das Modul ist in der Standard-Library von Python enthalten: `https://docs.python.org/3.9/library/ipaddress.html`

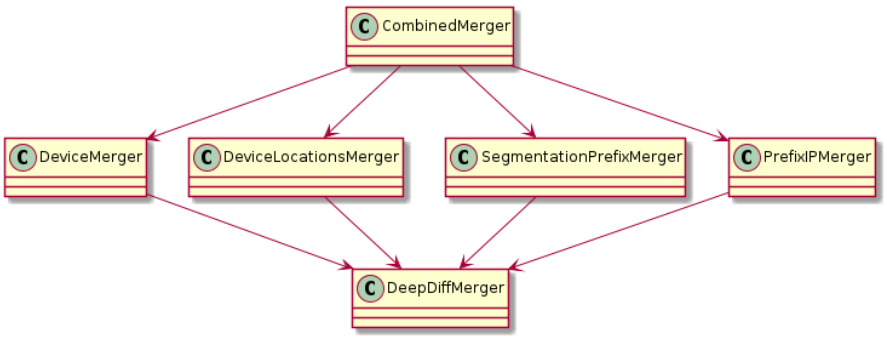


Abbildung 3.9.: C4 Component Diagram

So werden die Entitäten der Quellen über die `__eq__`-Funktion miteinander verglichen. Der Vergleich hat eine Laufzeitkomplexität von O(n^2), da jedes Element der einen Quelle mit jedem Element der anderen Quelle verglichen werden muss. Würde der Vergleich nur über ein Attribut erfolgen, könnte dieser mit Dicts umgesetzt werden, welche eine Laufzeit von O(n) hätten. So könnte aber keine Equals-Funktion verwendet werden, was den Vergleich einschränkt. Mehr dazu in Kapitel 3.4.4.

Nachdem die übereinstimmenden Objekte gefunden wurden, werden sie dem `DeepDiffMerger` übergeben, der dann die Attribut-Werte zusammenführt. Dafür werden die Objekte in eigene `Result`-Objekte verpackt und übernehmen den Namen und die Präzedenz der zugehörigen Quelle.

3.4.3.4. CombinedMerger

Der `CombinedMerger` kombiniert die verschiedenen Merger und koordiniert deren Verwendung. Am Schluss gibt er eine aggregierte Device-Liste zurück, die schlussendlich von Kraken ausgegeben wird. In der Abbildung 3.9 sind die Merger-Klassen in ihrer Beziehung zueinander abgebildet.

Damit nicht immer alle fünf Quellen angegeben werden müssen, werden nur die vorhandenen Quellen gemerged. Mindestens eine Quelle muss dabei aber entweder das Inventar oder die IP-Verwaltung sein, da mindestens eine Device-Liste vorhanden sein muss. Wenn beispielsweise kein `SourceLocationsConnector` angegeben wurde, werden die Daten des `SourceInventoryConnector` direkt dem `DeviceMerger` übergeben.

Die Quellen werden, wie in Abbildung 3.10 ersichtlich, miteinander verbunden.

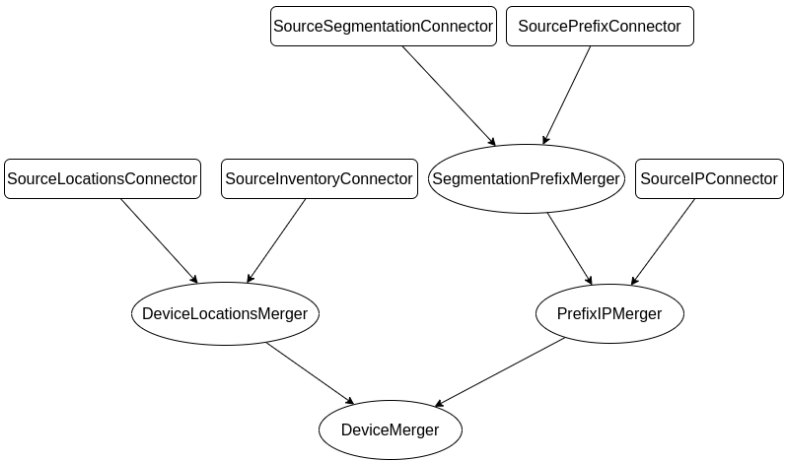


Abbildung 3.10.: Data-Flow Mergers und Konnektoren

Die Merger `DeviceLocationsMerger` und `PrefixIPMerger` geben jeweils eine Liste Devices zurück, während der `SegmentationPrefixMerger` eine Liste Prefixes zurückgibt. Diese Listen sind wiederum in Resultat-Objekten verpackt.

3.4.4. Vergleich der Objekte

Im Kapitel 3.3.3 wurden mehrere Optionen zur Verbesserung des Objekt-Vergleichs evaluiert. Die Ansätze der sekundären Attribute und des Similarity-Vergleichs werden nun umgesetzt.

In einem Prototyp[6] wurde diese Methode getestet, um deren Umsetzung anhand der Device-Klasse auszuprobieren.

Beim Vergleich der Objekte wird die in der jeweiligen Model-Klasse definierte `__eq__`-Methode verwendet.

Die Model-Klassen haben typischerweise ein Attribut, das als Identifikator verwendet werden kann. Im Fall der IP-Adresse gibt es die Adresse selbst und im Beispiel des Devices gibt es dafür den Device-Namen. Ein Device könnte beispielsweise über die Seriennummer identifiziert werden. Potenziell könnten neue Quellen auch die Media Access Control (MAC)-Adresse speichern und exportieren. Um mit mehreren eindeutigen Attributen umzugehen, wird zuerst das wahrscheinlichste Attribut überprüft, im Fall des Devices der Device-Name. Anschliessend wird anhand der anderen für den Vergleich definierten Attribute die Similarity, wie in Kapitel 3.3.3.3 beschrieben, berechnet.

[6] `https://gitlab.ost.ch/diktyo/ba-sena-adorabilis`

Um die gleiche Funktionalität nicht für jedes Model schreiben zu müssen wurde dafür eine Funktion geschrieben.

```
def equals_attr_list(obj1: Any, obj2: Any, attr_list: List[str], min_percentage=0.5)
    -> bool:
  num_maches = 0
  num_non_empty = 0
  for attr in attr_list:
      value1 = getattr(obj1, attr, None)
      value2 = getattr(obj2, attr, None)
      if value1 and value2:
          num_non_empty += 1
          if value1 == value2:
              num_maches += 1
  if num_non_empty == 0:
      return False
  mach_percentage = num_maches / num_non_empty
  return num_maches > 0 and mach_percentage >= min_percentage
```

Listing 3.6: Funktion equals_attr_list

Die ersten beiden Parameter sind die zu vergleichenden Objekte, der Parameter `attr_list` beinhaltet eine Liste von Attribut-Namen und der Parameter `min_percentage` setzt einen Threshold, ab dem die Similarity als Übereinstimmung akzeptiert wird.

Bei der Verwendung dieser Funktion muss mit dem *min_percentage* Parameter intelligent umgegangen werden, denn welcher Wert hier funktioniert, muss schliesslich aus Versuchen mit realen Daten entnommen werden. Gerade bei wenigen verglichenen Attributen dürfen auch weniger nicht übereinstimmen und der Einfluss des einzelnen Attributs ist stärker.

3.4.5. Serialisierung

Mit dem CLI-Argument `--output output_filepath` kann ein Output-File angegeben werden, in dem die Daten im JSON-Format abgelegt werden sollen.

Weil bei der bisherigen Applikation nur ein verschachteltes Dict serialisiert werden musste, konnte dort einfach die Funktion `json.dump(final_config, output_file, indent=4)` des Modules *json* verwendet werden, da eine Dict-Struktur praktisch gleich wie eine JSON-Struktur aufgebaut ist.

Obwohl Python-Klassen mit `__dict__` zu einem Dict transformiert werden können, wird diese Methode von `json.dump` nicht automatisch aufgerufen. Dafür kann der Funktion eine Encoder-Klasse mitgeben werden. Diese sieht in Kraken folgendermassen aus:

```
class Encoder(json.JSONEncoder):
  def default(self, obj):
    return obj.__dict__
```

Listing 3.7: JSON Encoder

Auf jedem Objekt wird nun von `json.dump` die `__dict__` Funktion aufgerufen.

Eine Alternative wäre die Verwendung von `dataclasses.asdict(obj, *, dict_factory=dict)`. Diese Methode wurde aber nicht verwendet, die Serialisierung nach JSON die interne `__dict__` Funktion ausreicht.

3.4.6. Template

Wenn man einen spezifisch formatierten Output haben möchte, kann man Kraken eine Python-Datei übergeben, in der man die Funktion `def output_template(device):` definiert. Diese Funktion wird auf jedes Element des Outputs angewendet. Da Kraken eine Liste an Devices ausgibt, ist das Argument, das dieser Funktion übergeben wird ein *Device*-Objekt.

3.4.7. Validatoren

Die in der Vorgängerarbeit implementierten Validatoren basieren auf einem Validations-Dict, das definiert, welche Attribute gesetzt werden dürfen und welche Werte und Typen erlaubt sind. Dabei macht die Library Cerberus[7] die eigentliche Validation, und falls gewisse Dinge nicht dem Schema entsprechen, werden diese in einer Liste aufgeführt[3, S. 46]. Diese Liste wird anschliessend im Terminal ausgegeben.

3.4.7.1. Kraken-Config

Das Config muss in dieser Arbeit nur auf die neuen Konnektoren und Optionen angepasst werden. In der Liste der erlaubten Konnektoren befinden sich nun die in dieser Arbeit implementierten Konnektoren, sowie die erlaubte Auswahl an Mergern, die im Config angegeben werden darf. Als Änderung im Vergleich zur Vorversion wurden hier keine hartkodierte Strings verwendet[8], sondern es wurde die Werte bestehende *INDEX*-Variable, welche eine Liste der Konnektoren beinhaltet, angepasst verwendet. Zudem

[7] Stand 12.01.2022 `https://docs.python-cerberus.org/en/stable/`

[8] Stand 12.01.2022 `https://gitlab.ost.ch/ins-stud/ba21-nettowel-kraken/-/blob/d8f12aaa341613f12ac3c45ee27d0020ef1bbaba/kraken/definitions.py`

wurde direkt bei den Mergern die Variable *MERGERS* erstellt, in der die Merger registriert werden.

```
KRAKEN_CONFIG_VALIDATION_SCHEMA = {
  "merger": {"type": "string", "regex": f"^{'|'.join(MERGERS.keys())}$"},
  "connectors": {
    "type": "list",
    "schema": {
      "type": "dict",
      "schema": {
        "name": {
          "type": "string",
          "allowed": list(INDEX.keys()),
        },
        "precedence": {"type": "integer"},
        "arguments": {"type": "dict", "allow_unknown": True},
      },
    },
  },
}
```

Listing 3.8: Kraken-Config Validations-Schema

3.4.7.2. Merge-Template

Bei der Validation des Merge-Templates wurde in der Vorgängerarbeit der gleiche Ansatz wie beim Kraken-Config gewählt. Der Unterschied ist jedoch, dass das Schema um ein vielfaches grösser ist, als das des Kraken-Configs, da alle Attribute des Normalisierungsformats in das Dict geschrieben wurden. Wärend das Kraken-Config ca. 13 Zeilen umfasste, betrug die Länge der Merge-Template-Definition ca. 80 Zeilen[9].

Um zu verhindern, dass jedes Attribut in dieser Definition bei Änderungen nachgetragen werden muss, wurde in dieser Arbeit eine andere Lösung gesucht. Die implementierte Lösung basiert nicht mehr auf Cerberus, sondern ist eine selbst geschriebene Funktion.

Im Package `models` in der Datei `__init__.py` befindet sich eine Auflistung aller Konnektorenklassen. Deren Attribute werden mit Hilfe der `dir()` Funktion extrahiert und das geladene Merge-Template wird überprüft. Dabei wird überprüft, ob die Modelnamen in Kraken enthalten sind, ob sich die genannten Attribute auf den entsprechenden Klassen befinden, ob jedem Attribut im Merge-Template ein existierender Konnektor im `PRECEDENCE`-Attribut hat.

[9]Stand 12.01.2022 `https://gitlab.ost.ch/ins-stud/ba21-nettowel-kraken/-/blob/d8f12aaa341613f12ac3c45ee27d0020ef1bbaba/kraken/definitions.py`

Kategorie	Thema	erreicht
Funktionelle Stabilität	Vollständigkeit	Ja
	Korrektheit	Ja
	Angemessenheit	Ja
Performante Effizienz	Ressourcenverwendung	*
Kompatibilität	Interoperabilität	*
Verwendbarkeit	Erkennbare Angemessenenheit	Ja
	Erlernbarkeit	Ja
	Bedienbarkeit	Ja
	Fehlerschutz	Ja*
Zuverlässigkeit	Fehlertoleranz	Ja
Security	Vertraulichkeit, Integrität	*
Wartbarkeit	Modularität	Ja*
	Analysierbarkeit	*
	Modifizierbarkeit	Ja*
	Testbarkeit	Ja
Übertragbarkeit	Installierbarkeit	Ja*

Tabelle 3.1.: Auswertung der Nichtfunktionalen Requirements

```
Device:
  device_type:
    PRECEDENCE: "SourceInventoryConnector"
IPAddress:
  dns_name:
    PRECEDENCE: "SourceInventoryConnector"
```

Listing 3.9: Beispiel valides Merge-Template

3.5. Statistische Auswertung

3.5.1. Nicht-funktionale Anforderungen

Performante Effizienz: Zeitverhalten Nicht getestet

Kompatibilität: Interopabilität Die Anforderung, dass Kraken in einem Docker-Container laufen können soll ist nicht mehr relevant, da kein Webserver mehr betrieben wird.

Das Normalisierungsformat entspricht nicht einem in der Netzwerkdomäne anerkannten Datenformat.

Verwendbarkeit: Fehlerschutz Es existiert derzeit keine manuelle Konfliktlösung.

Security Es existiert im Rahmen dieser Arbeit kein Webserver, daher ist dieser Punkt nicht mehr relevant.

Wartbarkeit: Modularität Die Quelle muss in einem DeepDiff kompatiblen Format sein, sonst muss ein separater Merger dafür entwickelt werden. Ausserdem ist es derzeit nicht möglich ohne Progammcodeänderungen Quellen hinzuzufügen.

Wartbarkeit: Modularität Kraken kann eine beliebige Anzahl an Quellsystemen anbinden, solange deren Konnektoren Device-Objekte zurückgeben.

Wartbarkeit: Analysierbarkeit Kraken ist Closed-Source, von dem her nicht analysierbar.

Wartbarkeit: Modifizierbarkeit Programmierkenntnisse sind derzeit für die Anbindung neuer Quelle vonnöten. Konfigurieren benötigt keine Programmierkenntnisse.

Übertragbarkeit: Installierbarkeit Die Anforderung, dass man Kraken in einem Docker-Container nutzen kann, ist nicht mehr relevant, da kein Webserver mehr betrieben wird.

Durch die kurze Implementationszeit war es nicht möglich alle Nicht-funktionalen Anforderungen zu erfüllen. Der Fokus wurde deshalb auf algorithmische Probleme gelegt. Zudem sind gewisse Anforderung nicht mehr relevant, da die manuelle Konfliktlösung nicht weiterentwickelt wurde.

3.5.2. Code-Statistik

Eine kurze Statistik:

- Files changed 376
- Lines inserted: 4646
- Lines deleted: 37529

Die vielen gelöschten Files sind auf das Wegfallen der Webapplikation zurückzuführen.

3.5.3. Test

Tests wurden auch wie in der Vorgängerarbeit mit Pytest geschrieben. Bei der Weiterentwicklung wurden die nicht mehr benötigten Test entfernt, neue hinzugefügt, und sichergestellt, das die alten Tests auch noch funktionieren. Im Anhang G befindet sich ein Output der Testausführung.

3.5.4. Testabdeckung

Wie auch in der Vorgängerarbeit wurde die Testabdeckung mit Coverage überprüft. Wie in der Tabelle ersichtlich 3.2 liegt Testabdeckung neu bei 85.80% gegenüber 91% in der Vorgängerarbeit. Hierbei muss beachtet werden, dass auch Tests der Vorgängerarbeit beinhaltet sind.

Modul	Stmts	Miss	Cover
kraken/cli.py	118	45	62%
kraken/collector.py	85	38	55%
kraken/config/programmatic_template.py	12	0	100%
kraken/connectors/connector.py	26	2	92%
kraken/connectors/file_connector.py	37	0	100%
kraken/connectors/filters.py	41	1	98%
kraken/connectors/legacy_connectors/legacy_function.py	26	9	65%
kraken/connectors/legacy_connectors/netbox/netbox_connector.py	122	24	80%
kraken/connectors/legacy_connectors/netbox/netbox_mapper.py	37	0	100%
kraken/connectors/legacy_connectors/powerdns/powerdns_connector.py	58	4	93%
kraken/connectors/legacy_connectors/powerdns/powerdns_mapper.py	57	3	95%
kraken/connectors/mapper.py	6	1	83%
kraken/connectors/sources/source_connectors.py	118	24	80%
kraken/connectors/sources/source_mappers.py	112	6	95%
kraken/connectors/utils.py	11	0	100%
kraken/definitions.py	7	0	100%
kraken/init_kraken.py	26	2	92%
kraken/merger/deep_diff_helpers.py	20	0	100%
kraken/merger/diff.py	11	0	100%
kraken/merger/merge.py	188	13	93%
kraken/models/models.py	121	10	92%
kraken/template_utils.py	22	2	91%
kraken/utils/exceptions.py	3	0	100%
kraken/utils/index.py	3	0	100%
kraken/utils/validator.py	50	3	94%
TOTAL	1317	187	86%

Tabelle 3.2.: Coverage Report

4. Software-Handbuch

Eine Anleitung zur Verwendung von Kraken befindet sich zusätzlich zu diesem Software-Handbuch im Source-Code in der `README.md` Datei.

Grosse Teile des Softare-Handbuchs wurden aus der Vorgängerarbeit übernommen, angepasst und aktualisiert[3, S. 75].

4.1. Installationsanleitung

Der Source-Code von Kraken kann auf dem Gitlab-Server der OST gefunden werden[1]. Der Programmcode wird zudem in einer Zip-Datei mit der Abgabe mitgeliefert.

Die Installation funktioniert gleich wie bei Kraken 1.0, mit der Ausnahme, dass kein Webserver installiert werden muss, und Kraken nun keine Datenbank mehr besitzt.

Für die Installation sind folgende Programme notwendig:

- Python: Mindestens Version 3.8 [2]
- Python Package-Manager pip [3]

Um die CLI Applikation zu installieren sind folgende Schritte nötig:

1. Das mitgelieferte Archiv kraken.zip an einem Ort der eigenen Wahl entpacken und in diesen Ordner wechseln.
2. Eine venv (virtuelle Umgebung) erstellen: `python3 -m venv .venv`
3. Das venv aktivieren:
   Auf Unix-Systemen: `source .venv/bin/activate`
   Auf Windows-Systemen: `.venv\Scripts\activate.bat`
4. Poetry[4] installieren: `pip install poetry`
5. Alle Dependencies von Kraken installieren: `poetry install`

[1] Stand 12.01.2022 `https://gitlab.ost.ch/diktyo/ba21-nettowel-kraken`
[2] `https://www.python.org/`
[3] `https://pypi.org/project/pip/`
[4] `https://python-poetry.org/`

6. Um das CLI zu verwenden, muss das Package kraken lokal via pip installiert werden. Mit dem Punkt nach pip install wird das Projekt, das im aktuellen Verzeichnis gefunden wird, installiert – in unserem Fall also Kraken:
   `pip install .`

Nun kann Kraken mit dem Befehl `kraken` als CLI verwendet werden.

4.2. Benutzeranleitung

Im Ordner `kraken/config` sind eine Kraken-Konfiguration und ein Merge-Template abgelegt.

4.2.1. Konfiguration

Die bereits vorhandene Kraken-Konfiguration enthält bestehende Konnektoren. Diese nehmen jeweils einen Datei-Pfad entgegen, z.B. zu einer CSV-Datei oder einer JSON-Datei. Diese Quellen können mit dem *CombinedMerger* gemerged werden.

Die momentan implementierten Konnektoren benötigen die Angabe eines Filepaths.

Im Merge Template kann auf der Stufe Attribut eine Präsedenz angegeben werden, die die globale Präzedenz überschreibt.

Ein Beispiel-Merge-Template kann z.B. so aussehen:

```
Device:
  dns_name:
    PRECEDENCE: "SourceInventoryConnector"
  device_type:
    PRECEDENCE: "SourceInventoryConnector"
merger: "default"
```

Auf der ersten Ebene wird eine Klasse des Normalisierungsformats angegeben, auf der Zweiten ein Attributname dieser Klasse, dem dann mit dem Attribut `PRECEDENCE` der bevorzugte Connector angegeben werden kann.

Mit der Einstellung `merger` kann man angeben, welcher Merger verwendet werden soll. Im Normalfall sollte der default Merger genügen, wenn jedoch ein Anderer verwendet werden soll, z.B. ein selbstgeschriebener, kann das hier angegeben werden.

Um die Korrektheit dieser Konfigurations-Dateien zu überprüfen, existiert eine Validierung, welche bei jedem Durchlauf die Dateien überprüft. Über die CLI kann man auch

separat überprüfen, ob die Konfigurationen valide sind und bekommt Hinweise, wenn etwas nicht stimmt.

Die bereits vorhandenen Dateien können entweder kopiert werden oder als Vorlage für eine eigene Konfiguration dienen. Mehr zur Verwendung der CLI findet sich im Kapitel 4.2.4.

4.2.2. Filterverwendung

Abhängig von der Kraken-Konfiguration sind nicht alle Filter verfügbar: Ist ein Konnektor nicht Teil der Kraken-Konfiguration, ist es nicht möglich Filter zu verwenden, die nur dieser Konnektor bereitstellt.

Folgende Filter sind im Kraken implementiert und entsprechend verwendbar:

Name des Filters	IP	Inventory	Locations	Prefix	Beispiel
DeviceName	✓	✓	✗	✗	leaf1
LocationName	✗	✗	✓	✗	Basel
DeviceType	✗	✓	✗	✗	Nexus 9300
IPAddress	✓	✗	✗	✗	10.20.0.202/24
DnsName	✓	✗	✗	✗	spine1.kraken.network

Die Filter können für eine AND-Abfrage kombiniert werden - eine OR-Abfrage ist nicht implementiert. Auch muss der Begriff exakt dem Wert in der Netzwerkverwaltung entsprechen (keine LIKE-Abfrage).

4.2.3. Template

Mit dem CLI-Argument `--template`, oder `-t` kann man den Filepath zu einer Python-Datei angeben, die eine Methode mit dem Namen `output_template` enthalten muss, die ein Argument entgegennimmt. Dieses Argument ist jeweils vom Typ Device, da von Kraken jeweils eine Liste von Devices zurückgegeben wird. Die Funktion wird auf jedes Element des Return-Werts angewandt.

4.2.4. CLI

Um die CLI zu verwenden, werden zwei separate Konfigurationsdateien benötigt, welche das Kapitel 4.2.1 Konfiguration ausführt. Sobald diese vorhanden sind, wechselt die Nutzerin in den korrekten Ordner und kann die folgenden Befehle einfach nur kopieren und einfügen. Falls man in einem anderen Ordner ist, müssen die Pfade zum Merge-Template und der Kraken-Konfiguration angepasst werden. Die Erklärung der Befehle ist jeweils direkt in der Kommentarzeile davor vermerkt. Beim Befehl muss beachtet werden, dass die Kraken-Konfiguration (`kraken_config.yml`) immer mitgegeben werden muss. Die Filter-Werte sind Beispiele und müssen durch tatsächliche Werte ersetzt werden.

```
# Infos zu Kraken anzeigen

## Alle Konnektoren anzeigen
kraken show systems
## Alle verfügbaren Filter anzeigen
kraken show filters

# Validierung

## Kraken-Konfiguration validieren
kraken check kc kraken_config.yml
## Merge-Template validieren
kraken check mt merge_template.yml

# Automatische Konfliktlösung

## Kraken mit der Kraken-Konfiguration "kraken_config.yml", einem optionalen Filter
    "device_name=leaf23"
## und optionalem Output der End-Konfiguration in die Datei endconfiguration.json
    ausführen
kraken run -f device_name="leaf23" -o output.json kraken_config.yml
## Kraken mit einem Merge-Template ausführen
kraken run -m merge_template.yml -o output.json kraken_config.yml
## Kraken mit mehreren Filtern ausführen:
kraken run -f device_name="spine1" -f ip_address="1.2.3.4/24" -o test.json\
       kraken_config.yml
## Kraken mit template ausführen
kraken run -t template_file.py kraken_config.yml

# Hilfestellung

## Hilfestellung zu Kraken
kraken --help

## Hilfestellung zu "kraken check"
kraken check --help
```

```
## Hilfestellung zu "kraken show"
kraken show --help

## Hilfestellung zu "kraken run"
kraken run --help
```

Listing 4.1: Beispiele CLI-Verwendung

4.3. Kraken erweitern

Die Erweiterung des Krakens funktioniert im Kraken 2.0 ähnlich wie im Vorgänger. Wer über Python-Kenntnisse verfügt, kann den Kraken-Core erweitern.

4.3.1. Neues System anbinden

Um ein neues System in Kraken einzubauen, sind folgende Schritte, auf welche nach dieser Auflistung genauer eingegangen wird, nötig:

1. Neue Konnektor-Klasse implementieren: Ordner `kraken/connectors`
2. Mapper implementieren: Ordner `kraken/connectors`
3. (optional) Models und `__eq__` Funktionen anpassen.
4. (optional) Filter implementieren: Ordner `kraken/connectors`
5. Konnektor zum Index hinzufügen: File `kraken/utils/index.py`
6. Konnektor zum Validator-Schema hinzufügen: File `kraken/utils/validator.py`

Je nach Anwendungsfall muss noch ein neuer Merger implementiert werden, wenn die bestehenden Merger die benötigte Logik nicht enthalten. Zum Beispiel im Falle der IP-Prefixe ist das der Fall, da die Zuweisung von IP-Addresse zu IP-Prefix speziefische Logik benötigt.

Neuer Konnektor Jeder neuer Konnektor ist ein weiteres Package unter `kraken/connectors`; die Klasse selbst leitet von der `AbstractConnector`-Klasse ab. Die Abbildung veranschaulicht den Aufbau: Die neue Klasse ist in einer Datei mit der Endung `_connector.py` und die zu implementierenden Methoden sind `get_all` und `get_many_by_query`. Die Konstruktor-Argumente der Konnektoren werden durch das Kraken-Config befüllt. Im Konfig gibt es zu jedem Konnektor ein Dict namens `arguments`. Diese Key-Values werden als Parameternamen und Wert an den entsprechenden Konstruktor übergeben.

Die in dieser Arbeit implementierten Konnektoren wurden im Package `sources` untergebracht. Somit kann man klar unterscheiden welche man selbst implementiert und welche schon existieren.

Die Konnektoren der Vorgängerarbeit heissen `netbox` und `powerdns` und wurden im Ordner `connectors` gelassen, weil viele Tests der Vorgängerarbeit von ihnen abhängen. Diese Tests testen Funktionalität, die nicht vom Normalisierungsformat abhängt, weswegen sie immer noch relevant sind. Aus Zeitgründen wurden nur die nötigen Tests angepasst, da somit mehr Zeit für die neue Implementation bleibt.

Neuer Mapper Um die Daten zu normalisieren, benötigt der neue Konnektor einen Mapper, das heisst, eine neue Klasse leitet von der abstrakten `Mapper`-Klasse ab, die zum Konnektor gehört. Die Datei trägt die Endung `_mapper.py` und ist im gleichen Package wie die Konnektor-Klasse. Deren einzige Methode `map_query` bringt die Rohdaten eines Systems auf das Normalisierungsformat. Das Normalisierungsformat sind die Datenmodelle in der Datei `models.py`. Wichtig dabei ist, dass Kraken vom Device aus arbeitet. Das heisst, wo möglich soll vom Konnektor eine Liste Devices zurückgegeben werden und dementsprechend vom Mapper ein Device mit verschachtelten Elementen. Dann kann der `DeviceMerger` verwendet werden um den Merge zu vollziehen.

Filter Filter funktionieren damit, dass man als Anwender nach dem Attribut des Normalisierungsformats suchen kann und im Konnektor dann die Quelle oder die Daten der Quelle mit deren eigenem Attribut zu durchsuchen.

Um einen Neuen Filter zu implementieren, muss eine Klasse von `AbstractFilter` erben und das Attribut `consumer_keyword` setzen

Models Wenn mehr Daten im Normalisierungsformat aufgenommen werden sollen, müssen die Models angepasst werden. Dabei muss beachtet werden, dass bei Listen wegen der Vergleichs-Library DeepDiff zwingend Dicts verwendet werden müssen. Zudem kann auch die `__eq__` verändert werden um den Vergleich der Objekte anzupassen. Wenn

eine neue Model-Klasse hinzugefügt wird muss sie noch in der `__init__.py` des Packages registriert werden.

Index ergänzen Kraken muss wissen, wo ein System bzw. Konnektor innerhalb der Code-Basis zu finden ist, um von der Kraken-Konfiguration die korrekte Klasse zu instanzieren. Dazu muss der Index in `index.py` ergänzt werden.

Validator ergänzen Der Validator prüft die Korrektheit der Kraken-Konfiguration und des Merge-Templates - er prüft auch, ob nur gültige Systeme in der Konfiguration stehen. Deswegen muss in der Datei `definitions.py` zum `KRAKEN_CONFIG_VALIDATION_SCHEMA` ebenfalls das `Name`-Attribut des neuen Konnektors hinzugefügt werden, und zwar unter `connectors|schema|schema|name`.

4.3.2. Tests laufen lassen

Neuer und bestehender Code wird mit tox und pytest getestet. Durch die Kraken-Installation sollten pytest und tox bereits installiert sein, ansonsten sind die entsprechenden Schritte in der folgenden Anleitung für Unix-Systeme angegeben.

1. Darauf achten, dass venv aktiviert ist.
2. tox und pytest im Projekt-venv installieren:
   `pip install tox pytest`
3. poetry shell aktivieren, oder die folgenden Befehle mit `poetry run` ausführen:
   `poetry shell`
4. Pytest-Tests aus dem Root-Verzeichnis des Projekts starten:
   `tox -e pytest`
5. Korrekte Formatierung prüfen:
   `tox -e black`
6. Korrektes Typing prüfen:
   `tox -e mypy`
7. Alternativ: Alle Tests ausführen (pytest, black, mypy):
   `tox`

5. Fazit

5.1. Persönliches Fazit von Felix Kubli

Diese Arbeit war eine grosse Herausforderung für mich, da ich mich mit Netzwerken in einem recht unbekannten Gebiet bewege und aus dem Bereich der Softwareentwicklung komme. Zudem arbeite ich zum ersten Mal mit Daniel zusammen. Die Zusammenarbeit verlief auch nicht reibungslos. Gerade am Anfang der Arbeit hatten wir Mühe mit der recht offen formulierten Aufgabenstellung, da wir nicht wussten, was die Ideen und Denkweisen des jeweils Anderen sind und was wir von ihm erwarten können. Durch unsere verschiedenen Hintergründe hatten wir auch verschiedene Herangehensweisen. Deswegen kam es zu Missverständnissen und Meinungsverschiedenheiten, die uns Zeit kosteten. Im Rückblick hätte ich am Anfang des Projekts viel genauer nachgefragt, was genau der Output der Arbeit sein soll, was zu dieser Zeit nicht realisierte.

Zudem habe ich bei und nach den Interviews erkannt, dass es für das Führen der Interviews einen Netzwerkengineer bräuchte, da die Beurteilung der Relevanz der Antworten durch fehlendes Wissen und fehlende Erfahrung im Netzwerkbereich erschwert wurde. Das betraf auch die Analyse der Umsysteme, da dadurch eine Art Informationsüberfluss entstand.

Es wurden in diesem Projekt keine direkten Verantwortlichkeiten definiert, jedoch im Verlauf des Projekts kümmerte ich mich mehr um die Implementation, während Daniel sich mehr um den Teil „Data-Engineering“, das Beispielnetz und um die Marktanalyse kümmerte.

Trotz der Schwierigkeiten, habe ich in dieser Arbeit einen wertvollen Einblick in das Management von Netzwerken erhalten, da ich bis jetzt kaum wusste wie diese in der Praxis verwaltet werden. Zudem konnte ich gut entnehmen in welche Richtung sich die Netzwerkverwaltung bewegt und welche Hürden bei der Netzwerkautomatisierung von Firmen zu überwinden sind.

In dieser Arbeit hätte ich vor allem mehr Zeit zum Implementieren gewollt um einen umfangreicheren Beitrag zu dieser Applikation leisten zu können, was leider zeitbedingt und durch den Verlauf der Arbeit nicht möglich war. Trotzdem bin ich zufrieden mit unserer Leistung, da wir unter diesen Umständen gut gearbeitet haben und schlussendlich die bei der Zwischenpräsentation definierten Ziele erreichen konnten.

5.2. Persönliches Fazit Daniel Steudler

Es war ein Risiko, die Bachelorarbeit mit Jemandem zu machen den man zuvor noch nicht kennt. Felix und ich haben uns noch nicht gekannt.

Bei Kraken nicht bei null anzufangen war Fluch und Segen zugleich. Ich hätte gerne mehr Zeit mit Aufräumen von Kraken verbracht. Im Kontext der Arbeit wäre das aber nicht Zielführend gewesen. Ausserdem hätte ich gerne mehr Zeit beim Implementieren investiert. Ich habe das Gefühl, das uns die Interviews zwar geholfen hat die Domäne zu verstehen. Aber sie haben auch viel Zeit und Energie gekostet. In einem Greenfield-Approach hätten mehr architektonische Entscheidungen getroffen werden können.

Es hat sich im Laufe der Arbeit herausgestellt das Felix und ich in gewissen dingen eine andere Philosophie verfolgen. Dies hat zu vielen Diskussionen geführt. Und ein Konsens zu finden war nicht immer einfach. Ich denke auch das hier der das unterschiedliche Alter und der unterschiedliche Backgrounds eine Rolle spielt.

Die Zwischenpräsentation hat mir vor Augen geführt das wir am Anfang sehr auf die Konfliktlösung von zwei Quellen fixiert waren. Und das als Kraken verstanden haben.

Die Interviews haben unsere Annahmen dann widerlegt. Was es schwierig gemacht hat für uns etwas zu finden, was sich mit den Interviews und der Aufgabenstellung vereinen lässt.

Durch das mehrmalige pivoten unseres Fokus hat sich der Bericht auch immer wieder verändert. Es musste immer wieder ein neuer Roter faden gefunden werden, was Zeit und Energie gekostet hat.

Anhang

Glossar

NETCONF NETCONF ermöglicht die Model-basierte Konfiguration. Daten werden dabei als XML übertragen und entsprechen jeweils einem YANG-Model. 11, 35, 36, 39

OpenConfig OpenConfig ist ein Standard für die platformunabhängige Konfiguration von Netzwerkgeräten. OpenConfig ist in YANG definiert.. 11–13, 64

RADIUS Remote Authentication Dial-In User Service (RADIUS) ist ein Protokoll, das für die Authentifizierung und Authorisierung verwendet wird.. 24, 28, 30

RESTCONF RESTCONF ermöglicht ebenso wie NETCONF die Model-basierte Konfiguration von Netzwerkgeräten. Die Daten werden dabei im JSON-Format übertragen und entsprechen jeweils einem YANG-Model.. 27

Single Source of Truth Eine Single Source of Truth ist eine Data Source, die als alleinige Wahrheit gehandelt wird. 5, 93

YANG YANG ist eine Definitionssprache, die verwendet wird, um Konfigurationsmodels im Netzwerkbereich zu definieren.. 11, 12, 27, 35, 36, 39

Akronyme

B2B Business to Business. 31, 34, 35

B2C Business to Client. 31

CLI Command Line Interface. 2, 23, 41, 60, 74, 81–84

CMDB Configuration Management Data Base. 28, 30

CRM Customer Relation Management. 24, 32–34

DCIM Data Center Infrastructure Management. 21, 22

DHCP Domain Host Configuration Protocol. 24, 27, 29

DNS Domain Name System. 27, 29

DSLAM Digital Subscriber Line Access Multiplexer. 8, 35

IPAM IP Adress Management. 21–24, 28–30

ISP Internet Service Provider. 8, 31, 38

LAN Local Area Network. 8

MAC Media Access Control. 73

ORM Object-Relational Mapping. 54, 64

OS Operation System. 27, 29

OST Ostschweizer Fachhochschule. 6, 25, 81

RANCID Really Awesome New Cisco config Differ. 21, 35

SDK Software Development Kit. 12

SNMP Simple Network Management Protocoll. 35

SoT Source of Truth. 9, 24, 31

SSoT Single Source of Truth. 5, 6, 8, 9, 93, *Glossary:* Single Source of Truth

VLAN Virtual Local Area Network. 8, 9, 27, 29, 31, 33

VRF Virtual Rooting and Forewarding. 27

WAN Wide Area Network. 8

Anhang A.

Literatur

[1] S. McGillicuddy, „A Network Source of Truth Promotes Trust in Network Automation,“ S. 7, Mai 2020.

[2] B. Claise. „Network Automation: The costly Data Models Integration and Mediation – Benoît Claise.“ (18. Mai 2021), Adresse: `https://www.claise.be/network-automation-and-the-costly-data-model-integration-mediation/` (besucht am 13. 11. 2021).

[3] M. Sieber und J. Fritsche, „Kraken - Ein Mediator Für Datenkonflikte in Der Netzwerkverwaltung,“ Bachelorarb., OST Ostschweizer Fachhochschule, 2021.

[4] „NetBox Documentation.“ (4. Jan. 2022), Adresse: `https://netbox.readthedocs.io/en/stable/` (besucht am 04. 01. 2022).

[5] J. Edelman. „Why did Network to Code Fork NetBox?“ Network to Code. (), Adresse: `https://blog.networktocode.com/post/why-did-network-to-code-fork-netbox/` (besucht am 25. 10. 2021).

[6] *Nautobot-Golden-Config*, Nautobot, 8. Okt. 2021. Adresse: `https://github.com/nautobot/nautobot-plugin-golden-config/blob/bf818b596852962b3f782f9c756b7cfacd71bf77/docs/navigating-compliance.md` (besucht am 15. 10. 2021).

[7] „StableNet® - Unified Network & Services Management,“ StableNet by Infosim. (), Adresse: `https://www.infosim.net/stablenet/` (besucht am 12. 10. 2021).

[8] L. Daigle. „Every network is a snowflake | TechArk,“ Better Internetworking through Collaboration. (19. Feb. 2018), Adresse: `https://www.techark.org/every-network-is-a-snowflake/` (besucht am 25. 10. 2021).

Anhang B.

Abbildungsverzeichnis

Anhang C.

Tabellenverzeichnis

Anhang D.

Listings

Anhang E.

Projektmanagment

E.1. Einführung

E.1.1. Zweck

Dieser Projektplan bildet die Grundlage für die Bachelorarbeit „Kraken 2.0“.

E.1.2. Gültigkeitsbereich

Dieses Dokument gilt in seiner ersten Phase als Grundlage für die Bachelorarbeit und wird während der Projektlaufzeit fortlaufend aktualisiert. Es gilt jeweils die aktuellste Version des Dokuments.

E.2. Projektübersicht

Kraken ist eine Software, die für die Netzwerkautomation eingesetzetzt werden soll.

E.2.1. Aufgabenstellung

E.2.2. Ziel

Wir setzen uns als Projektteam folgende Ziele:

- Bestehen der Bachelor-Arbeit.
- Untersuchung und Beantwortung der theoretischen Fragestellung.
- Untersuchung und Beantwortung der Praktischen Fragestellung.
- Bestehende Software *Kraken* verbessern

E.2.3. Lieferumfang

Folgende Inhalte werden am Ende des Projekts abgeliefert:

- Projektplan
- Bericht
- Allfälliger Quellcode
- Poster A0
- Abstract
- Persönliche Berichte / Fazit
- Erklärung zur eigenständigen Durchführung der Arbeit
- Einverständniserklärung Publikation auf eprint.hsr.ch

E.3. Projektorganisation

E.3.1. Übersicht

Das Team besteht aus Felix Kubli und Daniel Steudler. Aufgrund der geringen Teamgrösse, sowie dem breiten Umfang des Projekts, verzichten wir auf die Verteilung konkreter Rollen und Verantwortlichkeiten.

- Felix Kubli: felix.kubli@ost.ch
- Daniel Steudler: daniel.steudler@ost.ch

E.3.1.1. Externe Schnittstellen

- Projektbetreuer: Beat Stettler, beat.stettler@ost.ch
- Betreuer: Martin Stypinski, Méline Sieber

E.3.1.2. Besprechungen

Es finden jeweils Besprechungen am Montag um 15:30 bis 16:30 statt, um den Fortschritt mit dem Betreuerteam zu teilen.

E.3.2. Dokumentenhaltung

Die Projektdokumentation der Bachelorarbeit sowie auch der Projektplan befindet sich als LaTeX Dokument auf dem Gitlab der OST. Gitlab erlaubt uns die Dokumentationen wie Code zu behandeln und versionieren. Ausserdem ermöglicht uns LaTeX, dass immer ein Dokument mit Druckqualität bereitsteht, da die Dokumente bei jedem push neu gebuildet werden.

E.3.3. Issue Tracking

Das Issue-Tracking findet im Gitlab der OST statt. Die Issues werden im Projekt des Schlussberichts erfasst. Die Issues zu Kraken als Software werden im GitLab Projekt der Software erfasst, somit ergibt sich eine bessere Übersicht. Ein Kanban Board mit folgenden Lanes wurde angelegt: Open, Planned, In Progress, Waiting, Closed. Somit ist der Status eines Tickets immer einsehbar. Es bestehen Überlegungen das Issue Tracking im Entwicklungsteil in das Software Projekt selbst auszulagern.

Open Unser Backlog

Planned Tickets welche bearbeitet werden können

In Progress Tickets in Arbeit

Waiting Tickets welche auf Externe Events Warten. Auf was genau gewartet wird, wird mittels Label spezifiziert.

Closed Abgeschlossene Tickets.

Für die Tickets wurden ausserdem verschiedene Labels erstellt. Dies dient der Übersicht bei der Planung und wird für die Auswertung verwendet.

E.3.4. Time Tracking

Als Time Tracking Lösung wird Clockify verwendet. Das Team erhofft sich dadurch eine einfachere Auswertung der Zeitaufwände gegenüber dem Gitlab eigenen Time Management.

Folgende Clockify Tags werden verwendet:

Betreuersitzung Sitzung mit Betreuern.

Organisation Projektorganisation innerhalb des Teams inklusive Sitzungen.

Planung Sprint Planung und Projektplanung.

Recherche Grundlagenrecherche, Befragungen

Konzeptionierung Erarbeiten des Konzepts

Interviews Zeit für das Führen und die Nach- und Vorbereitungen

Prototyping Explorative Recherche, Testen von Hypothesen.

Zwischenpräsentation Vorbereitung, Durchführung und Nachbereitung

Umsetzung Implementationsarbeit

Dokumentation schreiben Arbeiten an der Dokumentation.

E.4. Managementabläufe

In dieser Arbeit wurde sich anfangs für das Scrum+ Modell entschieden, welches in der HSR in der SE2 Vorlesung gelehrt wird. Das ist eine Kombination aus Scrum und dem Rational Unified Process (RUP), der eine Aufteilung in Phasen vorsieht. Eine Planung mit den entsprechenden Phasen wurde zu Projektbeginn erstellt.

Aufgrund der recht offenen Aufgabenstellung, der explorativen Arbeit und der Integration von Interviews wurde dann jedoch Srum an sich gewählt, da ändernde Anforderungen beantwortet werden können müssen, was mit einer strikten Einteilung in Phasen nicht gut möglich ist. Um den Backlog besser zu organisieren und zu tracken, wurde in Gitlab mit einem Kanban gearbeitet.

Was jeweils in einem Sprint bearbeitet wird, wird anfangs Sprint beschlossen. Sollte es Unerwartetes geben, wird das besprochen und der Sprint angepasst. Am Ende eines Sprints findet eine kurze Retrospektive statt. Das Wöchentliche Betreuer Meeting legt dann den Fokus für den nächsten Sprint fest. Die Festgelegten Punkte werden dann in Arbeitspaketen abgebildet.

E.4.1. Zeitmanagement

Für das Projekt stehen insgesamt 17 Wochen zur Verfügung. Jeder Student erhält 12 European Credit Transfer System-Punkte (ECTS) für die Bachelor Arbeit. 1 ECTS entspricht einem Arbeitsaufwand von 30 Stunden. Total stehen somit 720 Arbeitsstunden zur Verfügung:

$$2 \text{ Projektmitglieder} \cdot 30 \text{ Stunden} \cdot 12 \text{ ECTS} = 720 \text{ Arbeitsstunden}$$

Pro Woche sind das 42.36 Stunden als Team oder 21h pro Person.

E.4.2. Zeitauswertung

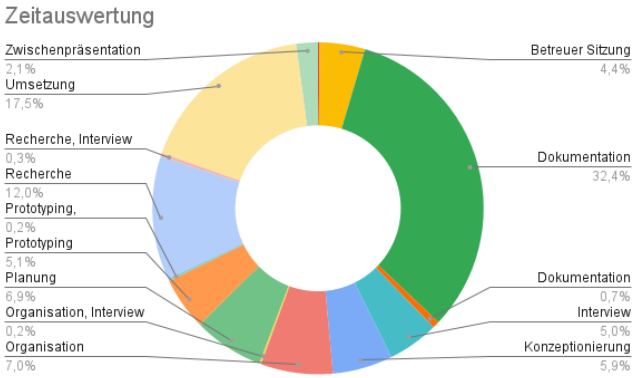


Abbildung E.1.: Zeitauswertung

Die Zeitauswertung wurde mit Clockify gemacht.

Total wurden 807 Stunden in Clockify geloggt. Davon fallen 437:27:34 Stunden auf Daniel, und 370:32:15 Stunden auf Felix.

E.4.3. Planung Soll

Planung zu Beginn der Arbeit:

- Inception: Bis 28.9.2021
  - Scope und Grobe Zielsetzung
  - Idee von Kraken
  - Kraken ausprobieren und verstehen
- Elaboration: Bis Woche 8
  - Theoretische Skizzen
    - Wie stellen wir uns vor, dass Datenformate funktionieren?
    - Wie soll die Applikation damit umgehen?
  - Analyse von Austauschformaten: Stärken und Schwächen
  - Analyse der Datenquellen

  - Auseinandersetzung mit Vergleichsalgorithmen
  - Interviews mit Industriepartner
    * Was sind die Painpoints?
    * Welche Datenquelle werden wie verwendet?
  - Funktionale Prototypen bauen
  - Requirements sind definiert
- Construction: Bis Woche 13
  - Verfeinern der Punkte aus Elaboration
  - Theorie ausarbeiten
  - Fokus auf Hauptformat
    * Entweder Austauschformat
    * Oder Industrieformat
  - Implementation ist abgeschlossen
- Transition: Bis Woche 17
  - Dokumentation

E.4.4. Planung ist

- Inception: Bis 28.9.2021
  - Scope und Grobe Zielsetzung
  - Idee von Kraken
  - Kraken ausprobieren und verstehen
- Elaboration: Bis Woche 10
  - Domänen wissen aufbauen.
  - Theoretische Skizzen
    * Wie stellen wir uns vor, dass Datenformate funktionieren?
    * Wie stellen wir und vor, dass die Applikation damit umgeht?
  - Analyse von Austauschformaten: Stärken und Schwächen

  - Analyse der Datenquellen
  - Auseinandersetzung mit Vergleichsalgorithmen
  - Interviews mit Firmen
    * Was sind die Pain-Points?
    * Welche Datenquelle werden wie verwendet?
    * Welche Daten werden wie verwendet.
  - Funktionale Prototypen bauen
  - Requirements sind definiert
- Construction: Bis Woche 14
  - Verfeinern der Punkte aus Elaboration
  - Daten modell am Protoypen austesten
  - Theorie weiter ausarbeiten
  - Eigenes Datenmodell in Kraken implementieren
    * Kraken Core austauschen
    * Konnektoren anpassen und neue schreiben
    * Diffing anpassen
    * Merge Strategien überarbeiten
    * Neue Merge-Strategie hinzugfügen
  - Implementation ist abgeschlossen
- Transition: Bis Woche 17
  - Fein-Tuning von Kraken
  - Dokumentation
  - Marktanalyse Abschliessen

E.5. Risikomanagement

Nr	Titel	Beschreibung	Schaden total [h]	Eintritts-W	Gewichteter Schaden
R1	Interviews verzögern sich	Die Interviews werden mit einem Industriepartner durchgeführt. Diese Können sich verzögern	16	10%	1,6
R2	Interviews liefern nicht das gewünschte Resultat	Die erhaltenen Informationen ensprechen nicht den erwarteten Informationen	16	30%	4,8
R3	Datenaufbereitung ist komplizierter als gedacht	Die vom Industriepartner gelieferten Daten erweisen sich als schwierig	16	30%	4,8
R4	Research von Formaten dauert länger		16	50%	8
R5	Datenformate entpuppen sich als Komplexer als angenommen	Der Vergleich von verschiedenen Datenquellen ist schwieriger als erwartet	16	60%	9,6
R6	Konzept benötigt mehr Zeit	Der Entwurf eines Programm Konzepts muss mehrmals verworfen werden und braucht somit mehr Zeit als erwartet	16	50%	8
R7	Scope der Arbeit ist zu gross angesetzt	Die Arbeit im Anfangsstadium birgt viele Unsicherheiten, was dazu führt, dass es schwer ist eine Realistische grösse in Scope zu finden	16	40%	6,4
R8	Verständniss der Vision von Kraken benötigt länger	Ein gutes Verständniss von Kraken und dessen Vision dient als Grundlage für gute Interviews	5	5%	0,25
R9	Gesundheitliche Probleme	Unerwartetes Auftreten von Gesundheitlichen Problemen	24	20%	4,8
R10	Kommunikationsschwierigkeiten innerhalb des Teams		8	20%	1,6

Tabelle E.1.: Risikotabelle

E.5.1. Mitigationen

Nr	Vorbeugung	Verhalten beim Eintreten
M1	Genügend Aufgaben finden, die man ohne Interview Informationen erledigen kann	Planung überarbeiten
M2	Gute Interviewfragen erarbeiten	Neue Planung und Erwägung von Möglichkeiten
M3	Reserve einberechnen	Scope anpassen
M4	Relevanz der Formate im Vorlauf beurteilen	Scope anpassen
M5	Reserve einberechnen	Scope anpassen
M6	Reserve einberechnen	Scope anpassen
M7	Fortlaufendes Einschätzen des Aufwands	neu Planen und das Scope anpassen
M8	Durchlesen der Vorgängerarbeit und Vison erstellen	Scope anpassen
M9		Scope anpassen
M10	Regelmässiger Autausch im Team	Besseren Kommunikationsprozess finden

Tabelle E.2.: Mitigationen von Risikos

E.5.2. Eingetretene Risiken

Im Verlauf des Projektes sind folgende Risiken eingetreten.

Der Total geschätzte gewichtete Schaden betrifft 49.85 Stunden.

Der effektiv eingetretene Schaden betrifft hingegen ca. drei Wochen. In Stunden sind das ca. 124 Stunden.

$$2 \cdot 42\text{h} + 5 \cdot 8\text{h} = 124\text{h}$$

Die Differenz davon ist 74.15h.

E.5.2.1. R1 Interviews verzögern sich

Da wir von der Stadt Zürich keine Antwort erhalten haben, haben wir nach weiteren Interviewpartnern gesucht und in der Zwischenzeit Nachforschungen betrieben.

Durch diese getroffenen Massnahmen entstand hier kein Schaden.

E.5.2.2. R2 Interviews liefern nicht das gewünschte Resultat

Bei den Interviews haben wir nicht die Informationen bekommen, die wir uns gewünscht hätten. Als Folge darauf wurde untersucht, ob der Fokus von Kraken verändert werden muss.

Insgesamt ist dadurch der Schaden von ca. einer Woche entstanden.

E.5.2.3. R5 Datenformate entpuppen sich als komplexer als angenommen

Der zuerst verfolgte Ansatz war mit der Verwendung von OpenConfig in Kombination mit dem YANG-Datentyp. Der Umgang mit YANG und OpenConfig hat sich als zu komplex herausgestellt und wurde deshalb von uns nicht weiter verfolgt.

Hier betraf der Schaden auch etwa eine Woche, da eine Woche in diese Nachforschungen und diesen Ansatz investiert wurden.

E.5.2.4. R6 Konzept benötigt mehr Zeit

Als Folge der eingetretenen Risiken R2, R5 und R8 trat dieses Risiko ebenfalls ein, da das Programmkonzept basierend darauf mehrfach überarbeitet werden musste. Aus diesem Grund musste das Scope der Entwicklung reduziert werden.

E.5.2.5. R8 Verständnis der Vision von Kraken benötigt länger

Der Zweck von Kraken wurde missverstanden, obwohl dem vorgebeugt wurde. Es gab Missverständnisse bei der Vision und dem Ziel von Kraken, obwohl dem vorgebeugt wurde.

Dadurch entstand ein Schaden von ca. 2 Tagen.

E.5.2.6. R10 Kommunikationsschwierigkeiten innerhalb des Teams

Unter den Autoren kam es zu Meinungsverschiedenheiten, was die Marschrichtung und die Idee von Kraken betraf. Dadurch konnte nicht gut geplant und gearbeitet werden, da hierfür eine Einigung nötig ist.

Dadurch entstand ein Schaden von ca. 3 Tagen.

E.6. Infrastruktur

- Gitlab: Gitlab für Versionsverwaltung und CI/CD
  - CI/CD
  - Wiki
  - Issue Tracker
  - Docker Registry
- Clockify als Zeiterfassungstool
- Google Drive: Kollaborativer Austausch. Für Entwürfe und Protokolle.
  - Google Docs
  - Google Sheets
- Microsoft Teams zur Teamkommunikation und zum Austausch mit dem Betreuerteam

- Zotero zur Quellenverwaltung[1]
- Miro für das Brainstorming.[2]

E.6.1. Gitlab

Gitlab spielt für uns in diesem Projekt eine zentrale Rolle. Sowohl der Source-Code der Software, als auch der Source-Code der Dokumente liegt in einem GitLab Repository. Mittels CICD Pipelines werden automatisiert Tests durchgeführt. Die Dokumente werden jeweils auch direkt kompiliert.

E.7. Qualitätsmassnahmen

Dieser Abschnitt erklärt, wie wir die Arbeitsqualität für die gesamte Dauer des Projekts sicherstellen. Die Qualitätsanforderungen an die Software werden in diesem Abschnitt jedoch explizit nicht abgedeckt, denn diese werden in den nicht-funktionalen Anforderungen beschrieben.

E.7.1. Generelle Qualitätsmassnahmen

- Regelmässige Besprechungen (innerhalb des Teams, als auch mit dem Betreuer) stellen sicher, dass die aktuellen Fortschritte und Rückschläge zeitnah ausgetauscht werden.
- Es findet jeweils eine Retrospektive der letzten Woche statt.
- Wenn nötig werden Refactorings gemacht.
- Eine Übersichtsseite in Gitlab beinhaltet Direktlinks zu allen wichtigen Dokumenten. Das Wiki wird fortlaufend aktualisiert.
- Beim Erstellen der Arbeitspakete in GitLab nutzen wir bei Bedarf die Möglichkeit, digitale Checklisten in der Arbeitspaketbeschreibung („Description“) zu erstellen, um den Fortschritt des Arbeitspakets festzuhalten.

[1] `https://www.zotero.org`
[2] `https://www.miro.com`

E.7.2. Git-Qualität

Grundsätzlich verwenden wir das Vieraugenprinzip für Mergerequests. Mit drücken auf Merge bestätigt das Teammitglied die Qualität des Mergerequestes. Wir verwenden das Github-Flow model welches auf Featurebranches setzt.

- Ein Feauturebranch soll kurzlebig sein
- Ein Feauturebranch hat einen begrenzten Scope
- Ein Featurebranch hat das Namingschema <User>/description or <User>/<IssueNr>
- Ein Featurebranch mit vielen Commit wird zuvor manuel rebased, und die beteiligten Commits gesquashed. Ein Mergerequest sollte maximal 3 Commits haben.
- Ein merge request soll nur etwas fixen oder hinzufügen. Dies stellt sicher das ein Commit revertable ist.

Eine richtig formulierte Git-Commit-Subject-Line sollte sich als kompletten folgenden Satz lesen lassen Ïf applied, this commit will <subject>". Die Commit Message endet ohne Punkt und der erste Buchstabe ist Grossgeschrieben. Falls möglich soll in der Commit Subject Line auch noch das Issue erwähnt werden: Ädded Readme file #34"

E.7.3. Code-Qualität

Folgende Massnahmen werden getroffen um die Code-Qualität sicher zu stellen:

- Black als Code-Formatierungstool
- Pytest für Unittests
- Unittests und Integrationtests: Es wird eine Test-Coverate von 65% angestrebt.
- Bei Merges wird das 4-Augen-Prinzip angewendet
  - Wurde ein Arbeitspaket abgeschlossen, so wird das Review des Codes durch ein anderes Projektmitglied durchgeführt.
  - Der Ersteller eines Merge-Requests merged nie seinen eigenen Merge-Request
  - Der Merge Button wird vom Reviewer als Zeichen eines durchgeführten Reviews gedrückt.
- Continous Integration in GitLab stellt duch Linter und Tests sicher, dass gepushter Code den Code- und Testrichtlinien entspricht.

E.7.4. Testing

- Die bereits bestehenden Unittests werden erweitert.

E.8. Rückblick

Die Interviews haben mehr Zeit in Anspruch genommen als erwartet. Zudem haben sie nicht die erhofften Antworten geliefert. Somit konnte nicht nach einem klaren Phasen-Modell vorgegangen werden, weshalb der Wechsel zu Scrum erfolgte. Da spielte auch das Verständnis von Kraken eine Rolle, weil der Anwendungszweck durch die Interviews nicht klarer wurde und in der Vorgängerarbeit einen Fokus auf der Konfliktlösung hat, auch wenn Kraken eigentlich als Datenbasis für Konfiggeneratoren fungieren soll. Das führte wiederum dazu, dass nicht auf lange Sicht geplant werden konnte, da damit gerechnet werden musste, dass die gezogenen Erkenntnisse wieder invalidiert werden würden.

Anhang F.

Dokumente

F.1. Interview Fragebogen

Interview Fragebogen

Ablauf

40min	Fokus auf was ist, Prozess und Setup kennen lernen
20min	Elevator Pitch und herausfinden ob man das Problem lösen kann

Intro

Wir sind Bachelor Studenten der OST und versuchen herauszufinden wie im Netzwerkbereich gearbeitet wird.

Fragen

Grundsätzlich Netzwerkverwaltung.
Grundsätzlich interessieren wir uns dafür wie Migros eine Site Managed. Von der Konzeptionierung bis zur Aufschaltung. (Was für Probleme können wir lösen)

Zur Einordnung des Interviewpartners
In welchem Umfang werden bei Ihnen ungefähr Geräte und Konfigurationen verwaltet? (grobe Grössenordnung)

- Gesamtheitlich
- Pro Site
- Wie ist eine Site definiert?
- Haben Sie ein Big Picture (Logisch oder Grafisch) des Netzwerks, gesamtheitlich und Pro Site?-

Wie sieht der Derzeitige Workflow aus: (Gerne auch Aufzeichnen)

- **Beim Aufschalten einer Site**
  - **Was für Informationen benötigen Sie?**
  - **Woher kommen die Informationen und wie werden diese vereinigt (Tool, von Hand)?**
  - **Inventar, IPAM, Routing, Topologie, User-Management, Security, Berechtigungen**
- **Beim Change einer Site?**
- **Beim Debuggen einer Site?**
- Wie werden die Daten verifiziert? Wie oft?
- Werden Prozesse aktiv verbessert? Mit oder ohne Metriken?
- Tribal-Knowledge Bekämpfung und Busfaktor
- Backup

- Disaster Recovery
- Versionierung

Konsistenz und Ambiguity

- Bei Problemen mit einem Gerät: Wie findet man heraus, in welchem Config das Gerät sich befindet?
- Sind Konventionen schriftlich festgehalten. Wie hoch ist die Time to Comprehend?
- Wo hat es versteckte Informationen? (DNS, IP's)
- Wie wird Konsistenz sichergestellt und Ambiguity vermieden?

Welche Arten von Problemen treffen Sie beim Zusammenführen von Informationen an?

- Pain Points?
- Wie gehen Sie um mit Namenskonventionen?
- Redundanzen?
- Aktiv oder Passiv (Deterministisch oder im Kopf)
- Unberechenbarkeit
- Manipulationssicherheit (Wer bearbeitet die Configs? Wer darf?)
- Inkonsistenz beim Naming
- Ambiguous Naming

Wie sieht Netzwerk Toolchain aus, vor allem im Betracht auf Erfassung und Validierung der Netzwerk Konfiguration.

- Konfiguration und Verwaltung von Netzwerkgeräten
- Gilt eine Quelle als Source of Truth?
- Wie gehen Sie mit Konflikten in den Infos und dem Netzwerk um?
- Datenquellen?
- Wie oft wird nach verwaisten Informationen gesucht? Wie wird damit umgegangen?

Elevator Pitch

- Welche Problemstellung wollen wir mit unserem Programm lösen?
    - Kombination verschiedenen Inventur Quellen (Excel, CiscoDNA, etc) zu einer.
    - Informationen in unterschiedlichen Quellen sind überlappend müssen aber nicht. Erkennen von Konflikten.
    - Kreation einer "Masterconfigs"
    - Ein Schritt näher zur Netzwerk Automation.
    - Vergleich von Desired zu Actual. (Zukunft)
    - Device Spezifische Konfigs generieren (Zukunft)
    - Information über Teilgebiete, Big Picture (Zukunkft)

Nach dem Gespräch: Wichtig; einen Ansprechpartner für weitere Fragen festlegen/abmachen.

Nachbearbeitung

F.2. Elevator Pitch

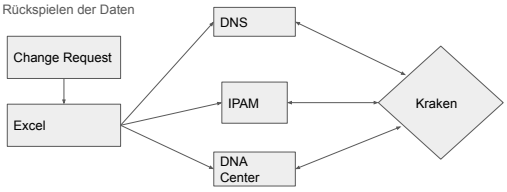
Kraken

Probleme

- Manuelle Tasks sind fehleranfällig
- Je mehr Quellen desto mehr Inkonsistenzen
- Informationen sind verteilt.
- Änderungsverifikation

Netzwerk Inventar

- Kraken: Verschiedene Quellen => “Single Source of Truth”
  - Import aus Datenquellen
  - Aggregation der Daten
  - Konflikterkennung
  - Rückspielen der Daten



Vision

- Eine Single Source of Truth
  - Merge von Daten aus Inventar Datenquellen
  - Big Picture des Netzwerks
- Konflikterkennung

F.3. Gespräch mit Migros

Information über die Firma

Firma:	Migros Genossenschaftsbund
Grösse:	50k Network Devices
Sektor:	

Gesprächsteilnehmer

Kürzel	Name
Felix Kubli:	OST
Daniel Steudler:	OST
Martin Stypinski:	Betreuer OST
Simon Hauri:	MGB
Oliver Fischer:	MGB
Luca Heger:	MGB

Gespräch vom 13.10.2021. Das Gespräch wurde an der Heinrichstrasse 216, 8031 Zürich geführt und aufgezeichnet.

FELIX KUBLI: Wir interessieren uns für die Netzwerkverwaltung und wir interessieren uns wie Migros eine Site managed, wie die Prozesse sind, und wie die Daten und das Inventar gehalten wird und deshalb mal die Frage, was ist so ungefähr die Grössenordnung der Geräte, pro Site und insgesamt?

SIMON HAURI: Aktuell haben wir ungefähr 25000 Access Points und etwa 5000 Switches und 3000 Router

OLIVER FISCHER: Ich hätte gesagt bei etwa 6000 Switches und 28000 Access Points

SIMON HAURI: Und dann kommt natürlich noch das Data Center dazu und ob das Data Center auch noch dazu gehört, aber das ist ja nicht Branch, ja und Firewalls haben wir in der Branch aussen sehr wenige, etwa 20-30 in sehr grossen Produktionsstätten.

DANIEL STEUDLER: Dann nimmt uns vor allem wunder wie sieht jetzt ein Workflow aus, wenn ein Branch oder eine Site aufgeschaltet wird.

SIMON HAURI: Jetzt gibt es zwei Antworten: und zwar wir haben auf der einen Seite Migros-Net 2.0, das ist die heutige Legacy enrollment das wir machen. Auf der anderen Seite haben wir Luca da, da haben wir Migros-Net 3.0, das automatisiert wird, in diesem Zusammenhang. Wir haben, wenn wir heute einen Branch verwalten, wenn wir den LAN-Anschluss anfangen, das ist ein Bestellformular bei einem Provider. Komplett entfernt von Automatisierung. Wir haben einen Router, den wir in die Filialen tun. Den machen wir, Luca du strafst mich Lügen, per Serienbrief Konfiguration generieren und danach auf diesen darauf stellen. Ist schlicht ergreifend noch nicht anders automatisiert, weil der Druck zu klein gewesen ist.

MARTIN STYPINSKI: Aber es löst das Problem, also ist es valide.

SIMON HAURI: Ja, es ist, wie soll ich sagen, es löst es nur einmalig, nicht nachhaltig, in diesem Zusammenhang. Es ist kein Git-Repo wo man das Konfig nachher speichert, sondern man generiert es einmal und es ist „fire and forget“ und es ist im System drin.

MARTIN STYPINSKI: Ich habe mehr gemeint im Sinn von, ihr müsst nicht schnell für eine Lösung, weil es ist irgendwann aufgebaut worden und hat funktioniert und ich sag das bei vielen Projekten die etwas mehr Legacy enthalten. Man hat damals in der Zeit in der man das Entschieden hat man vielleicht kein besseres Tool gehabt. Man hat es nach bestem Wissen und Gewissen gemacht und dann ist man ein paar Jahre mit dieser Lösung gefahren und in der Zeit hat sich der Markt entwickelt und ich finde es eigentlich noch eine clevere Idee, Serienbriefe um Konfigs zu generieren, ich meine ja wieso nicht.

LUCA HEGER: Gut man muss dazu sagen, die sind ja nicht von uns verwaltet, die Router.

OLIVER FISCHER: Nicht die Subito-Router. Die werden von uns verwaltet. Die werden per Serienbrief gemacht, die Swisscom Router werden von der Swisscom gemacht, Switches kannst du besser erzählen.

LUCA HEGER: Bei Switches ist es eigentlich so, dass die On-Site Leute gewisse Switches an Lager haben, und sie haben einen Konfiggenerator, den wir ihnen zur Verfügung stellen, und sie die Variablen jeweils ausfüllen und somit zusammenstellen und draufpasten, wir geben die Recommended Software vor auf einer Website, von der sie es herunterladen können und somit ist der Switch eigentlich einsatzbereit.

SIMON HAURI: Also das ist eine eigen gestrickte Lösung, die von Luca als seine Arbeit.

LUCA HEGER: Nein, das war noch die Arbeit des Vorgängers, die ich dann weiter entwickelt habe. Das wäre jetzt dann noch Netz zwei in dem Sinn, die Webseite bei der sie die Variablen einfüllen.

SIMON HAURI: Und die Access Points werden nach [nicht verstanden 4:30] Template vom Controller heraus gepusht und sonst die DHCP Option, nein die werden pregestaged, Entschuldigung, von uns vorgängig vorher konfiguriert und über Kontroller über Templatevisierung.

OLIVER FISCHER: Wenn ich noch ganz kurz einen historischen Rahmen machen darf, ganz am Anfang ist natürlich, was braucht der Standort für eine Konnektivität. Also auf der Fläche, wie viele Geräte kommen dort hin, welche Kassenlösung gibt es, bekommt der Subito oder nicht und für den Bereich WLAN gibt es dann auch eine ganz klassische Ausmessung, dort ist einfach die Schwierigkeit, dass man das WLAN schon braucht, bevor der Laden fertig gebaut ist und die Ausmessung macht meistens erst Sinn, wenn die Glaswand und das Palette mit dem Mineralwasser dort steht, wo es dann eigentlich auch wirklich dauerhaft ist und aufgrund von so einer Ausmessung wir dann auch die Anzahl Access Points gestaged, oder auch Aufträge an den Elektriker gegeben, der diese Kabel dann auch wirklich zieht.

SIMON HAURI: Ist es spannend auch noch zu hören, was geplant ist oder nicht?

FELIX KUBLI: Durchaus, denke ich.

SIMON HAURI: Das wäre Lucas Baustelle.

LUCA HEGER: Ja also bei Migros-Net drei ist eigentlich so das Projekt, dass die Legacy-Umgebung ablösen soll und wir sind dort mit einigen Sachen am Testen, ein Beispiel ist, ein Web-Portal das auf Django aufbaut. Wo man gewisse Daten mit Python-Skripts zusammenziehen kann, gerade z.B. für die Variablen in einem Konfig. Wir haben den Ansatz, dass wir Kontroller nutzen, z.B. das DNA Center, also wir machen mal ein Initialkonfig auf das Gerät, also Entschuldigung, ich muss vielleicht etwas weiter vorne anfangen, DHCP Option 43, kommt das Gerät ans Netz, dann erkennen wir das auf dem Kontroller, respektive über den Kontroller in unserem Tool, anhand von dem wo es DHCP-mässig steht, wissen wir schon welcher Standort es ist und somit können wir eigentlich Daten zusammen ziehen, sprich IP-Adressen VLAN die es dort braucht, Hostnamen etc. Dann machen wir ein Initialkonfig über das DNA Center, das quasi einfach die IP-Addresse und den Hostnamen weitergibt, einfach so viel, dass wir das Gerät danach erreichen können. Danach

haben wir quasi das Setup gemacht mit RESTCONF und wir haben uns dazu entschieden, dass wir einfach eine strukturierte Basis für das Konfig haben. Wir haben dort ein JSON-File anstelle eines CLI, in dem wir nichts REGEXen müssen und wir gehen dann quasi schauen, wir machen ein Diff, also wir holen die Initialkonfig die drauf ist, und natürlich nicht komplett ist und vergleichen sie mit einer sogenannten Soll-Konfig die wir „on the fly“ herstellen, die immer individuell ist pro Switch. Natürlich gibt es ein Diff und dann sagen wir, nein ok das ist einfach das Soll-Konfig, dass wiederum bei uns im Git abgelegt wird, um die Versionierung sicher zu stellen. Aus unserer Sicht hat das den Vorteil, dass wir in zwei drei Wochen, oder in einem Jahr immer noch sagen können, der Switch muss genau so aussehen und wir können schauen gehen, ob er auch immer noch so aussieht. Wenn mal ein Feature verteilt wird, oder nur auf ein paar Switches, das temporär war, wird man das in einem Jahr herausfinden und wir können es begradigen.

SIMON HAURI: Und das haben wir halt per FTP gelöst, respektive in der alten Umgebung per CISCO Management Tool und per FTP ist jetzt nicht die schönste Variante, da muss man die Konfig suchen gehen. Von dem her denke ich, fahren wir dann mit Git besser.

LUCA HEGER: Genau, und dann haben wir natürlich auch die Nachvollziehbarkeit. Wir haben jetzt z.B. denn Umstand beim CLI basierten Compliance, dass wir eigentlich wissen was darauf muss und nicht darauf ist, aber wir wissen nicht immer, oder nur über Umwege, was eigentlich nicht mehr darauf sein müsste. Ein zusätzlicher DNS als Beispiel. Soll ich noch weiter ausholen?

OLIVER FISCHER: Ich denke wir müssen mal die Chance für eine zweite Frage geben, sonst kommen sie am Ende gar nicht durch.

DANIEL STEUDLER: Nein, ich möchte gerne etwas weiter bohren. Die Informationen zu wie das alles parametrisiert wird, woher kommen die? Du hast gesagt DNA Center ist ein Teil wo ihr es Konfiguriert, aber sind das deterministische Parameter?

LUCA HEGER: Also wir haben verschiedene Variablen, also wenn wir von einem Switch sprechen, z.B. einem Hostnamen oder einer IP, dann wird es über irgendein Schlüssel zusammengestellt und bei uns ist das jetzt momentan noch der Standort und unsere Idee ist dann eine Standort-DB, die wir auch zum Teil haben, das ist auch zum Teil von der Unternehmensstruktur abhängig. Wir haben quasi eine Standort-DB für alle Filialen.

SIMON HAURI: Das das ihr auf der Website abrufen könnt und wir im Backend per API Schnittstelle bereitstellen und wir die Filialdaten abrufen können.

LUCA HEGER: Und aufgrund von dem würden wir den Hostnamen zusammenstellen. Dann gibt es natürlich noch mehr Unternehmen, da sind wir jetzt Lösungen am Suchen wie wir das machen könnten. Die IP-Adresse wäre wieder über DHCP Option 43, die Idee ist, dass der nicht hier gestaged wird, sondern z.B. in der Filiale Altstetten, wenn er dort ans Netz kommt, und über die IP die er dort bekommt, können wir ja schon im IPAM herausfinden in welchem Standort der ist und so entsprechend die IP vergeben.

SIMON HAURI: Die Informationen, die wir heute vergeben sind im Filial-API als Source, alles andere definieren wir insofern selbst. Das heisst wir definieren das Template selbst, und das einzige, dass man abziehen kann ist der Standort mit den Parametern zum Standort. Die IP-Range muss man von Hand definieren, das ist ein Thema bei dem wir uns auch überlegen, wie wir das in Zukunft lösen wollen, und sonst für die Konfiguration, was für SSID darauf sind, ist von Hand definiert, aufgrund wovon der Laden wünscht. Konkret, heute mangelt es uns an einer CMDB, oder an einem Filial-Repo, oder einem Inventory, das man mit Services verknüpfen kann.

OLIVER FISCHER: Ich denke vor allem an einem Repository, das ein wenig Tagesaktuell ist. Also es gibt schon Systeme, die sich CMDB nennen, die werden aber meistens für interne Weiterverrechnung genutzt, aber nicht so, dass wir dem Vertrauen, dass das jetzt wirklich die Komponenten, sind die, per viertel nach eins jetzt aktiv in unsrem Netz sind.

SIMON HAURI: Die Diskussion, die wir für das neue Projekt noch nicht gelöst haben, weil es ginge ja darum, dass wenn du eine Bestellung pro Filiale hast, willst du ja gerne, dass in der Bestellung hinterlegt ist, ich brauche die SSID die für ein Bringer-Only-Device ist, oder für ein IoT-Netz und solche Sachen, das haben wir heute nicht, das ziehen wir heraus. Work-in-Progress.

LUCA HEGER: Bei den vielen verschiedenen Unternehmen, die wir betreuen ist es relativ komplex, wir haben eine Filial-DB, aber was ist mit der Industrie, mit den Freizeit-Zentren, die irgendwie noch zur Genossenschaft hinzugehören.

DANIEL STEUDLER: Werden die denn ganz anders behandelt, oder sind die mehr oder weniger ähnlich?

SIMON HAURI: Status heute, ist ein Fitnesszentrum das zur Genossenschaft Luzern gehört nicht gleich wie ein Fitnesszentrum, das zur Genossenschaft Zürich gehört. Du hast bei jedem Kunden seine Eigenheiten. Das wird standardisiert, aber nicht von uns, sondern vom Konzern. Aber nichtsdestotrotz, du hast die ganze Industrie und jeder darf selbst Sachen entscheiden, die uns betreffen. Kleines Beispiel, wenn du bei uns in den Laden läufst, hast du eine SSID, die heisst Migros-Wifi, das Wifi, dass wir den Kunden anbieten. Die Klubschule

hingegen, will nicht, dass die Migros-Wifi heisst, denn man ist ja offensichtlich nicht die Migros, sondern die Klubschule. Und dann gibt es Sachen, die nirgends hinterlegt sind, oder die wir nicht abrufen können.

OLIVER FISCHER: Was auch noch hineinspielt sind die Freiheiten, die sie haben. Sie können gewisse IT-Lösungen selbst beschaffen, die bei uns auf dem Netz laufen. Wenn wir jetzt auf die Freizeitzentralen schauen, hat vielleicht jeder eine andere Software um seine Members zu managen, oder auch die Fitnessgeräte die auch immer mehr Konnektivität brauchen. Je nachdem steht dann das Backend System vor Ort oder im Data-Center oder in irgend einer Cloud und das heisst für uns dann im Moment wir bauen immer wieder ein eigenes Netz, schützen das über irgend eine Firewallzone, routen das irgendwo durch. Die Hardware ist standardisiert, aber die Konfigs widerspiegeln die Unternehmens-DNA des Ladenkonstrukts.

DANIEL STEUDLER: Und das kann der Laden dann auch selbst ändern.

OLIVER FISCHER: Ich denke was die Migros gesamtheitlich hat ist das Genossenschaftsdenken, wo man ja individuell etwas will und andererseits auch mit was für Aufwänden das verbunden ist und wo auch Skaleneffekte dadurch verloren gehen. Wir versuchen natürlich immer aus dem Netz heraus auch in einer Generaldirektion oben das hereinzugeben und sagen, hey, kann jetzt nicht sein, dass die für das genau gleiche Thema die Mitbewerberlösung einsetzen und wir wieder eine eigene Konfig pflegen müssen die nur in dieser Region läuft. Sodass wenn einer eine Idee hat, die eine IT Lösung benötigt, lass uns eine Lösung suchen, die für die ganze Migros ist. Dann können wir es einmal implementieren und haben es dann auf allen 28'000 Access Points.

SIMON HAURI: Technisch gesprochen, man kann es nicht ändern, man kann es wünschen, dass wir es ändern, umsetzen, müssen wir es. Also der Selfservice Gedanken ist momentan noch nicht etabliert, der ist dann im Migros-Netz 3.0, aber auch dort wird er dann wohl so etabliert werden, dass man sagt, man hat ein Zwischen-Layer von einer lokalen IT, du gibst den Selfservice wahrscheinlich nicht dem Filialleiter heraus, sondern du schaltest auch dort eine Zwischenstufe, weil sonst muss man hinten auch noch einen Diktionär hinschreiben.

MARTIN STYPINSKI: Ich meine eben, es ist dunkle Magie.

SIMON HAURI: Genau. Wir managen das SSchwarze Loch"für viele.

MARTIN STEINSKI: Vielleicht eine etwas einfache Frage: Wie viele Files, oder wie viele Objekte müssen angefasst werden, um eine solche Konfig zu erstellen.

Ich meine, sagen wir ich konfiguriere einen Access Point, dann wird der vorkonfiguriert und dann eingesteckt, aber dann hat man gewisse Standortanforderungen, die sind sicher irgendwo abgelegt in einem File. Man hat irgendwo eine IP-Range abgelegt.

OLIVER FISCHER: Irgendwelche Firewall-Rules, oder...

MARTIN STYPINSKI: Genau

SIMON HAURI: Also was wir haben ist, wenn wir Access Points haben dann gibt es sehr viele verschiedene Varianten, zuerst machst du mal die Gebäudepläne,...

MARTIN STYPINSKI: Nein ich meine nicht den physischen Standort des Geräts, sondern eher den digitalen Standort. Ich meine, wenn ich jetzt so eine Konfig auf einen Switch aufspiele, dann hat der ja einen Namen, der setzt sich dann vielleicht zusammen aus dem Filialstandort und irgendeiner Nummer, aber die Nummer ist ja irgendwie inventarisiert. Wie viele, falls muss man dann anfassen, um so eine Switch Konfig zusammenzustellen.

LUCA HEGER: Also momentan für eine Switchkonfig muss man das IPAM anfassen, um die IP herauszuziehen, aber man hat eigentlich die Homepage die variablisiert ist, sodass man ein Unternehmen auswählen kann, und das Konfigfile wird dementsprechend angefasst. Zum Beispiel ein Standard VLAN, oder so.

MARTIN STYPINSKI: Also ihr habt beim Deployment eigentlich keinen manuellen Eingriff. Ausser das Ausführen dieser Website

LUCA HEGER: Doch momentan schon, weil man muss dann die Konfig nehmen und manuell darauf tun.

OLIVER FISCHER: Du lässt es generieren und ladest es herunter.

MARTIN STYPINSKI: Aber es ist quasi besser als Deployment. Der letzte der es aufgespielt hat es noch irgendwo auf der Festplatte.

SIMON HAURI: Ja also es ist eigentlich eine einmal Generierung. Darum war AP ein besseres Beispiel das Spiel denn dort war die Physis relevant. Beim Switch ist die Physis nicht so relevant. Beim Switch hast du eine Laufnummer und du gehst im IPAM das erfassen. Dann hast du eine Laufnummer, die du in der Regel weisst, weil es in der Regel eine 1m oder eine 2m ist. Dann siehst du auch im IPAM, wie viel, dass du in der Regel hast. Also du musst zum Beispiel den Radius eintragen um zu sagen, dass der Radius-Requests entgegennehmen darf. Du musst das im Management System nachtragen, also in einem Cisco DNA Center oder in einem Cisco Prime musst du denn auch noch eintragen und du musst im IPAM den DNS Namen nachtragen.

Das sind die Umsysteme die du in diesem Moment machst, die wir noch nicht automatisiert haben. Weil wenn du das automatisiert dann machst du automatisch auch gleich die add, remove und change. Das haben wir noch nicht gemacht und darum verschieben wir jetzt schlicht ergreifend das Projekt, weil dort der Prozess noch nicht angepasst ist.

MARTIN STYPINSKI: Was ist denn insofern der Pain-Point in dieser Arbeit? Also Ich bin ehrlich ich kriege es hin auf den PC beim deployen, zwischen builden und deployen, der biegt sich das Universum und das macht etwas anderes, beziehungsweise Ich habe danach Ärger. Also die manuellen Deployments bin ich wirklich einer der viel Flüchtigkeitsfehler macht.

LUCA HEGER: Also durch den Konfigurationsgenerator, muss ich sagen, läuft es ziemlich gut, bei der On-Site, weil das sind ja 4 Deals machen und sie müssen den Text eigentlich nur Copy-Pasten.

SIMON HAURI: Also du sagst welches US das du hast und welchen Switch, dass du hast, sag bei welchem Unternehmen du bist, und aufgrund von dem macht er die VLANs, du sagst in welcher Ortschaft das musst du logischerweise eingeben sagst Strasse Gebäude Stockwerk, du sagst, wo das Management ist, weil das konnten wir nicht automatisieren weil, es einfach zu unstrukturiert ist, und dann sagst du Konfig speichern und dann bekommst du ein ZIP-File und das nimmst du danach. Es ist eigentlich ein RAR-File, dass du danach rüber pastest. Auf das Einzige, dass du hier aufpassen musst ist, dass sich der USB-Serial-Input mit 9600 Port nicht verschluckt.

LUCA HEGER: Aber auch das was Simon vorher angesprochen hat, mit den Umsystemen, ist ein Pain-Point. Wir haben das Monitoring, DNS, CMDB und da haben wir manchmal Inkonsistenzen. Also Geräte die dort sind und dort nicht oder umgekehrt.

SIMON HAURI: Und warum haben wir es noch nicht automatisiert, in dem Zusammenhang? Wir haben eine CMDB für die Assets, das ist Helpline und das ist jetzt nicht gerade das etablierteste Produkt, um nicht zu sagen ein Nischenplayer. Und dort musst du Schnittstellen immer wieder programmieren, weil es wäre schön, wenn man die Asset Datenbank direkt könnte pflegen, dort wo sie sein sollte. Da müssen wir heute über die Umwege von Prime machen. Im Moment war das grösste Problem und der Grund, weil wir noch nicht weiter automatisiert haben. Wer pflegt den Status hier zum Update der Umsysteme. Also wenn Helpline ein Update macht, wenn du ihm IPAM ein Update machst, wenn du in Prime ein Update machst, haben wir dann einfach zu viele Probleme gehabt, damit dass die Prime Schnittstelle ändert, Prime legt es nicht gleich an das Datenformat ist anders. Das beste Beispiel

ist die Verrechnung. Man muss nach jedem Update erneut kontrollieren. Und da hatten wir schlicht und einfach zu wenig Ressourcen um weiterzugehen. Und das sind danach Themen, die wir in der Zukunft werden lösen müssen, die wir aber im Moment auch gemieden haben wie des Teufels Weihwasser. Das ist so der Treiber warum wir dort nicht weitergemacht haben, denn sinnvoll wäre es.

MARTIN STYPINSKI: Ich verstehe aber auch den Frust, weil es ist Software mässig eine sehr undankbare Arbeit, das ist eine Software, die man irgendwo in ein sehr heterogenes System reinsetzen muss, wenn man dann Major und Minor API Releases hat dann rennt man einfach immer hinterher.

SIMON HAURI: du bist zum Teil nicht mal bei API Releases, sondern beim automatisierten CSV Import und CSV Export. Ans Bluecat können wir jetzt seit eineinhalb Jahren per API sauber andocken. Aber eben gerade Prime ist am Morgen CSV Import und am Abend CSV Exports und da würde ich sagen, das ist im Moment schon nicht mehr so zeitgemäss. In der aktuellen Situation haben wir nicht eine klare Definition wo die Source ist, für die Daten die wir reintun. Wahr würde ich sagen das ist im Moment unsere Konfig, wenn wir nicht Filialen schauen. Das ist die einzige Wahrheit die Wir haben, das soll aber auch ändern.

LUCA HEGER: Was auch viel Aufwand generiert, ist zum Beispiel, die automatisierten Tasks, die wir im Moment laufen lassen können, machen wir im Moment anhand des Hostnamens, denn die sind im Moment so generiert, dass zuerst das Unternehmen kommt, dann die Stadt, dann die Strasse. Und jetzt durch das, dass das auf historischer Ebene gewachsen ist und Unternehmen hinzugekommen sind und gegangen sind, Stimmen diese Hostnamen nicht immer, in dieser Prime CMDB. Oder es fehlen Geräte und das ist dann halt eine laufende Korrektur.

OLIVER FISCHER: Darf ich auch noch kurz eine Frage stellen? Nehmen wir eigentlich die Strasse der Filial-Adresse, also dort wo der Eingang ist, oder nehmen wir den Warenzugang?

SIMON HAURI: Das erste.

OLIVER FISCHER: Weil das sind wir auch zwei Adressen, welche nicht immer gleich sind.

LUCA HEGER: Das war auch ein Thema bei meiner Arbeit. Worauf basiere ich jetzt? Ich habe darauf passiert auf was auf migros.ch steht, das Problem ist einfach manchmal steht dort bei einem Unternehmen einfach Tivoli und keine Adresse.

MARTIN STYPINSKI: Wie machen es denn andere Firmen von ihrer Grösse? Wo machen die ihre Abkürzungen und wo glaubt ihr das bei denen der Pain-Point ist?

SIMON HAURI: Also die Swisscom wäre ja im alten Konstrukt genauso unterwegs gewesen wie wir. Die haben das Produkt die Ortschaft Die Strasse und die Laufnummer darin. UPC hat es sich einfacher gemacht Komma die haben sich einfach generische SAP-Nummern genommen und platzieren sie, wo sie wollen.

MARTIN STYPINSKI: Aber die haben ja das SAP Single Source of Truth?

SIMON HAURI: Ja aber, das Problem ist, SAP steht für Service Access Point und nicht für das Produkt SAP von Walldorf, das ist der etwas fiese Teil dahinter.

OLIVER FISCHER: Also Ich habe bei den Swisscom auch noch geschaut, wie sie es machen. Also der Teil den ihr jetzt händisch mit Drag-and-Drop macht, machen sie mit irgendeiner Automatisierungssoftware, die dann auf einem laufenden PC Makro gesteuert in alle Fenster herein klickt, in der Hoffnung, dass keines davon abstürzt. Das machen die heute so, also wie früher bei einem Windows Makro Rekorder und kopieren dann die Files so hinein.

MARTIN STYPINSKI: Also der klassische Roboter.

OLIVER FISCHER: Ja genau, der Roboteransatz aber die machen das heute wirklich so. Wir haben gefragt wie macht ihr das denn so in eurer Grösse. Das ist jetzt nicht genau das wo wir hinwollen, aber man kann es so machen.

SIMON HAURI: Was man auch sagen kann, der Serienbrief Ansatz, den wir als gewählt haben, den haben nicht wir erfunden.

SIMON HAURI: Als wir Leute von unseren Partnern eingestellt haben, war das so wie die es gemacht haben, und ich behaupte, dass auch heute noch sehr viele mit diesem Ansatz arbeiten.

MARTIN STYPINSKI: Ja genau, das machen sehr viele Leute so. Das ist nicht das erste Mal, dass ich so etwas gehört habe.

SIMON HAURI: Und sonst von der Namensgebung her, sind wir in der aktuellen Situation drin und hab und haben ein neues Namenskonzept generiert und ich kenne eigentlich nur die beiden Varianten, dass du eine Unique ID nimmst, mit einer Service Access Point Nummer und du diese durch generierst, oder dass du etwas nimmst das auf dem Standort passiert. Wir haben uns so entschieden dass wir in der Zukunft nur den Standort hineinnehmen und nicht mehr die Unternehmung. Weil das Unternehmen ist der grösste Pain-Point. Weil, wie Luca vorhergesagt hat, das Fitnesscenter gehört zur Genossenschaft

Zürich und wird jetzt an die Genossenschaft Aarau verkauft. Dann muss man den Namen ändern. Jetzt bei der Swisscom, ist das Stamm VRF, das im MPLS drauf ist. Also wenn jetzt der Router von der Genossenschaft Zürich zur Genossenschaft Aarau wechselt dann müssen sie den Router ersetzen, weil sie es im System nicht abbilden können. Das ist einer der grossen Gründe warum wir kein Unternehmen mehr darin haben wollen. Dann ist man freier, weil man will ja die Geräte automatisieren und nicht das Unternehmen.

LUCA HEGER: Eine hinter Idee von dem ist, dass man, wenn man mal eine CMDB hätte, dass man dann den Standorten IDs vergeben könnte und dass der Name am Schluss gar nicht mehr so relevant wäre. Dass man eigentlich auf die ID geht und genau weiss, um was es sich handelt, ob das jetzt automatisch ist oder von Hand nachgeschaut.

MARTIN STYPINSKI: Es ist halt schon so, dass das physische Business eine ganz andere Struktur hat als der IP-Layer.

LUCA HEGER: Ja, und trotzdem müssen sie On-Site Leute wissen, wo sie hinfahren müssen.

SIMON HAURI: Was wir auch noch sagen können, Basil kennst du wahrscheinlich, mit ihm sind wir bei uns am Namenskonzept, also Beschriftungskonzept.Wo wir denken am Schluss ist die Unique ID eigentlich das einzige relevante, das zählt nicht für die Leute, die daran arbeiten. Du musst Human-Readable-Attributes haben, und die müssen genug gross sein, das ist etwas, wo die Leute die Herleitung des Namens verstehen.

MARTIN STYPINSKI: Aber das ist so, das Beste ist eigentlich ein UUID Schlüssel, aus Software Sicht und das schlimmste aus User Sicht

SIMON HAURI: Also da haben wir jetzt im grossen Rahmen Umfragen gemacht, zu diesem Thema. Wir wollten eigentlich mit dem QR-Code ins Rennen, wo du das Handy hervornehmen könntest und sagen können es das ist der AP und das ist Staging, aber da sind wir von On-Site Leuten fast geköpft worden.

MARTIN STYPINSKI: Ich dachte QR-Code sei noch eine coole Idee.

SIMON HAURI: Nein. Es kamen so Themen auf wie ich möchte einen AP stagen, ich muss in die Filiale, habe aber noch kein WLAN und unter Umständen auch kein 4G Empfang. Der Erste AP wird mühsam. Und wer darf diese Handy-App installiert haben? Wir haben dann die Security im Rücken, die dann sagt die App braucht eine 2FA. Und das wird dann mühsam. Wir haben auch getestet wie weit der Range ist wo man noch ein QR-Code scannen kann, es sind etwa 10 Meter.

DANIEL STEUDLER: Wie merkt ihr, das etwas an verschiedenen Orten falsch eingetragen ist?

SIMON HAURI: Entweder die Verrechnung oder beim Troubleshooting. Die Verrechnung ist in der Regel das Wirksamste.

OLIVER FISCHER: Die Qualität mit der Datenbank muss Wasserdicht sein. Wenn dort ein Fehler drin ist, dann lösche ich schnell ein Netz. So wie ich mir das vorstellen muss da ein Vergleich zwischen heute Morgen und heute Nachmittag, und wie hoch darf die Abweichung sein.

SIMON HAURI: Das ist das, was ich gerade gesagt habe mit der Source und GitLab. Dort hast du halt nicht den Status der Umsysteme.

MARTIN STYPINSKI: Dann kann man das quantifizieren, drei von hundert Access-Points gehen auseinander und geben Fehler.

SIMON HAURI: Bei den Access-Points ist die Fehlerrate am geringsten. Wir prestagen die Access-Points. Wir haben dort den gesamten Prozess unter unserer Hoheit. Dort wo wir nicht alles unter unserer Hoheit haben, dort ist die Fehlerrate höher. Wir wollen in Zukunft die Konfiguration überschreiben. Wenn die Wahrheit abweicht, dass wir jeden Abend sagen können, hier ist das Diff, ihr werdet nun wieder zurückgesetzt.

MARTIN STYPINSKI: Wieso gibt es überhaupt eine Diskrepanz zwischen Wahrheit und der Komponente?

OLIVER FISCHER: Da hat jemand von Hand eingegriffen.

SIMON HAURI: Du hast gesagt du bist ein Mensch und machst Fehler. Wir haben in der Distribution zwischen 50 und 100 Leuten welche Zugriffe haben im On-Site.

LUCA HEGER: Es haben sehr viele Leute Zugriff. Wir haben nicht wirklich ein Privilege-Level. Entweder du hast Zugriff und kannst alles machen oder gar nichts. Das ist zum Teil auch historisch. Auch weil nach und nach Unternehmen hinzugekommen sind. Zu deiner vorherigen Frage, was wir probieren ist ISE und Prime zu korrelieren. Im Prime muss es sein fürs Backup und im ISE muss es sein, sonst hat das Gerät keinen Netzwerk Zugriff.

SIMON HAURI: Managment-Systeme gegenüberstellen und ein Diff machen. Luca ist das am Automatisieren, ich hab das damals einmal im Jahr gemacht.

DANIEL STEUDLER: Wie hoch ist der Aufwand?

LUCA HEGER: Die Arbeit ist vor allem das Ausbügeln. Du musst jeden Fall einzeln anschauen. Du hast einen Hostnamen und weisst, das ist ein alter Hostname. Du musst herausfinden, woher er kommt, von der IP kann ich sagen, woher er kommt, dann wieso ist er dort? Ich hab einen Router gefunden welcher mit drei IPs in einem System eingetragen ist, weil er drei SPIs hat. Mit der einten IP war er da und mit der anderen dort. Die grosse Arbeit ist dem nachgehen.

SIMON HAURI: Per Definition, es ist schnell aufgesetzt, es ist einfach eine repetitive Arbeit. Du kannst nicht quantifizieren. Du kannst höchstens sagen jede Woche zwei Stunden Aufwand.

LUCA HEGER: Das zu Automatisieren ist schwierig.

DANIEL STEUDLER: Gibt es bei euch die Möglichkeit eine Übersicht über eine Site zu bekommen?

SIMON HAURI: Wir haben daran gearbeitet. Cisco hat uns mit ihren Managment-Systemen eine Möglichkeit gegeben. Die Problematik ist, wir sind von der Grösse her eine Zwischenstufe zwischen einem Provider und einem Campus Netzwerk und wir haben kein Tool ab der Stange, was uns das bereitstellen kann. Wir hatten das im Prime und das ist uns dann explodiert. Wir haben dahingegen nichts derzeit. In Zukunft wollen wir Richtung SD-Access und SDN Technologien, dort ist es dann ein Must-Have.

LUCA HEGER: Wo wir es ein wenig haben ist im Monitoring.

SIMON HAURI: Mein Wunsch wäre ein Automatic-Topology-Generator.

OLIVER FISCHER: Bei uns ist nur schon das Problem mit der Anzahl Komponenten. Cisco hat nirgends sonst 28k Access Points im Einsatz.

LUCA HEGER: Ist immer wieder ein Masse-Problem. Wir mussten es auf mehrere Monitorings aufteilen.

MARTIN STYPINSKI: Ist noch spannend das nicht andere das mit der Anzahl Geräten nicht haben.

SIMON HAURI: Wenn du Walmart oder Aldi anschaust, dann sind die viel mehr segmentiert. Jede Region hat ihre eigene IP. Da sind wir schon speziell und dort müssen wir auch schauen wie wir uns segmentieren. Wir machen es Schweizweit am grössten, aber im Ausland bist du so viel grösser, dass du segmentieren musst. Bei Cisco hast du Service-Provider und Enterprise-Produkte. Wir kaufen zwischendurch Service-Provider-Produkte aber alles was LAN oder Access ist, gibt es keine Service-Provider Produkte. Da erreicht man einfach das Skalenlimit.

MARTIN STYPINSKI: Man gibt Accesspoints heraus, aber gleichzeitig ist man kein Service Provider.

SIMON HAURI: Korrekt

FELIX KUBLI: Hat dann einfach das Programm Probleme mit so vielen Geräten?

SIMON HAURI: Beispiel Prime: Wir haben eine Oracle-Lizenz. Die ist nicht full-blown, sonst wäre die Lizenz doppelt so teuer. Dann fügst du alle Geräte hinzu und dann macht Oracle den Schirm zu. Und wenn es kein Lizenzproblem ist, dann ist es ein Ressourcenproblem. Alles, was du Verschlüsseln musst, ist die Verschlüsselung dann in Hardware.

...

[*Wir präsentieren unser Elevator Pitch*]

...

SIMON HAURI: Wenn du die Truth anschaust. Nicht die ganze Konfiguration ist die Wahrheit aus meiner Sicht. Die Wahrheit ist ein Teil der Konfiguration. Die Portkonfiguration passt sich automatisch an, je nach dem was drin steckt. Weil das ist ja nachher eine Differenz.

FELIX KUBLI: Dann bist du wieder Out-of-Date, das ist so.

SIMON HAURI: Was verstehst du unter Konflikterkennung?

DANIEL STEUDLER: Im einten Tool heisst die Ressource so, im anderen Tool ist sie noch nicht drin oder falsch drin.

SIMON HAURI: Also, wenn es nicht überall erfasst ist.

MARTIN STYPINSKI: Man hat vielleicht ein IP-Range an Ort A angepasst, aber an Ort B ist der IP-Range nicht angepasst. Wenn man die Daten zusammen zieht, sieht man, dort ist etwas nicht gut.

FELIX KUBLI: Die Konfliktlösung ist bidirektional.

SIMON HAURI: In der jetzigen Situation, wir haben den Stand der rapportiert wurden und du hast Schnittstellen. Jede Schnittstelle ist doof, da diese ändern kann. Eine API für mich ist weniger problematisch. Was siehst du als anderlei? Wenn ich mit Luca spreche, dann sagt er mir Github sei die Wahrheit, weil dort die Daten drin sind. Das ist für mich spannend, weil ich sagen kann das ist ein standardisiertes Produkt. Dort kann ich unseren Leuten auch das Management anvertrauen.

MARTIN STYPINSKI: Kraken soll bestehende Lösungen nicht ablösen, sondern ergänzen, vor allem um Inkonsistenzen aufzudecken. Für mich ist die Idee als Github als Single Source of Truth gar nicht so schlecht. Aber der Punkt ist, man ändert etwas im Github zum Beispiel ein Konfigurations-Change, aber dann ist in einem der Umprodukte das Feld nicht nachgeführt worden. Und dann kann man den Kraken laufen lassen und dann sieht man dort hat es Inkonsistenzen.

LUCA HEGER: Ist das in Python geschrieben? Also stösst Kraken einfach die einzelnen Prozesse an?

FELIX KUBLI: Das ist noch nicht so weit entwickelt worden. Du kannst es ausführen und den Konflikamanger drüber laufen lassen. Auch via UI.

MARTIN STYPINSKI: Wir haben Git und wir haben die Umsysteme, wenn wir Git isoliert als Entwicklungsworkflow anschauen. Es gibt ja auf Github die coole Ansicht mit den zwei Spalten, für das Anschauen der Diffs. Das können wir derzeit auch nur auf Git. Mit Kraken wäre dann das Ziel das zu visualisieren und Inbound und Outbound das Ganze auch anpassen. Es gibt eine Unique-ID eines Gerätes. Man hat aber auf gewissen Systemen den Namen und auf anderen Systemen den Namen auch. Und so kann Kraken die Github Ansicht machen und dann sehen welche Attribute sind bei welchem Gerät anders in welchem Umsystem.

SIMON HAURI: Schnell ein wording Thema, das, was du sagst ist für mich nicht die Quelle der Wahrheit, sondern ein Compliance-Thema. Heute gehe ich hin und mache ein Compliance-Check. Ich gehe davon aus Git ist die Wahrheit, mache einen Compliance-Check, und kann dann sagen, diese Systeme sind nicht compliant. Die Wahrheit muss ja trotzdem auf einem System sein.

LUCA HEGER: Das ist ein interessanter Punkt. Unser Ansatz ist, wir haben Prozesse einen Switch zu stagen, einen Switch anzupassen. Bildet ihr dann die Prozesse ab auf Kraken? Je nach Prozess ist die Source of Truth ja an einem anderen Ort.

MARTIN STYPINSKI: Ist derzeit händisch. Es ist noch nicht so weit entwickelt, wo man angeben kann, welcher Hierarchiepunkt ist in welchem Ort entscheidend.

LUCA HEGER: Ich spreche jetzt nur mal aus der Sicht der Konfiguration. Die Konfiguration ist schon im Git. Aber die IP wird im IPAM gehandelt, also ist IP der Chef vom IPAM. Änderst du dann im IPAM und es ändert im Git oder umgekehrt.

MARTIN STYPINSKI: Nein das ist derzeit händisch.

FELIX KUBLI: Im bestehenden Produkt, dem aus der Vorgängerarbeit, gibt es ein Merge-Template.

SIMON HAURI: Auf der einen Seite hast du das, was du gesagt hast mit dem Prozess. Auf der anderen Seite hast du im Campus verschiedene Technologien, welche je nach Hersteller auf verschiedenen Ebenen behandelt werden. Mit dem DNA Center haben wir die Hoffnung bei Cisco, dass wir ein Produkt haben welches sowohl LAN und WLAN löst. Aber der WAN Teil ist dann nochmals ein anderes Problem. Git ist von daher lukrativer Teil, da es Produkt unabhängig ist. Wo wir noch nicht sind, wie weit haben wir überhaupt die Freiheit etwas die vorhandenen Management-Systeme etwas zu definieren.

MARTIN STYPINSKI: Das wollen wir mit dieser Arbeit ergründen. Ich glaube man muss aufhören über eine globale Konfiguration sprechen. Man muss überlegen welche Attribute haben gewisse Geräte und wie kann man diese Generalisieren. Die Probleme sind ja überall gleich. Man muss sich dort über eine Normalisierung Gedanken machen, damit nicht mehrere Produkte einbezogen werden müssen.

LUCA HEGER: Wir haben Controller, VManager und DNA Center, und wollen diese mit allen Features ausreizen. Jetzt gibt es aber Global-Settings, welche du übers DNA Center machst. Jetzt hast du das aber im Git. Wenn sich jetzt aber der DNS ändert, machst du das über den Controller auf den Switch, wenn sich aber etwas wie FTP ändert dann gehst du direkt über den Switch.

SIMON HAURI: Du bist noch ein Schritt zu weit Luca. Aus meiner Sicht, wir mussten zwei Produktevaluationen machen. Wir haben Classic. Alles, was uns ein SDN lösen kann, können wir über Automatisierung lösen. Ich kann ein Switch nehmen, ich kann dort mit Python meine ganze Switch-Config wie in einem SDN machen. Ich muss ein Command schreiben, welches für Cisco und Huawei funktioniert. Wenn ich aber ein SDN nehme, dann sind wir in den Borders gefangen die Luca vorher erwähnt hat. Wir sind derzeit im zweiten Teil und es ist Work-in-Progress.

MARTIN STYPINSKI: Das Thema an sich ist schon sehr schwierig, weil jeder macht die Konfigurationsverwaltung wieder anders.

F.4. Gespräch mit Netrics

Information über die Firma

Firma:	Netrics
Grösse:	100 bis 500 Network Devices
Sektor:	Telekommunikation

Gesprächsteilnehmer

Name	Firma
Felix Kubli:	OST
Daniel Steudler:	OST
Beat Graf:	Netrics

Gespräch vom 15.10.2021. Das Gespräch wurde über Teams geführt und aufgezeichnet.

FELIX KUBLI: Ja, wir wollen dich ein paar Fragen fragen: Wie ihr bei der Netrics mit Netzwerkgeräten umgeht, ob ihr Workflows habt und wie ihr das Inventar speichert. Wir gehen dann da noch genauer ein. Zum Einstieg, in welcher Grössenordnung werden bei euch Geräte und Konfigurationen verwaltet?

BEAT GRAF: Wie viele Geräte wir haben?

FELIX KUBLI: Eine grobe Schätzung reicht.

BEAT GRAF: Das ist eine gute Frage.

DANIEL STEUDLER: In Order-of-Magnitude reicht. Sind es hundert oder tausend oder zehntausend.

BEAT GRAF: Wir haben zum Teil noch Kundengeräte, welche wir verwalten. Ich würde sagen zwischen zweihundert und dreihundert.

FELIX KUBLI: Hab ihr vorallem Kunden Netze welche ihr betreut oder sind es auch eigene Geräte, die ihr in der Firma verwendet.

BEAT GRAF: Bei den Kunden ist es so, dass wir das Aufsetzen, dann dem Kunden schicken und der Kunde verwaltet es dann. Betreuen würde es danach der Kunde selber. Wir gehen auch selten vor Ort. Wenn etwas kaputt geht, macht das auch der Kunde. Das was wir aktiv betreuen, ist unser eigenes Netz.

DANIEL STEUDLER: Wir hätten gerne noch ein wenig Insight wie ihr genau arbeitet. Was ist normarlerweise euer Auftrag.

BEAT GRAF: Wir haben ein Netzwerk, dass sich über 3 Datacenter hin streckt. Wir betreuen das und die Kunden sind dort dran angeschlossen. Wir schauen, dass der Backbone läuft, neue Kundenanschlüsse angeschlossen werden oder auch Kündigungen zurückgebaut werden. Plus wir haben viele Firewalls, die wir betreuen, auf denen wir Regeln machen. Halt einfach bei Changes wenn der Kunde etwas wünscht.

DANIEL STEUDLER: Beim ausschalten einer Firewall oder eine Netzwerkgerätes, was braucht ihr da für Informationen als Netzwerkengineer und woher kommen die bei euch? Wie sind Informationen verteilt?

BEAT GRAF: Firewall Regeln kommen per Mail vom Kunden. Dort musst du einfach wissen; Source und Ziel und Port. Kannst es aber auch genauer machen, zum Beispiel via Geo-Daten hinderlegen falls Zugriff nur aus der Schweiz kommen darf. Das kommt via Ticket rein und wir müssen es dann umsetzen.

DANIEL STEUDLER: Bezüglich Netzwerkgeräte also Router und Switches, habt ihr dort Inventarinformationen über diese?

BEAT GRAF: Ja, wir haben eine Box, welche das meiste beinhaltet. Aber das ist Work in Progress.

DANIEL STEUDLER: Wie muss ich mir einen Workflow bei euch in der Netbox vorstellen? Ist dort alles drin?

BEAT GRAF: Ich würde sagen vielleicht 80% ist in Netbox. Sehr viele Sachen, wie Crossconnects innerhalb vom Datacenter, sind nicht drin, weil sie länger existieren und diese nicht nachgepflegt worden sind. Die neuen Sachen sind drin. Geräte sind die meisten drin. Aber so Verbindungen oder Patches sind nicht drin.

DANIEL STEUDLER: Habt ihr verschiedene Subnetze, die ihr verwaltet?

BEAT GRAF: Wir haben RFC1918 Subnetze, welche wir den Kunden zuteilen. Das sind die privaten Ranges. Dort müssen wir schauen, dass sich diese nicht überlappen. Verschiedene Kunden sollten nicht das selbe Subnetz erhalten.

DANIEL STEUDLER: Und das funktioniert gut?

BEAT GRAF: Ich glaube ich bin erst einmal über eine Überlappung gestolpert, aber ich bin auch noch nicht solange hier.

DANIEL STEUDLER: Dann ist es bei euch nicht so ein Ding ein Big-Picture zu haben?

BEAT GRAF: Im Netbox sieht man das Netzwerk nicht. Im Netbox ist nur die Verwaltung der Netzwerkgeräte und die Patches. Wie das Netzwerk aufgebaut ist, sieht man in einem Monitoring oder den Netzwerkplänen.

DANIEL STEUDLER: Diese Netzwerkpläne sind aktuell? Werden die automatisch generiert?

BEAT GRAF: Das ist alles manuell. Das zeichnet jemand im Visio.

DANIEL STEUDLER: Und das Funktioniert gut?

BEAT GRAF: Dokumentation ist immer das letzte was gemacht wird und schlussendlich auch das, was darunter leidet. Aber ich würde sagen wir haben schon von fast überall Netzwerkpläne, aber teileweise nicht so aktuell.

DANIEL STEUDLER: Wir interessieren uns vor allem für die Painpoints, welche derzeit existieren, im Zusammenhang mit Automation.

BEAT GRAF: Derzeit haben wir fast keine Automation, also im Netzwerkbereich nicht. Wir haben teilweise versucht etwas mit Ansible zu machen, aber nicht so, dass die Informationen aus Netbox kommen, sondern wir haben zum Beispiel eine Änderung die auf 100 Geräten ausgerollt werden muss. Ausrollen machen wir dort mit Ansible, aber die Konfiguration ist Manuell geschrieben. Je nach Hersteller geht das besser.

DANIEL STEUDLER: Versionierung ist hier bei euch ein Thema?

BEAT GRAF: Nein, nicht so dass wir das im Git haben. Bei uns ist RANCID[1], das macht das Backup der Geräte, welches drei mal im Tag ein Backup der Geräte erstellt.

DANIEL STEUDLER: Wo spürt ihr den Automatisierungsdruck?

BEAT GRAF: Der Druck ist da. Es soll immer weniger Zeit aufgewendet werden, um Wünsche von Kunden zu deployen. Auf der anderen Seite hat man zu wenig Zeit um hier sich ernsthaft Gedanken zu machen und etwas umsetzen. Ich kann schon ein Ansible Playbook schreiben, aber das hat dann drei Zeilen.

FELIX KUBLI: Die Konfiguration eines neun Gerätes macht ihr von Hand oder mit einem Template?

[1]RANCID: Really Awesome New Cisco confIg Differ, ein Tool

BEAT GRAF: Wir nehmen entweder ein Template oder ein Refernzgerät und passen die Konfiguration an.

DANIEL STEUDLER: Was muss ich mir unter dem Wort Template bei euch vorstellen?

BEAT GRAF: Template ist ein paar Zeilen Code, welche Authentisierung konfiguriert. Man muss dann noch Dinge wie IPs anpassen. Templates sind Textstücke, welche man anpassen muss. Alles Handgestrickt.

FELIX KUBLI: Du hast gesagt, Ihr habt keine Versionierung? Sind euere Geräte mehr oder weniger statisch?

BEAT GRAF: Kundengeräte sind mehr oder weniger statisch. Wenn man einen Kunden aufschalten muss, dann muss man schon ein zwei Geräte verändern.

FELIX KUBLI: Aber bei euch ist das dann relativ statisch?

BEAT GRAF: Nein dort passiert schon relativ viel, wenn du einen Kunden aufschaltest musst du schauen wo der Weg eines VLANs durchgeht. Unter Umständen musst du das bei zwei, drei Switches durchschlaufen. Du musst eigentlich alle Geräte ändern, bis du beim Router bist.

DANIEL STEUDLER: Wie sieht es mit der Informationsgewinnung aus in so einem Fall? Wie weisst du, was du anfassen musst?

BEAT GRAF: Ich nehme die Netzwerkzeichnung hervor und schaue wo es durchgeht, oder ich gehe von Gerät zu Gerät.

DANIEL STEUDLER: Wo ist dein Painpoint beim jetztigen Workflow?

BEAT GRAF: Es ist viel zu statisch. Ich muss sehr viel Zeit aufwenden um herauszufinden, was ich wo konfigurieren muss. Einen neuen Kunden zum Beispiel oder eine neue Firewall-Regel. Dort musst du schauen wo das reinkommt beim Kunden. Es ist mühsam, bis man die Informationen zusammen hat.

DANIEL STEUDLER: Müsst ihr Dinge auch Debuggen? Wie müsst ihr den Kunden unterstützen?

BEAT GRAF: Ja, das muss man schon regelmässig machen. Der Kunde meldet meist, dass etwas nicht vollständig fuktioniert. Dann müssen wir schauen ob das an uns liegt.

FELIX KUBLI: Habt Ihr beim Aufschalten von Kunden ein Namingschema?

BEAT GRAF: Ist nichts niedergeschrieben, aber 80% haben wir definiert und versuchen das weiter so zu machen. Aber man stolpert immer mal wieder über Geräte die dem nicht entsprechen.

DANIEL STEUDLER: Habt ihr versteckte Informationen? Zum Beispiel IP Adressen oder Hostnamen.

BEAT GRAF: Man kann vom Hostnamen ablesen, wo das Gerät sich befindet. „GLB“ ist zum Beispiel Glattbrugg. Kundengeräte beginnen mit dem Kundennamen oder einer Abkürzung davon.

DANIEL STEUDLER: Darf der Kunde auch Änderungen an der Konfiguration vornehmen, oder ist alles von euch gemanaged?

BEAT GRAF: Es gibt beides. Entweder sind wir verantwortlich oder der Kunde, aber nicht beides. Es gibt aber auch wenige Ausnahmen, wo beide Zugriff haben.

FELIX KUBLI: Verifizieren von Netzwerken, dass es genau so ist wie ihr es erwartet, macht ihr beim Aufsetzen oder auch noch danach?

BEAT GRAF: Beim Aufsetzen wird überprüft, dass es läuft. Wenn du einen Fehler suchen musst, dann findest du heraus, dass etwas nicht richtig ist. Danach höchstens noch das Monitoring. Aber es ist keine regelmässige Überprüfung.

DANIEL STEUDLER: Wie kommen die Daten ins Monitoring?

BEAT GRAF: Wir haben ein selbstgeschriebenes Inventartool. Dort muss man es neben dem Netbox auch noch eintragen. Sobald es dort ist, kommt es automatisch ins Monitoring, aber auch ins Rancid für das Backup der Konfiguration.

DANIEL STEUDLER: Und ihr überprüft, ob das im Inventartool auch auf dem Netzwerk oder im Netbox ist?

BEAT GRAF: Ist alles händisch. Es können Fehler passieren, beziehungsweise es passieren Fehler.

DANIEL STEUDLER: Wie gehst du vor, in einem Fehlerfall?

BEAT GRAF: Dann muss ich das einfach ändern.

DANIEL STEUDLER: Und das kommt oft vor?

BEAT GRAF: Inventarmässig bin ich bisher noch nicht so viel über Fehler gestolpert. Es ist dort der Gerätenamen, eventuell noch die IP-Adresse und eine Seriennummer. Ob die Seriennummer stimmt, merkt man wohl erst wenn du einen Austausch machen musst, oder dem Hersteller eine Seriennummer angeben musst.

FELIX KUBLI: Ist der Umfang der Geräte am wachsen?

BEAT GRAF: Ja genau, mit mehr Kunden kommen mehr Geräte. Das Wachstum ist linear.

[*Wir präsentieren unser Elevator Pitch*]

...

BEAT GRAF: Und das heisst, dass Kraken alle Daten selbst holt. Stellt es die Daten dann auch dar?

FELIX KUBLI: Ja es stellt derzeit vorallem die Konflikte dar. Das heisst es gibt ein Interface welches die Konflikte anzeigt.

BEAT GRAF: Die gelösten Konflikte werden dann auch wieder zurückgeschrieben?

FELIX KUBLI: Ja genau, das ist die Idee. Derzeit schreibt es in ein eigenes Inventar. Du hast noch etwas gesagt zu Darstellung von Daten.

BEAT GRAF: Abstrakt wäre ein Netzwerkplan, aus meiner sicht wäre interessant logisch und physisch zu unterscheiden. Die Geräte sind logisch so verbunden aber hinten rum sind noch folgende Patchungen und Crossconnects involviert. Um herauszufinden wo man sonst noch Fehler finden könnte. Das macht bei uns derzeit der Netzwerkplan und das Netbox.

FELIX KUBLI: Den Netzwerkplan müsst ihr dann manuell anpassen?

BEAT GRAF: Ja genau. Es gäbe glaub ein Plugin für Netbox, dass das machen würde. Ihr müsst in dieser Arbeit Plugins für alle Quellen schreiben.

FELIX KUBLI: Im Endeffekt schon. Wir müssen eine Art finden, wie wir die Geräte generalisieren können. Wie wir die Infos abspeichern können. Es gibt ja noch OpenConfig. Wir müssen schauen wie wir das übertragen können.

...

DANIEL STEUDLER: Ich kann kurz erzählen woher die Arbeit kommt. Die Single Source of Truth im Netzwerk gibt es nicht. Wir versuchen die verschiedenen Quellen zusammen zu ziehen und dadurch eine Source-of-Truth bilden, dann daraus zu automatisieren.

BEAT GRAF: Wir versuchen die Single Source of Truth im Netbox zu haben. Ich weiss nicht wenn wir noch ein weiteres Tool haben zu den zwei die wir schon haben, ist das nicht parktikabel. Was ich sehe ist, Informationen zu holen, Aufbereiten und zurück ins Netbox spielen.

DANIEL STEUDLER: Mein Concern ist auch wie integriert sich ein solches Tool in ein bestehenden Prozess. Ein Tool soll ja da sein sich in bestehende Prozesse zu integrieren. Vorallem in Netzwerkbereich gibt es kein One-Size-Fits-It-All. Informationen sind Kontext abhängig. Netzwerke sehen anders aus. Es sind Layers involviert. Vielleicht noch ein Overlay Netzwerk obendrauf.

DANIEL STEUDLER: Ihr habt nur WAN Geräte?

BEAT GRAF: Uns interessiert uns normalerweise nur bis zu Ihrem Router. Wir sehen nicht in sein Netzwerk hinein. Es gibt ausnahmen wie zum Beispiel VLAN deployen, wo wir auf seine Geräte zugreiffen müssen.

BEAT GRAF: Die Konflikterkennung ist bei uns ein blinder Punkt. Ich denke man braucht es schon, aber es besteht hier noch keine Awareness. Manuell wird es aus Zeitgründen nicht gemacht.

F.5. Gespräch mit Datapark

Information über die Firma

Firma:	Datapark
Grösse:	10-20 Network Devices
Sektor:	Telekommunikation

Gesprächsteilnehmer

Name	Firma
Felix Kubli:	OST
Daniel Steudler:	OST
Sandro Bolliger:	Datapark

Gespräch vom 21.10.2021. Das Gespräch wurde über Teams geführt und aufgezeichnet.

FELIX KUBLI: Wie viele Geräte werden bei euch verwaltet?

SANDRO BOLLIGER: Wir haben sechs Backbone-Geräte, alles Juniper, fünf Edge-Router, auch Juniper, zwei Router und zwei Kunden-Switches, auch Juniper. Es hat dann da noch ein paar Um-Router, etwa sieben Geräte. Wir haben noch zwei CMTS[2], Casa[3], und noch etwa 25 GPON OLTs, das ist dann der Access für ins Fiber.

DANIEL STEUDLER: Was macht Dataparkt genau?

SANDRO BOLLIGER: Internet-Service für Kabelnetzbetreiber anbieten. Wir haben bei unserem grössten Kunden auch IPTV und ein Kunde betreibt ein Softvoice-Switch. Wir betreiben dort die Integration und den Transport der Zuleitungen aus Zürich. Wir sind vor allem in der Region Wil vertreten. Wir sind aber auch noch in zwei Rechenzentren in Zürich vertreten, da wir dort Transit- und Internet-Exchange-Peerings beziehen.

[2]Cable Modem Termination System
[3]`https://www.casa-systems.com`

DANIEL STEUDLER: Ihr macht Business-to-Business.

SANDRO BOLLIGER: Genau mehrheitlich B2B. Für Privatkunden haben wir gar nichts, aber unser Kunde, der Fiber oder Kabelfernsehnetz baut, der hat dann den Endkunden. Dieser macht dann aber auch den Support für die. Wir haben auch einen grösseren Kunden, welcher in der Region ein Fiber und Kabelnetz baut. Wir haben auch kleinere Firmen, welche ein Firmenanschluss haben, welche dann auf eine Darkfiber, welche wir dann beleuchten. Die Darkfiber ist dann aber auch immer eingekauft.

DANIEL STEUDLER: Uns nehmen ein wenig die Workflows wunder.

SANDRO BOLLIGER: Man muss sagen was an Datapark speziell ist, dass wir vor zwölf Jahren eine Software fürs Management von DSLAMs und Endkundengeräte geschrieben haben und das nennen wir SAM, Service Access Manager. Das Management der DLSAMs macht alles SAM im Moment, das ist aber alles SNMP und von dem müssen wir weg. Die Plattform im Access kann aber derzeit nur per SNMP verwaltet werden. Also wir sind derzeit am Schauen was wir als neues Produkt verwenden können, damit wir von SNMP weg kommen. Wenn wir jetzt einen neuen Kunden aufschalten, geht SAM hin und konfiguriert das DSLAM mit SNMP. Unsere Daily Tasks sind gar nicht so massiv. Wir schalten vielleicht mal ein Peering auf oder ein neues Peering, oder bereiten die Aufschaltung eines neuen DSLAMs. Ports hinzufügen etc. Wir bieten unserem Endkunden den Service bis zum Endgerät alles zu konfigurieren, müsste das alles ansprechbar sein. Entweder aus einem Tool, welches wir schon haben oder ein GUI für unseren Endkunden zur Verfügung stellen.

DANIEL STEUDLER: Die Informationen wie ihr Dinge verwaltet ist vor allem im SAM?

SANDRO BOLLIGER: Man muss das vielleicht mal aufzeichnen. Wir halten uns stark an die Struktur von Backbone, Distribution, und Access. So das klassische Cisco-Design haben wir eigentlich. Im Backbone ist die Basiskonfiguration mit einem statischen Ansible-Inventar gebaut worden letztes Jahr. Wenn ich jetzt hingehen würde, würde ich auch ins Ansible packen und das dann die Konfiguration generieren lassen. Alles was im Access ist. Für unsere Kunden ist alles im SAM.

DANIEL STEUDLER: Was muss ich mir unter statischem Inventar vorstellen?

SANDRO BOLLIGER: Es wurde einmal von Hand definiert und ich schreibe von Hand ins Inventar, wenn sich was ändert. Es ist ein YAML File pro Device.

DANIEL STEUDLER: Das ist dann euer Inventar?

SANDRO BOLLIGER: Also einfach ein Ansible-Inventory. Grundsätzlich was wir noch haben, wo welche IP vergeben ist, das ist im Netbox drin. Irgendwann wäre es noch cool, wenn man etwas ins Netbox hinzufügt, dass man den „Apply“ Button drückt, man ein Diff von einer Konfiguration bekommt und man diese bestätigen kann und diese ausgerollt wird. Das wäre dann aber Backbone und Distribution. Für unsere Kunden bräuchten wir dann ein Tool, dass wir unserem Kunden geben können. Wir gehen keine Abos konfigurieren, das macht er.

DANIEL STEUDLER: Da ist SAM derzeit da?

SANDRO BOLLIGER: Genau, aber es ist halt SNMP und durch das sehr Träge. Wenn das einen DSLAM mit zwei- dreitausend Kunden abfragen geht, geht das halt einen kleinen Moment.

DANIEL STEUDLER: Changes habt ihr in dem Fall nicht so viele?

SANDRO BOLLIGER: Im Backbone ist es eher selten. Im Access mehrere Changes täglich.

DANIEL STEUDLER: Ist bei euch DNS involviert?

SANDRO BOLLIGER: Reverse ist sehr stark. Vorwärts haben wir noch das Problem, jedes Gerät hat eine Management-IP und derzeit suchen die Leute noch diese raus. Ich hätte gerne A-Records. Kunden welche eine fixe IP haben, können ein Mail schreiben und wir tragen dann einen Reverse-Eintrag bei uns im DNS ein. Was die Endkunden haben ist ein Portal, wo sie Einstellungen zur Telefonie ändern können. Jeder Endkunde bekommt eine eigene Mailadresse. Es gibt also auch eine Konfiguration des Mailservers durchs SAM. Privatkundenanschlüsse sehen alle gleich aus, was Hostnamen angeht.

DANIEL STEUDLER: Hat aber keinen Einfluss aufs Managment?

SANDRO BOLLIGER: Wir bräuchten halt A-Records für ein einfacheres Managment.

DANIEL STEUDLER: Wie sieht die Businessseite vom Inventar aus?

SANDRO BOLLIGER: Abschreibungen werden im Wiki und in Excelsheets erfasst. Wir müssen ein neues Gerät kaufen, wenn das alte Gerät abgeschrieben ist. Das grösste Inventar ist das Netbox.

DANIEL STEUDLER: Eine Verbindung zwischen Netbox und Business gibt es nicht?

SANDRO BOLLIGER: Nein, es ist nicht verbunden. Meistens will man als Netzwerkengineer etwas verbessern und dann beginnen die Abklärungen, ob die Geräte schon abgeschrieben sind. Was speziell ist, die DSLAMs sind von uns verwaltet, aber gehören dem Kunden. Er kommt nicht via SSH darauf. Wir können ihn dann nur darauf hinweisen, er soll es ersetzen, aber wir können nicht wissen, ob er es schon abgeschrieben hat.

DANIEL STEUDLER: Du hast Verbesserungen angesprochen. Was verstehst du darunter?

SANDRO BOLLIGER: Wir hatten bis vor kurzem nur zwei Router im Core. Dort war alles dran. Sachen sind von Zürich angekommen, sind von Zürich auf Wil transportiert worden. In zwei Standorten in Wil sind die Router gestanden. Falls eine Strecke von Zürich verloren haben, dann haben wir die gesamte Kapazität verloren. Der Transport war auf Layer 1 und konnte dort nicht weg. Wir haben jetzt den neuen Backbone aus sechs Routern. Die bilden ein Viereck, vier sind in Wil, da wir mehr Ports benötigt haben, zwei sind in Zürich. Von den Aussenanbindungen terminiert eine in Zürich und eine in Wil. Fällt jetzt die Verbindung Zürich Wil aus, geschieht einfach ein Umrouting. Wir sind weggekommen von grossen Chassis-Router, wir hätten acht Linecards verbauen können. Der grösste von denen hatte vier Linecards verbaut, da wir noch auf 10Gbit waren. Im neuen Backbone können wir 100Gbit Strecken bauen. Wir haben nur noch 1HE Router. Sie sind schneller, verbrauchen weniger Platz und weniger Strom. Und wir haben den Effekt, dass falls wir eine Strecke verlieren, dass es einfach umrouted. Diese Verbesserungen werden wir aber nicht ewig machen können. Derzeit haben wir noch vieles auf Layer 2 switched und wir haben das jetzt auf MPLS gewechselt, also wir verschieben gewisse Sachen auf Layer 3. Das nächste ist das Design unseres Kunden, also der Distribution Layer. Der wird umgebaut, vereinfacht, es wird kompakter mit weniger Geräten aber mehr Bandbreite.

DANIEL STEUDLER: Ihr habt das, was ihr automatisieren könnt im SAM oder im Ansible.

SANDRO BOLLIGER: Genau. Ansible ist noch nicht ganz fertig. Derzeit spuckt es eine Konfiguration aus. Das Angenehme an Juniper ist, dass das Gerät selber diffen kann. Ich copy-paste die Konfiguration und werfe sie auf den Juniper. Thats it. Was komplett fehlt, ist ein GUI. Mein Gedanken wäre das Sales, wenn sie ein Abo erfasst haben, dann soll alles dahinter losgehen, also im Bezug auf unsere Kunden, weil das ist das, was im Access mit dem SAM schon geht. Die neue Plattform im Access wird voraussichtlich alles NETCONF/YANG, so viel haben wir schon gesehen. Dazu haben wir aber noch keine Schnittstelle. Wir haben noch nichts das mit den Nokia-Geräten, die

wir derzeit anschauen, sprechen kann. Das haben wir noch gar nicht. Meine Sicht ist, wenn wir etwas haben könnten, für alle Plattformen, wäre das optimal. Es gibt zwar nicht so viele Changes im Backbone, aber es gibt Changes. Es gibt Peerings die hinzukommen, es gibt neue MPLS Strecken.

FELIX KUBLI: Habt ihr dann schon Anstrengungen gemacht zur Umsetzung mit NETCONF/YANG? Oder ist das schon zu weit in der Zukunft?

SANDRO BOLLIGER: Absolut nicht. Der Access Layer ist [Dozen Zone (nicht verständlich 25:32)] das sind zehn bis zwölf Jahre alte Geräte. Die funktionieren seit anno dazumal mit SNMP. Wir schauen dort jetzt für den Access Layer und sind Juniper und Nokia am Anschauen. Unser Kunde baut immer PON-Netze, wir können also GPON oder XGSPON verwenden. Beide Hersteller kommen nur mit NETCONF/YANG. SNMP fällt hier schon mal raus. Wir haben jetzt ein Nokia-Chassis in unserem Lab. Wir haben noch nichts damit gemacht, aber wir haben es. Und das Gerät kann nur mit NETCONF/YANG umgehen. Entweder über die Managementsoftware von Nokia, oder über NETCONF/YANG. Es hat keine CLI. Ich kann dort nur die Managment IP setzen. Ihre Managementsoftware kann das Gerät komplett konfigurieren. Die Software kann dann aber nicht mit unseren Junipers umgehen. Aber das ist grundsätzlich überall so in der Industrie. Jeder möchte dir dein Controller für dein Gerät verkaufen. Bei 24 Cisco Geräten gibt es 24 verschiedene Managemten Softwares. Die Kosten davon sind jenseits von Gut und Böse.

DANIEL STEUDLER: Habt ihr ein Schema des Netzwerkes? Wenn ja, ist es aktuell?

SANDRO BOLLIGER: Ja haben wir und wir pflegen es. Es ist aber auch relativ übersichtlich. Wir haben ja nicht sehr viel Geräte. Die Schwierigkeit, die wir sehen, dass es unübersichtlich wird. Es ist jetzt schon ein A0 damit alles darauf Platz hat. Kundenverbindungen sind nur angeschrieben, aber nicht eingezeichnet.

DANIEL STEUDLER: Wie einfach oder komplex ist es mit den involvierten Layers?

SANDRO BOLLIGER: Du meinst Layer 1 bis 3? Layer 1 haben wir unsere ein zwei Leitungen, das ist nicht wahnsinnig komplex. Das sind fünf verschiedene Geräte an fünf verschiedenen Standorten. Layer 2 versuche ich auszulöschen. Es soll also nur noch Punkt-zu-Punkt Verbindungen geben und dann eine IP obendrauf. Das macht es wieder komplizierte mit den MPLS Services, aber das sind nicht so viele Services. Das sind derzeit wenige, sollen aber mehr werden. Layer 3 ist halt einfach Layer 3. Irgendwo ist eine IP konfiguriert, dann ist sie konfiguriert. Basis Sachen wie OSPF, normalerweise hat bei uns ein Gerät zwei oder drei Neighbours und obendrauf gibt es dann noch ein iBGP, eBGP. Was fehl ist, wir sehen nicht, wo die MPLS-Strecken hingehen.

Das sieht man in der Konfiguration aber nicht im Schema. Es gibt keine separate Doku dazu. Es gibt etwas das sagt: Der Kunde X hat eine Verbindung von A nach B, die kauft er bei uns als Transportservice.

DANIEL STEUDLER: Und dann gehst du davon aus es MPLS ist?

SANDRO BOLLIGER: Wir können den Kunden entweder auf Layer 1 oder in den MPLS stecken. Anhand von den Informationen über Locations A und B, dann ist ziemlich klar auf welchem Gerät er sein könnte. Die Chance ist 50:50.

DANIEL STEUDLER: Gibt es Pain-Points welche noch nicht erwähnt wurden?

SANDRO BOLLIGER: Ich glaube, dass Dinge mit NETCONF/YANG konfiguriert werden müssen, ist sehr wichtig. Der grösste Punkt ist, dass Konfigurationen nicht konsistent aussehen, seit wir das in Ansible drin haben, aber je länger das es geht, bis es eine Lösung gibt, um eine Lösung zusammen zu klicken, läuft das ganze auseinander, weil nicht jeder Ansible so nutzt.

DANIEL STEUDLER: Die Konfiguration im Ansible entspricht nicht dem, was auf dem Backbone ist?

SANDRO BOLLIGER: Ich bin derzeit Dinge am Nachtragen. Es ist halt mühsam. Du musst dich auf dem Gerät anmelden, die Konfig, welche du von Hand generiert hast, kontrollieren und die Konfiguration ausrollen. Und in dieser Zeit setzt du auch die Commands ab. Der Zwischenschritt, welcher das Automatisiert, fehlt. Ich renne dem Ganzen nach und inventarisiere wie es derzeit aussieht. Was man auch anschauen müsste, ist das Staging der Geräte. Ein Zero-Touch-Provisioning für unsere Industriekunden wäre toll. Was mir noch in den Sinn gekommen ist, ist Testing. Wir haben ein Hardware-Lab, aber das stellt immer nur ein Zustand dar. Derzeit versucht es die reale Welt abzubilden. Aber, wenn wir etwas komplett Neues testen wollen dann fehlt und dort ein wenig die Versionierung.

FELIX KUBLI: Wie oft musst du die Konfiguration auf den Geräten nachtragen?

SANDRO BOLLIGER: Immer, wenn ich es merke. Ich bin derzeit der Einzige der etwas mit Ansible macht, weil ich der Einzige bin, der sich fit genug fühlt. Das ist ein Schulungsthema, und haben wir definiert werden wir anschauen. Ich mache dann meistens ein Config, will sie auf das Gerät aufspielen und das Diff sagt mir dann es würde sieben Sachen löschen. Normalerweise überschreibe ich hart immer alle Konfigurationen und dann fällt es halt auf. Es ist nun oft genug aufgefallen, dass ich sagen musste, ich muss nochmals alles kontrollieren. Aber es sind ja nur elf Geräte in Ansible, das ist überschaubar.

DANIEL STEUDLER: Habt ihr eine Konvention beim Naming?

SANDRO BOLLIGER: Ja. VLANs, MPLS Strecken, Router. Nur schon die Router waren ein Thema. Wir haben immer <Location>, bei uns IATA-Code, Bindestrich, Rack, Bindestrich, Funktion und Nummer nach der Funktion. Funktion kann Access-Switch, Core oder Edge. Das wäre das Naming der Devices. Im VLAN steht INET falls es ein Internet Kunde ist und der Firmenname, und vom Firmennamen will ich weg, weil die ändern sich. Das Problem mit den Kundennamen wäre gelöst, wenn einfach die Kundennummer hinterlegt wäre oder aus meiner Sicht die Service-ID. Das fehlt bei uns auch noch ein wenig, ein Service hat keine Bezeichnung im klassischen Sinn. Das liegt daran, dass der Business-Teil, Vertrag zu Service, der existiert nicht. Es kommt jemand, dieser Kunde hat diesen Service und dann wird dieser Service gebaut. Der Kunde telefoniert mit Sales und dann gibt es eine Anpassung. Wir sind derzeit auch noch zu klein, als dass es ein Ticket für das gibt.

FELIX KUBLI: Also das Finden eines Gerätes ist bei euch weniger das Problem.

SANDRO BOLLIGER: Genau. Ich zeig euch mal kurz den Plan.

...

[*Wir präsentieren unser Elevator Pitch*]

...

SANDRO BOLLIGER: Single Source of Truth ist das angesagte Thema derzeit im Netzwerk und wenn das funktioniert und wenn es auch noch Modular ist, dann ist das super. Wichtig ist halt das es auch mehrere Hersteller unterstützt. Ich war bei Init7 und dort haben wir genau das versucht aus Netbox. Also ein Tool, welches ein Gesamtbild aus dem Netzwerk hat und kann dann modellieren, wenn ich dem Tool sage, ich brauche eine MPLS Strecke von A nach B. Das Tool schaut sich an was die Änderungen am Netzwerk sein müssen und anhand von dem kann es dann die Konfiguration generieren. Und dort waren wir relativ weit. Es ist halt so, dass alle derzeit Plugins für Netbox bauen. Netbox ist cool, weil es sehr simpel. Es wird aber schnell extrem komplex, wenn du Services abbilden möchtest, wenn nicht nur physikalische Sachen involviert sind. Dort sind wir angestanden. Und was dort auch noch war, die Datenbank ist separat und jedes Gerät ist ein Modul. Also ob das Gerät Cisco oder Juniper ist, ist dem Tool egal. Das Tool ist der Heilige-Gral, den du haben kannst. Nokia verkauft einfach für ihre Geräte eine Lösung, aber sobald du mehrere Hersteller im Netzwerk hast, bringt dir ein Tool, das nur mit einem Hersteller umgehen kann nichts.

FELIX KUBLI: Also alles, was vendorunabhängig agieren kann, ist also erwünscht.

SANDRO BOLLIGER: Genau. Toll wäre natürlich ein Open-Source bei dem man aber Support kaufen kann. Aber wo man Nokia ein Schleifchen umbinden kann, es ist einfach ein Kubernetes mit vielen Open-Source Lösungen obendrauf. Es sind Locker 24 verschiedene Services die da drauf leben. Zum Beispiel Prometheus, Grafana. Postgres wird als Datenbank verwendet. Dass sie nicht für alles etwas Eigenes bauen ist schon mal gut. Wenn sie Sachen ins Prometheus exportieren, dann brauche ich nicht unbedingt ihren Cluster, sondern kann meinen eigenen Prometheus verwenden.

DANIEL STEUDLER: Du sagst ein Bedarf für eine Single Source of Truth ist vorhanden.

SANDRO BOLLIGER: Definitiv. Wir versuchen das ja auch zu mappen. Wenn eine IP nicht im Netbox steht, darf sie nicht vergeben sein. Andere Firmen sagen, wenn die IP nicht im Forward oder im Reverse ist, dann ist sie noch frei. Es traut halt kein Netzwerkengineer seiner Single Source of Truth, dass man nicht kurz einen Ping macht oder kurz auf dem Router schaut, ob die Route existiert. Ich hab schon Leute gesehen die sagen, unser Netzwerk ist zu 100% automatisiert. Im Pikettfall geschieht eine Änderung, „on the fly“. Am nächsten Tag fällt es dann der Person auf, welche eine Änderung machen will. Dann gibt es ein Gespräch. Falls es kein Gespräch gibt, dann wird die Konfig wieder überschrieben, die Single Source of Truth hat immer recht und die Kunden welche ein Problem hatten während dem Pikettfall, haben das Problem wieder und es muss gelöst werden.

F.6. Gespräch mit init7

Information über die Firma

Firma:	Init7
Grösse:	500-1000 Network Devices
Sektor:	Telekommunikation

Gesprächsteilnehmer

Name	Organisation
Felix Kubli:	OST
Daniel Steudler:	OST
Thomas Fritz:	init7

Gespräch vom 21.10.2021. Das Gespräch wurde über Teams geführt und aufgezeichnet.

FELIX KUBLI: Wie viele Geräte werden bei euch verwaltet?

THOMAS FRITZ: Spezifizier die Geräte?

FELIX KUBLI: Switches, Router etc. Wir haben mit mehreren Firmen gesprochen. Einfach so das wir euch einordnen können.

THOMAS FRITZ: 90 Core-Router, wir reden hier etwa von 700 Devices. Die Kunden sind hier nicht dabei.

DANIEL STEUDLER: Uns interessiert es, was es beim Aufschalten von einem Gerät, was da für Quellen und Informationen involviert sind.

THOMAS FRITZ: Wir sind hier in einer Übergangsphase. Momentan basiert hier noch relativ viel auf Textfiles, Excelfiles, Informationen in Konfigurations-Dateien in einem Wiki drin. Neu soll das Ganze im Netbox gemacht werden. Im Netbox wird die Information ganz umfangreich abgebildet und dort soll auch alles rauskommen. Es soll via Templates geschehen, sodass das via Copy-und-Paste auf das Gerät geschoben werden kann. Später dann auch via TFPT

oder DHCP-Server geschoben werden können, sodass man damit auch Zero-Touch-Provisioning machen kann. Ist also noch recht traditionell bei uns. Es geht dann so weit, dass wir über Ansible die ganzen Devices auslesen aus Netbox und dann via Ansible weiter konfigurieren. Es ist da wirklich derzeit im Umbau bei uns.

DANIEL STEUDLER: Du hast Excel erwähnt?

THOMAS FRITZ: Ja Excel habe ich eigentlich nicht gemeint. Wiki habe ich gemeint. Sind halt ganz traditionell Textfiles. Es gibt relativ viele Quellen, an denen wir schauen müssen, um zum Beispiel eine IP als frei zu betrachten.

DANIEL STEUDLER: Wo schaust du dann nach, ob die IP frei ist?

THOMAS FRITZ: Im Zweifel in der Routingtabelle, auf unseren Core Routern. Schneller gehts via DNS. Aber dann schaut man doch nochmals nach auf dem Core Router. Zum Teil sind die IP's dann auch schon im Netbox drin.

FELIX KUBLI: Welche Quellen habt ihr? Du hast mehrere angesprochen. Wiki, Router...

THOMAS FRITZ: Gut, im Router ist keine Quelle, das ist das lebende Produkt. Dokumentiert ist es primär im Wiki. Es gibt noch ein-Customer Relationship-Management-System. Dort sind Kundendaten hinterlegt und mit anderen Informationsquelle vernetzt. Wir holen Informationen aus dem CRM, welches selbstgestrickt ist, raus. Wir haben Überwachungstools, welche wir als Informationsquelle heranziehen können, weil diese automatisiert neue Geräte abfragen können und dann so gewisse Details eines Gerätes zur Verfügung stellen. Aber bisher als Dokumentationsquelle dient das Wiki, aber das wird nun durch das Netbox abgelöst.

DANIEL STEUDLER: Werden diese Daten dann auch überprüft, dass diese richtig sind? Klassischerweise ein Pikettmitarbeiter macht eine Änderung und dann stimmt es nicht mehr in der Dokumentation.

THOMAS FRITZ: Nein leider nicht, der jeweilige Engineer muss dann seine drei, vier Quellen abklappern. Ist leider eben so, durch die fehlende Struktur stimmen die Daten im Wiki oftmals nicht und das es einem nicht aufgedrückt wird und wir die Single Source of Truth nicht haben. Du musst die Single Source of Truth durchdekliniert haben, bevor wir das für Automatisierungszwecke verwenden können. DNS updaten, Konfirmationen automatisiert erstellen. Das ist ein schmerzhafter und langwieriger Prozess, wo wir mittendrin sind. Jeder Engineer hat hier seine eigene Taktik. Ich schaue gerne auf die Systeme. Die anderen schauen gerne auf DHCP und DNS Systeme. Es ist derzeit ein wenig mühselig.

DANIEL STEUDLER: Das soll Netbox bei euch machen.

THOMAS FRITZ: Genau. Das Netbox soll dann aus unserer Sicht von Menschen befüllt werden. Manche Sachen werden wir automatisiert auslesen. Unser CheckMK liest dann automatisiert die Devices aus und weist uns dann auf Änderungen hin. Irgendwie haben wir schon ein wenig ein Überprüfungsregelkreis, weil Menschen auch einfach Fehler machen. Aufgrund der Meldungen sollen dann die Änderungen vorgenommen werden. Netbox wollen wir lieber von Hand befüllen, obwohl Menschen Fehler machen. Wobei man sagen muss, dass durch die Struktur von Netbox viele Fehler vermieden werden, weil Netbox einfach nicht zulässt den Namen doppelt vergeben werden können, ganz banale Fehler. Oder wenn wir Leitungen verwenden, welche schon von jemand anderem verwendet werden. Da bietet Netbox eine gute Struktur, die uns weiterhilft. Für andere Sachen wie falsche Interfaces, das sehen wir dann ganz gut im Überwachungstool, da muss es immer mal wieder einen Abgleich geben. Das ist der Regelkreis, den ich dort präferiere.

FELIX KUBLI: Habt ihr ein einheitliches Naming?

THOMAS FRITZ: Ja da haben wir ein Schema, das wird aber für nichts herangezogen. Es hilft nur dem Engineer, dass er sich keine IPs merken muss. Ob es ein Router ist, „R“ für Router, „S“ für Switch, dann eine Indexnummer und drei Buchstaben für die Region und nochmals eine Indexnummer, falls es mehrere Sites in der Region gibt und dann noch die Unterscheidung, ob es ein Access oder Core Router ist. Die Informationen, welche uns Netbox hier zur Verfügung stellt, wird verwendet, ist aber unabhängig von dem, was Netbox braucht, bzw. überschneidet sich mit diesem. Die strukturierte Namensgebung wurde recht bald eingeführt. Wir hatten zuerst lateinische Ausdrücke von Tieren. Das macht es aber für mich als Engineer nicht wirklich einfacher. Kann man machen, aber ich finde der CNAME sollte aussagekräftig sein, da ist DNS zum Glück ein wenig geduldig. Aber das ist auch „Out of my Scope“.

DANIEL STEUDLER: Was mich interessiert, ist die Customizability. Ihr habt sehr stark templatebare Sachen? Ein Switch unterscheidet sich nicht stark von einem anderen Switch?

THOMAS FRITZ: Das ist mal der Start. Danach wird sehr viel dazu gebaut. Dabei verwenden wir automatisierte Systeme, welche uns die Konfigs runterziehen und die Diffs darstellt, wenn sich was ändert. Wir sind hier relativ schnell in der Lage, wenn das System weg ist, ein Ersatz-System wieder hochzunehmen.

DANIEL STEUDLER: Mit dem macht ihr derzeit Desaster-Recovery.

THOMAS FRITZ: Genau

DANIEL STEUDLER: Zum Nachhaken, würdest du sagen eure Systeme sind ziemlich einheitlich?

THOMAS FRITZ: Was das Grundgerüst betrifft, sind wir natürlich schon einheitlich. Wir haben auf jedem Router fünfzehn verschiedene Kunden darauf, die alle ein wenig anders angebunden sind. Wir haben etwa zehn Varianten, wie wir Kundensysteme anbinden können. Wir haben fünf durchdekliniert. Aber dann gibt es immer noch Sonderlösungen, die man leider durchziehen muss. Diese Interfaces sind unterschiedlich. Aber wie im Backbone alles zusammenhängt, dort ist alles einheitlich, aber es wächst natürlich alles. Was den Access betrifft, dort ist es so, dass alles gleichartig ist. Dort ist es so, dass die Konfig sogar aus dem CRM kommt. Bis auf die Hostnamen und IP ist alles gleich. Dort ist alles einheitlich um es wartbar zu halten. Im Access-Bereich sehr stark, im Backbone eher weniger. Das reinschreiben von Beschreibungen machen wir nur fürs Troubleshooting, das könnte man sich sparen. Aber es ist ganz angenehm.

DANIEL STEUDLER: Uns interessiert nun noch die Pain-Points. Ist bei euch schwierig, da ihr derzeit in einem transienten Zustand seid.

THOMAS FRITZ: Ich kann euch hier schon Pain-Points nennen, unabhängig von der Init7, aus meiner Sicht. Ein Pain-Point ist, dass jeder Hersteller eine andere CLI hat. Es lässt sich schon automatisieren, aber es braucht halt immer ein Tool, welches diese CLI nachbildet, sich per SSH anmeldet und dann Änderungen macht. Es wäre cool, wenn sich hier NETCONF/YANG sich ein wenig mehr durchsetzen würde, und dass es hier besser wird, was die Schnittstellen angeht. Wir haben mehrere Hersteller, Cisco und Juniper. Dann ist es mühsam mit mehreren Templates zu arbeiten. Wie man das machen soll bei der Automatisierung? Ersetze ich die Konfig, mit dem Risiko das alles kaputtgeht, oder merge ich das Ganze? Das ist nach wie vor uneinheitlich. Grössere Anbieter sind eher unterstützt von Systemen. Kleinere Firmen im Netzwerkumfeld werden meist gar nicht berücksichtigt. Das sind so meine Pain-Points.

DANIEL STEUDLER: Habt ihr dann schon Erfahrungen gemacht mit NETCONF/YANG?

THOMAS FRITZ: Bei Init7 nicht, aber bei meinem vorherigen Arbeitgeber habe ich damit herumgespielt, aber das war auch relativ zäh. Schlussendlich lief auch die Automatisierung beim vorigen Arbeitgeber über SSH und NAPALM. Es fehlt aber leider ein wenig Struktur, jedes System ist hier ein wenig anders. NETCONF/YANG sollte doch langsam kommen. Es fehlt dort aber noch ein wenig an den Tools finde ich.

FELIX KUBLI: Ist dort eher das Problem das die bestehenden Tools NETCONF/YANG nicht unterstützen oder braucht es neue Tools.

THOMAS FRITZ: Ich habe das Gefühl es ist noch nicht in der Entwicklergemeinde angekommen. In dieser Gemeinde, die jetzt mit NAPALM arbeitet und das halt irgendwie Automatisieren wollten, bei denen ist das noch nicht so durchgedrungen. Ansible kann es irgendwie. Der Transportweg NETCONF/YANG geht ja ganz gut, aber bei NETCONF/YANG hast du die starke YANG-Struktur wo du irgendwo die Daten reinfüllen musst. Und die Daten musst du irgendwie aufbereiten, zum Beispiel aus YAML in ein YANG. So das ein simples, lesbares Format in YANG übergeht. YANG ist halbwegs menschenlesbar, aber nicht so, dass es Tools gibt, zwischen dem Anwender und YANG. Also YANG zum Menschen fehlt. Oder man müsste sagen, YANG zu Netzwerkengineer, da sich ein Mensch nicht mit YANG beschäftigen will.

DANIEL STEUDLER: Ein Netzwerkengineer kann sicher mehr anfangen mit dem Shell-Output, als mit NETCONF/YANG.

THOMAS FRITZ: Ja, wobei die Netzwerk Engineers haben realisiert, dass man mit Copy-Pasting von Konfigs nirgendwohin kommt. Es gibt im Netzwerk halt die unterschiedlichsten Konstellationen. Man spricht hier ja vom Snowflake-Netzwerk. Weil jedes Netzwerk sich vom nächsten unterscheidet. Darum gibt es auch tausende von Produkten. Und Netzwerkengineer dürfen meisten bei den Investitionen nicht mitreden. Es wäre toll, wenn es einen einheitlichen Standard geben würde.

DANIEL STEUDLER: Und dann gibt es noch verschiedene Controller-Software, welche die Hersteller verkaufen möchten.

THOMAS FRITZ: Du sprichst das Vendor-Lockin an. Wir sind davon aber weit entfernt. Wir setzen auf Open-Source, wo es nur geht. Das haben wir schon immer so gemacht und wird bei uns grossgeschrieben. Man kann sich dem auch hingeben und sagen, wir nehmen alles von Vendor A, aber dann bist du dem einfach ausgeliefert, falls dieser zum Beispiel die Preise erhöht.

DANIEL STEUDLER: Und Ihr kommt auch gut ohne diese Controller aus?

THOMAS FRITZ: Im Netzwerkengineering verwenden wir keine Controller. Durch das sind wir auch noch mehr technisch involviert, da diese Controller dort auch gewisse Sachen weg abstrahiert, aber das wollen wir uns auch nicht nehmen lassen und sind deshalb auch noch stark mit Templates unterwegs.

DANIEL STEUDLER: Prozesse, welche ihr habt, wie schaut ihr, dass diese verbessert werden? Zum Beispiel Router schneller aufsetzen.

THOMAS FRITZ: Bei uns geht es weniger ums Tempo. Bei uns geht es mehr in die Breite. Bei uns besitzen viele Netzwerk-Kollegen das Wissen, um Geräte aufzusetzen. Die Frage ist wie lange wir das noch aufrechterhalten können. Es geht dann halt oftmals ein wenig länger. Du hast Rechte. Es geht um die Geschwindigkeit. Oftmals muss man Quellen ausfindig machen, zum Beispiel wo die Templates liegen. Dies ein wenig zu vereinheitlichen, ist das Ziel. Damit wir schneller werden im Systeme aufsetzen und dass auch Kollegen schneller in der Lage sind Systeme aufzusetzen. Schneller aufsetzen und mehr Leute die Aufsetzen können, das geht Hand in Hand.

DANIEL STEUDLER: Habt ihr Tribal-Knowledge?

THOMAS FRITZ: Es ist alles abgelegt, aber die effizientesten Suchwege sind noch die Herausforderung für jemanden, der erst wenige Monate da ist. Jeder legt sein Wissen auch ein wenig anders ab. Es fehlen uns noch die Prozesse, dies zu vereinheitlichen. Wir erhoffen uns aber auch hier Hilfe von Netbox.

[*Wir präsentieren unser Elevator Pitch*]

...

THOMAS FRITZ: Ihr seid sehr stark abhängig von den eingesetzten Netzwerkgeräten. Wie arbeitet Kraken mit den Netzwerkgeräten?

FELIX KUBLI: Derzeit funktioniert es mit Netbox. Ziel ist das man es mit anderen Geräten vereinen kann. Mir fällte es gerade schwierig zu formulieren.

DANIEL STEUDLER: Also es sollen mehrere Quellen zu einer zusammengeführt werden.

THOMAS FRITZ: Den Konflikt sehe ich nur bei den Quellen, aber nicht bei den Geräten. Ich habe beim Netbox mein Gerät, eins mit dem Interface 17 irgend eine IP-Adresse, und jetzt gibts den Router 1 aber es ist an Interface 18. Kann das Kraken erkennen?

DANIEL STEUDLER: Es sind verschiedene Sachen. Konflikterkennung in den Quellen. Sind die Quellen korrekt? Und das Andere ist, stimmt das, was Konfiguriert ist, mit dem, was im Netzwerk ist überein?

THOMAS FRITZ: Das ist sehr spannend. Das kann wirklich gebraucht werden. Tut das dann über Mail oder eine andere Art notifizieren. Oder sogar automatisiert lösen?

DANIEL STEUDLER: Andenken lässt sich vieles. Wir sind derzeit noch den Markt am Analysieren und uns überlegen, was wir hier machen können.

FELIX KUBLI: Wir sind noch am Herausfinden in welche Richtung wir die bestehende Applikation weiterentwickeln werden. Das bestehende Tool kann zum Beispiel noch nicht mit YANG umgehen.

DANIEL STEUDLER: Eine Idee ist, dass es Konfiguration aus diesen Informationen generieren kann. Der bestehende Kraken sieht schon gut aus.

F.7. Zweites Gespräch mit Migros

Gesprächsteilnehmer

Name	Firma
Felix Kubli:	OST
Daniel Steudler:	OST
Luca:	MGB

Gespräch vom 23.11.2021. Das Gespräch wurde über Teams geführt und aufgezeichnet.

FELIX KUBLI: Euer Konfiggenerator, aus welchen Umsystemen holt der Informationen? Beziehungsweise mit welchen Daten befüllt ihr ihn?

LUCA HEGER: Je nach Router-Modell haben wir verschiedene Config-Templates hinterlegt. Dann die Unternehmung. Wir haben verschiedene VLAN Konzepte pro Unternehmung. Wir haben andere VLANs pro Unternehmung. Der Hostnamen hat ein Namenskonzept, aber den muss man händisch umsetzen. Ortschaft und Strassennamen sind SNMP-Location-Informationen, die werden beim Einloggen angezeigt im Banner.

DANIEL STEUDLER: Beim Hostnamen ist euer Konzept und das Tool setzt das automatisch um?

LUCA HEGER: Nein dieses Tool noch nicht. Wir haben die Hostnamen im Format <Unternehmung>-<Stadt>-<Strasse>-<Typ>, aber das muss man händisch eintragen.

DANIEL STEUDLER: Aber das sind Informationen, die du eigentlich hast oder?

LUCA HEGER: Ja, der Hauptuser dieses Tools ist ein On-Site-Engineer. Er weiss eigentlich alles. Ausser IP-Adresse und Subnetzmaske. Da erfolgt eine Rücksprache mit uns. Die Information für die IP's kommt aus dem IPAM[4]. Welche VLANs sie benötigen, wissen sie auch selber.

DANIEL STEUDLER: Je nach Feature die sie benötigen, brauchen sie andere VLANS[5]?

LUCA HEGER: Genau.

DANIEL STEUDLER: Was machen die VLANs?

[4]IP-Address-Management
[5]VLAN: Virtual Local Area Network (VLAN)

LUCA HEGER: Die VLANs kommen dann ins entsprechende VRF[6]. Die Subnetzbezeichnung ist nur ein Name. Die VLANs sind Layer 2 Unterteilung welche im VRF sind.

FELIX KUBLI: Wie funktioniert die Zuteilung von VRF an VLANs.

LUCA HEGER: Layer 3 haben wir nicht unter Kontrolle in den Filialen. Das wird von den Providern gemacht. Wir sagen den Providern welche VRFs wir benötigen und somit haben wir auf dem Trunk dann die entsprechenden VLANs.

DANIEL STEUDLER: Wofür braucht ihr die VRFs?

LUCA HEGER: Wir verwenden es für die Trennung der Routing-Instanzen. Wir haben zum Beispiel ein POS[7] VRF und wenn jetzt eine Kasse ins Datacenter muss, dann ist der Standard-Gateway die Firewall, und die Firewall kann dann das Ganze dann ins nächste VRF Routen. So funktioniert die Segmentierung bei uns auf Layer 3.

DANIEL STEUDLER: Und die VLAN Segmentierung spielt hier wie rein?

LUCA HEGER: Das ist Layer 2, es kann aber sein, dass mehrere VLANs auf demselben VRF landen.

FELIX KUBLI: Ihr speichert dann die generierten Konfigs?

LUCA HEGER: Ja, die Konfigurationsdateien werden einmal generiert und weg sind sie. Was wir machen sind Backups. Wir haben noch keinen Infrastructure-as-Code Ansatz. Zum Teil macht man aber auch Fehler. Zum Teil vergessen sie diese Zeilen.

DANIEL STEUDLER: Speichert ihr auch Informationen, welche nicht in dieser Konfig sind?

LUCA HEGER: Über den einzelnen Switch meinst du? Ich weiss nicht ob du das ansprichst, aber wenn du einen Switch aufsetzt, geht ein Mail ans NOC[8]. Dort heisst es der Switch ist neu, bitte in den Umsystemen eintragen. Dann muss er ins Prime[9], Monitoring, ISE[10] und ins IPAM.

DANIEL STEUDLER: Und wie kommt es in die Tools?

LUCA HEGER: Das ist ein händischer Prozess.

[6]VRF: Virtual Routing and Forwarding
[7]POS: Point of Sale, Kassenarbeitsplatz
[8]NOC: Network Operation Center
[9]Cisco Prime
[10]Cisco Identity Services Engine

DANIEL STEUDLER: Brauchst du dann noch mehr Informationen dazu?

LUCA HEGER: Hostnamen, IP und wo das er ist.

FELIX KUBLI: Und diese Informationen werden dann in all die Umsysteme eingetragen.

LUCA HEGER: Es braucht in allen Systemen den Hostnamen und die IP, ausser im IPAM und im PRTG[11] brauchen wir noch die Ortschaft. Da wir es dort hierarchisch haben wollen. Wir arbeiten oft mit dem Hostnamen. Zum Beispiel gibt es in der ISE Regeln, basieren auf den Hostnamen.

DANIEL STEUDLER: Was braucht ihr als IPAM?

LUCA HEGER: Bluecat.

DANIEL STEUDLER: Und was habt ihr dort drin?

LUCA HEGER: IPAM, DNS und DHCP. Wir machen einen statischen Eintrag für den Switch, dann ist der dort drin. Domäne ist immer, bzw. sollte immer identisch sein. Ist aber nicht immer so. MAC-Adresse verwenden wir nicht.

FELIX KUBLI: IP und Hostnamen ändern sich nicht?

LUCA HEGER: Doch, die können sich ändern und das bin ich jetzt am Aufräumen. Es kann vorkommen das es im PRTG nicht eingetragen wird oder im Bluecat nicht nachgeführt wird.

DANIEL STEUDLER: Wie gehst du vor wenn du einen Konflikt hast?

LUCA HEGER: Das was am meisten matcht gewinnt, bzw. auf dem Gerät nachschauen. Ist ein händischer Task. Unsere Datenbank ist momentan die ISE. Die ISE ist unser RADIUS[12] und TAGX. Wenn das Gerät an der ISE nicht eingetragen ist, kannst du dich nicht anmelden. Darum ist es unsere Device-Datenbank. Wir schauen nun was ist in der ISE aber im Prime nicht. Aber solange wir den Prozess nicht haben, ist das eine Aufräumarbeit.

DANIEL STEUDLER: Was gewinnt normalerweise?

LUCA HEGER: Wir haben RADIUS, wir haben Monitoring, Prime und Monitor. In diesem Fall würde der Radius gewinnen, weil wenn es dort falsch wäre könntest du dich nicht anmelden.

DANIEL STEUDLER: Es ist dort am richtigsten, wo es am meisten weh tun würde.

LUCA HEGER: Genau

[11]PRTG: PRTG Network Monitor, ein Netzwerküberwachungstool
[12]RADIUS: Remote Authentication Dial-In User Service

DANIEL STEUDLER: Und das ist in jedem Kontext der Radius?

LUCA HEGER: Ja du kannst schon davon ausgehen. Vorher war Prime die Quelle. ISE ist unsere Device-Datenbank für Cisco-Geräte. Prime unterstützt eigentlich nur Enterprise-Geräte. Wir haben einen Collector von Cisco welcher alle Geräte ausliest, Software und Geräteinformationen. Prime hat nicht alle Geräte unterstützt darum haben wir gewechselt.

FELIX KUBLI: Wie heisst das RADIUS Produkt?

LUCA HEGER: Das ist *Cisco ISE*.

DANIEL STEUDLER: Ein kleiner Wechsel letztes Mal haben wir auch noch über Cisco-DNA-Center gesprochen. Das sammelt nur Access-Points.

LUCA HEGER: Also LAN Sachen auch also auch Access-Switches.

DANIEL STEUDLER: DNA-Center nutzt ihr dann im Fall für?

LUCA HEGER: Management von WLAN-Access-Points. Verwalten Konfig-Push, Assurance, Troubleshooting. Zum Anderen, im Access würden wir gerne SD-Access verwenden. SD-Access ist eigentlich eine Abstrahierungs Schicht oben drüber. Momentan ist es einfach Layer 2. Du kannst wie nochmals den Layer 2 abstrahieren und Broadcast Domains nochmals kleiner machen. Und wir probieren das nun aus.

FELIX KUBLI: Cisco-DNA-Center sieht auch den Hostnamen und die IP?

LUCA HEGER: Ja, es ist ähnlich wie Prime. Du gibst die IP und dann macht es den Lookup. Es liest dann noch andere Informationen aus. Du könntest es stagen, updaten. Du kannst Assurance machen, du kannst gewisse automatisierungs Sachen machen. Es macht minimale Compliance-Checks. Du kannst zum Beispiel ein Golden Image setzen und es zeigt dir dann alle an, welche nicht auf dem Release sind.

DANIEL STEUDLER: Die Umsysteme, die haben alle eine API?

LUCA HEGER: Ja die haben alle eine API. Ich hab das mal gemacht in Form einer Arbeit.

FELIX KUBLI: Wie werden die IP's vergeben? Beim Aufschalten einer neuen Site, wie werden diese vergeben? Haben neue Sites einen gewissen Range zur Verfügung oder ist das DHCP?

LUCA HEGER: Ich bin mir da nicht 100% im Bild, aber wenn es eine neue Filiale ist, dann melden sie was für Netze sie brauchen. Wir geben ihnen dann vordefinierte Ranges. Wir haben hier Templates. Switches bekommen einfach eine statische IP im Data Netz zum Beispiel. Und dann haben wir die fest adressierten Router und HSLP Adresse. Die Switches sind in der Adresse 4-10.

DANIEL STEUDLER: Das sind jetzt alles private IPs, ihr verwaltet keine public IP's?

LUCA HEGER: Wir verwalten ganz viele public IPs. Aber für Webserver oder ähnliches.

DANIEL STEUDLER: Aber nicht im Filialen Netz?

LUCA HEGER: Nein dort nicht.

DANIEL STEUDLER: Ihr segmentiert dann die Subnetze in den Filialen?

LUCA HEGER: Genau, wir haben da Templates. Ich bin mir nicht hundert Prozent sicher. Aber zum Beispiel Genossenschaft Aare mit einem MM Markt bekommt dann ein /24 Netz. Und da ist ein Template hinterlegt. In diesem Falle im Range 80-229. Falls wir eines zurück gehen. Genossenschaft Aare hat ein /16 für sich. Und dann nimmt man ein Stück raus für Data.

DANIEL STEUDLER: Du nimmst ein Stück raus für Data, ein nächstes Stück raus für POS. Lokal sind die dann im VLAN getaggt. Und wenn nötig auch auf dem VRF.

LUCA HEGER: Genau. Wir sagen dem Provider was wir brauchen. Das ist ein Excel, welches von den On-Site-Leuten ausgeführt wird und die Bestellung beim Provider triggert.

FELIX KUBLI: Wo nehmen die On-Site Leute ihre Informationen her?

LUCA HEGER: Ich habe nie im On-Site gearbeitet von dem her, ich denke die haben eine Planung des Filialbaus. Wir haben Subito, also brauchen wir ein Subito Netz und entsprechend wird das aufgelöst.

DANIEL STEUDLER: Die VLAN-ID wird definiert, über was für ein Subnetz es ist und von welcher Firma es ist. Den Namen des Netzes, den wir hier sehen, muss vom Bluecat noch hier eingetragen werden.

LUCA HEGER: Genau, aber das hier ist nur ein Name, damit es die On-Site-Leute debuggen können. Das ist angenehmer für die Leute, die es troubleshooten, falls du ein Device hast welches nicht läuft. Wenn du nur das VLAN 2479 hast, musst du die Firma gut kennen um zu wissen, was das genau ist. Darum hier der Name.

DANIEL STEUDLER: Woher wird genau konfiguriert, welches VLAN was ist?

LUCA HEGER: Das wird auf dem Router gemacht.

FELIX KUBLI: Du hast gesagt du hast auch schon eine Arbeit gemacht, welche den Konfiggenerator erweitert. Kann dort der Konfiggenerator alle Informationen selbst zusammenziehen?

LUCA HEGER: Ja nicht ganz alle. Wir haben das Problem, angenommen man weiss, wo ein Gerät installiert wird, damit habe ich die Ortschaft. Ich zeig euch das schnell. Das ist nun im Verbund mit dem DNA-Center. Einer geht vor Ort und steckt das Gerät ein. Durch die Seriennummer kann ich das Gerät identifizieren. Person X sagt mir sie habe das Gerät installiert, z.B. in Fahrwangen und dann kann man die Filiale suchen und dann generiert es dir den Hostnamen. Was ich nicht machen konnte, sind zum Beispiel die Netze. Unser IPAM ist zu inkonsistent, was die Netze angeht. Wenn ich im IPAM danach suche, komme ich im schlimmsten Fall ganz viele Netze über und wie weiss ich nun welches welches ist. Mittlerweile habe ich die Lösung gefunden. Wenn du DHCP machst im Netz, dann wird dir DNA-Center bereits die richtige IP anzeigen bevor er gestaged ist. Die Umsysteme müssen entsprechend konfiguriert sein, zum Beispiel mit Identifiern, damit du von wenig Informationen auf viele Informationen kommst.

FELIX KUBLI: Damit du das Gerät in den Umsystemen eindeutig Identifizieren kannst?

LUCA HEGER: Nein, das wäre dann erst der nächste Schritt. Ich muss herausfinden welches Netz, welcher Hostname. Es kann sein, dass aufgrund vom Hostnamen die Unternehmung ändert. Dann muss ich das Template hinten dran ändern. Dann muss ich wissen welches Gerät es ist, um auf die richtige IOS-Version upzugraden. Welche VLAN es gibt, je nachdem welche Unternehmung es ist, ändert das. Wir haben die Schwierigkeit in dieser Riesen-Umgebung, dass die Umsysteme auch die Umsysteme entsprechend sein müssen.

DANIEL STEUDLER: Du kannst bei dem Output auch die VLANs auswählen?

LUCA HEGER: Ich hab das hinterlegt, wenn es die Unternehmung ausspuckt. All diese Netz-Sachen sind nur Beschreibungen und konnte ich noch nicht auslesen. In diesem Beispiel würde es zwar funktionieren, da der Name Informationen enthält, aber darauf kannst du nicht gehen. Sobald du in dieser Grösse etwas automatisieren möchtest, setzt es Standardisierung voraus.

DANIEL STEUDLER: Es beginnt eigentlich beim Prozess. Zuerst müssen die Daten auf einem Stand sein, auf dem man sie verwenden kann.

LUCA HEGER: Es gibt schon Umwege, aber Standardisierung macht alles einfacher.

DANIEL STEUDLER: Hast du ein Beispiel von Inkonsistenz?

LUCA HEGER: Das was wir vorher angeschaut haben. Das Device hat noch die falsche Domain.

DANIEL STEUDLER: Ich meinte wegen den VLAN-Namen.

LUCA HEGER: Da zum Beispiel, Shopping-Center in Spreitenbach, dort hat es ein Fitnesscenter, ITRIS hat dort ein Staging-Center, dann hat es zwei Filialen, eine im Shoppi und eine im Tivoli. Vielleicht ist im Namen FM oder MM, also was für einen Mensch schon Sinn ergibt. Man sieht, es ist ein Fachmarkt.

DANIEL STEUDLER: Es ist eigentlich eine Sammlung von Edgecases.

LUCA HEGER: Genau. Wir sagen mal, wir fangen mal mit etwas an, aber wir haben so viel. Restaurants, Fachmärkte, Büros, Erlebnisparks. Es gibt so viel.

DANIEL STEUDLER: Und die VLANs haben dann nicht einfach eine Nummer?

LUCA HEGER: Doch, die VLANs sind eindeutig pro Unternehmung.

DANIEL STEUDLER: Kannst du uns mal so eine Konfig zukommen lassen?

FELIX KUBLI: Die ist riesig.

LUCA HEGER: Kann ich schon, aber dort seht ihr einfach die Features.

DANIEL STEUDLER: Habt ihr eine Anleitung im Wiki, wie das installiert wird?

LUCA HEGER: Ja glaub schon.

DANIEL STEUDLER: Wenn der Konfiggenerator ausfallen würde, gibt es eine Anleitung, wie du das Ganze von Hand machen würdest?

LUCA HEGER: Nein das gibt es nicht.

FELIX KUBLI: Das Ziel soll ja dann sein, dass man einfach den Standort angeben kann und es dann alles Konfiguriert wird.

LUCA HEGER: Ja da sind wir dran. Der jetzige Konfiggenerator wird Geschichte sein, in ein zwei Jahren. Wir machen hier ein Greenfield-Approach. [...]

LUCA HEGER: Mein Problem ist ein wenig, die Wahrheit für verschiedene Sachen ist an verschiedenen Orten. Zum Beispiel IP ist das IPAM Chef. Aber IP brauchst du in anderen Systemen. Du hast Hostnamen, wer ist Chef von Hostnamen? Ist das eine Datenbank?

DANIEL STEUDLER: Ist es deterministisch oder nicht oder Random oder generiert.

LUCA HEGER: Ja es ist Herausfordernd. [...]

LUCA HEGER: Wir sind halt viele Unternehmungen und jeder hat ein wenig eine andere Organisationsstruktur. Wir haben On-Site, die können viel selbst, und andere die können fast nichts. Ich denke wichtig ist der Prozess, wenn man es von Anfang an richtig machen kann, bei neuen Einträgen. Das andere ist das Brownfield, was machst du mit dem was schon existiert.

DANIEL STEUDLER: Ja das merken wir.

Anhang G.

Pytest Output

,,

```
pytest inst−nodeps: /Users/danielsteudler/coding/ba21−nettowel−kraken/.tox/.tmp/package/1/kraken−1.0.0.tar.gz
pytest installed:
        attrs==21.3.0,Cerberus==1.3.4,certifi==2021.10.8,charset−normalizer==2.0.9,click==8.0.3,colorama==0.4.4,commonmark==0.9.1,coverage==6.1.2,deepdiff==5.5.0,distlib==0.3.4,filelock==3.4.2,Flask==1.1.2,Fla
        @
        file:///Users/danielsteudler/coding/ba21−nettowel−kraken/.tox/.tmp/package/1/kraken−1.0.0.tar.gz,MarkupSafe==2.0.1,mirakuru==2.4.1,multidict==5.2.0,ordered−set==4.0.2,packaging==21.3,pdns−api−client
        @
        git+https://github.com/pieterlexis/pdns_api_client−py.git@baa22a600e1313ed29e35e6ac9b754a0708ad771,platformdirs==2.4.1,pluggy==0.13.1,pony==0.7.14,port−for==0.6.1,psutil==5.8.0,psycopg2−binary==2.9
pytest run−test−pre: PYTHONHASHSEED='2751643569'
pytest run−test: commands[0] | pytest tests
============================= test session starts ==============================
platform darwin −− Python 3.9.9, pytest−6.2.4, py−1.11.0, pluggy−0.13.1 −− /Users/danielsteudler/coding/ba21−nettowel−kraken/.tox/pytest/bin/python
cachedir: .tox/pytest/.pytest_cache
rootdir: /Users/danielsteudler/coding/ba21−nettowel−kraken, configfile: pyproject.toml
plugins: cov−2.12.1, postgresql−3.1.1, mock−3.3.1
collecting ... collected 105 items

tests/test_cli.py::test_kraken_run_no_config PASSED [ 0%]
tests/test_cli.py::test_kraken_show_filters PASSED [ 1%]
tests/test_cli.py::test_kraken_show_systems PASSED [ 2%]
tests/test_cli.py::test_kraken_check_kc PASSED [ 3%]
tests/test_cli.py::test_kraken_check_kc_no_arg PASSED [ 4%]
tests/test_cli.py::test_kraken_check_mt PASSED [ 5%]
tests/test_cli.py::test_kraken_check_mr_no_arg PASSED [ 6%]
tests/test_collector.py::test_dns_only_in_powerdns_rest_needs_multilevel_filter PASSED [ 7%]
tests/test_collector.py::test_model_only_in_netbox_multilevel_for_powerdns SKIPPED [ 8%]
tests/test_collector.py::test_model_only_in_netbox_ip_in_all SKIPPED [ 9%]
tests/test_collector.py::test_name_and_ip_in_all_boolean_and PASSED [ 10%]
tests/test_collector.py::test_filters_available_in_all_connectors PASSED [ 11%]
tests/test_collector.py::test_filters_not_available_in_all_connectors PASSED [ 12%]
tests/test_collector.py::test_find_filter_support_per_query_key_all_have_filter PASSED [ 13%]
tests/test_collector.py::test_find_filter_support_per_query_key_not_all_have_filter PASSED [ 14%]
tests/test_connector_utils.py::TestConnectorUtils::test_add_if_not_exists_element_exists PASSED [ 15%]
tests/test_connector_utils.py::TestConnectorUtils::test_add_if_no_exists_new_element PASSED [ 16%]
tests/test_connector_utils.py::TestConnectorUtils::test_map_data PASSED [ 17%]
tests/test_connectors_fileconnector.py::test_fileconnector_json PASSED [ 18%]
```

```
tests/test_connectors_fileconnector.py::test_fileconnector_json_filenotexists PASSED [ 19%]
tests/test_connectors_fileconnector.py::test_fileconnector_xml PASSED [ 20%]
tests/test_connectors_fileconnector.py::test_fileconnector_xml_filenotexists PASSED [ 20%]
tests/test_connectors_fileconnector.py::test_fileconnector_csv PASSED [ 21%]
tests/test_connectors_fileconnector.py::test_fileconnector_csv_filenotexists PASSED [ 22%]
tests/test_connectors_netbox.py::TestNetboxFilters::test_netboxfileconnector_has_expected_filters PASSED [ 23%]
tests/test_connectors_netbox.py::TestNetboxFilters::test_netboxapiconnector_has_expected_filters PASSED [ 24%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_getall PASSED [ 25%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_getmanybyquery_returns_empty_if_not_found PASSED [ 26%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_getmanybyquery_name PASSED [ 27%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_getmanybyquery_location PASSED [ 28%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_getmanybyquery_model PASSED [ 29%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_getmanybyquery_ip PASSED [ 30%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_raises_error_if_devices_file_is_missing PASSED [ 31%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_raises_error_if_ipam_file_is_missing PASSED [ 32%]
tests/test_connectors_netbox.py::TestNetboxFileConnector::test_netbox_fileconnector_returns_empty_if_files_empty PASSED [ 33%]
tests/test_connectors_powerdns.py::TestPowerDnsFilters::test_powerdnsapiconnector_has_expected_filters PASSED [ 34%]
tests/test_connectors_powerdns.py::TestPowerDnsFilters::test_prepare_query_values PASSED [ 35%]
tests/test_connectors_powerdns.py::TestPowerDnsFilters::test_remove_duplicates PASSED [ 36%]
tests/test_connectors_powerdns.py::TestPowerDnsFilters::test_cleanup_results PASSED [ 37%]
tests/test_connectors_powerdns.py::TestPowerDnsFilters::test_cleanup_results_returns_toplevel_name_directly PASSED [ 38%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_interface[spine2.kraken.network.garden.-spine2-Loopback0-0] PASSED [ 39%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_interface[spine1-gig-5.kraken.network.garden.-spine1-GigabitEthernet5-0]
    PASSED [ 40%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_interface[spine2-gig-1-1.kraken.network.garden.-spine2-GigabitEthernet1/1-0]
    PASSED [ 40%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_interface[spine2-gig-1-1_2000.kraken.network.garden.-spine2-GigabitEthernet1/1-2000]
    PASSED [ 41%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_hostname[spine2.kraken.network.garden.-spine2-Loopback0-0] PASSED [ 42%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_hostname[spine1-gig-5.kraken.network.garden.-spine1-GigabitEthernet5-0]
    PASSED [ 43%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_hostname[spine2-gig-1-1.kraken.network.garden.-spine2-GigabitEthernet1/1-0]
    PASSED [ 44%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_hostname[spine2-gig-1-1_2000.kraken.network.garden.-spine2-GigabitEthernet1/1-2000]
    PASSED [ 45%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_sub_interface[spine2.kraken.network.garden.-spine2-Loopback0-0] PASSED [
    46%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_sub_interface[spine1-gig-5.kraken.network.garden.-spine1-GigabitEthernet5-0]
    PASSED [ 47%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_sub_interface[spine2-gig-1-1.kraken.network.garden.-spine2-GigabitEthernet1/1-0]
    PASSED [ 48%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_convert_pdns_hostname_to_sub_interface[spine2-gig-1-1_2000.kraken.network.garden.-spine2-GigabitEthernet1/1-2000]
    PASSED [ 49%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_expand_interface_name[Gig1/0/1-GigabitEthernet1/0/1] PASSED [ 50%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_expand_interface_name[Gig2/1/2-GigabitEthernet2/1/2] PASSED [ 51%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_expand_interface_name[Fa1/0/1-FastEthernet1/0/1] PASSED [ 52%]
tests/test_connectors_powerdns.py::TestPowerDnsConverters::test_expand_interface_name[SomeInterface1/0/1-SomeInterface1/0/1] PASSED [ 53%]
tests/test_deep_diff_helpers.py::TestDeepDiffHelpers::test_solve_value_changed_from_attr_str_with_objects PASSED [ 54%]
tests/test_deep_diff_helpers.py::TestDeepDiffHelpers::test_solve_value_changed_from_attr_str_with_dicts PASSED [ 55%]
tests/test_deep_diff_object_linking_and_merging_sandbox.py::TestDeepDiffObjectLinkingAndMerging::test_deep_diff_result_no_difference SKIPPED [ 56%]
tests/test_deep_diff_object_linking_and_merging_sandbox.py::TestDeepDiffObjectLinkingAndMerging::test_deep_diff_result_different SKIPPED [ 57%]
tests/test_deep_diff_object_linking_and_merging_sandbox.py::TestDeepDiffObjectLinkingAndMerging::test_deep_diff_connected_with_one_link SKIPPED [ 58%]
tests/test_deep_diff_object_linking_and_merging_sandbox.py::TestDeepDiffObjectLinkingAndMerging::test_deep_diff_with_array_included SKIPPED [ 59%]
tests/test_deep_diff_object_linking_and_merging_sandbox.py::TestDeepDiffObjectLinkingAndMerging::test_with_root_list SKIPPED [ 60%]
tests/test_filters.py::TestFilters::test_all_implemented_filters_found PASSED [ 60%]
```

```
tests/test_filters.py::TestFilters::test_one_filter_as_dict PASSED [ 61%]
tests/test_filters.py::TestFilters::test_all_filters_of_a_connector_as_dict PASSED [ 62%]
tests/test_filters.py::TestFilters::test_query_filters_in_connector PASSED [ 63%]
tests/test_filters.py::TestFilters::test_query_filters_not_in_connector PASSED [ 64%]
tests/test_filters.py::TestFilters::test_use_connector_keyword_for_query PASSED [ 65%]
tests/test_filters.py::TestFilters::test_find_filter_by_consumer_keyword PASSED [ 66%]
tests/test_filters.py::TestFilters::test_find_filter_by_connector_keyword PASSED [ 67%]
tests/test_init_kraken.py::TestKrakenInit::test_init_kraken_default_config PASSED [ 68%]
tests/test_init_kraken.py::TestKrakenInit::test_init_kraken_netbox_fileconnector_has_correct_filenames PASSED [ 69%]
tests/test_init_kraken.py::TestKrakenInit::test_init_kraken_connectors_sorted_according_precedence PASSED [ 70%]
tests/test_mapper.py::TestMapper::test_raw_powerdns_to_mapped PASSED [ 71%]
tests/test_mapper.py::TestMapper::test_mapped_powerdns_to_normalized_data PASSED [ 72%]
tests/test_mapper.py::TestMapper::test_raw_netbox_to_mapped PASSED [ 73%]
tests/test_mapper.py::TestMapper::test_mapped_netbox_to_normalized_data PASSED [ 74%]
tests/test_merger.py::TestMergerAutoSolve::test_solve_diffs_automatically PASSED [ 75%]
tests/test_merger.py::TestMergerAutoSolve::test_merge_template_with_explicit_filepath PASSED [ 76%]
tests/test_merger.py::TestMergerAutoSolve::test_merge_with_objects PASSED [ 77%]
tests/test_merger.py::TestMergerAutoSolve::test_merge_with_device_merger PASSED [ 78%]
tests/test_merger.py::TestMergerAutoSolve::test_prefix_ip_merger PASSED [ 79%]
tests/test_merger.py::TestMergerAutoSolve::test_combined_merger PASSED [ 80%]
tests/test_merger.py::TestMergerAutoSolve::test_device_locations_merger PASSED [ 80%]
tests/test_merger.py::TestMergerAutoSolve::test_segmentation_prefix_merger PASSED [ 81%]
tests/test_sources.py::TestInventorySource::test_map_from_inventory PASSED [ 82%]
tests/test_sources.py::TestLocationSource::test_map_from_locations PASSED [ 83%]
tests/test_sources.py::TestLocationSource::test_mocked_input PASSED [ 84%]
tests/test_sources.py::TestLocationSource::test_dict_serialization PASSED [ 85%]
tests/test_sources.py::TestLocationSource::test_filter_by_keywords PASSED [ 86%]
tests/test_sources.py::TestNetboxPrefixSource::test_map_from_json PASSED [ 87%]
tests/test_sources.py::TestNetboxIPSource::test_map_from_json PASSED [ 88%]
tests/test_sources.py::TestNetboxIPSource::test_final_amount_of_devices PASSED [ 89%]
tests/test_sources.py::TestNetboxIPSource::test_interfaces_keys PASSED [ 90%]
tests/test_sources.py::TestNetboxIPSource::test_iterface_with_multiple_ips PASSED [ 91%]
tests/test_sources.py::TestSourceSegmentationConnector::test_maps_from_segmentation_file PASSED [ 92%]
tests/test_static_template.py::TestStaticTemplate::test_finds_the_right_attributes PASSED [ 93%]
tests/test_template_utils.py::TestTemplateUtils::test_apply_template PASSED [ 94%]
tests/test_validator.py::TestValidator::test_invalid_merge_template PASSED [ 95%]
tests/test_validator.py::TestValidator::test_valid_merge_template PASSED [ 96%]
tests/test_validator.py::TestValidator::test_invalid_kraken_config PASSED [ 97%]
tests/test_validator.py::TestValidator::test_valid_kraken_config PASSED [ 98%]
tests/test_validator.py::TestValidator::test_path_to_yaml_wrong PASSED [ 99%]
tests/test_validator.py::TestValidator::test_for_kc_parse_error PASSED [100%]

========================== 98 passed, 7 skipped in 1.10s ==========================
_____________________________________ summary _____________________________________
  pytest: commands succeeded
  congratulations :)
```