OST
Ostschweizer
Fachhochschule

Kraken

Ein Mediator für Datenkonflikte in der Netzwerkverwaltung

Bachelor-Arbeit (BA), Frühlingssemester 2021

Abteilung Informatik
Ostschweizer Fachhochschule OST, Rapperswil-Jona (ehem. HSR)
www.ost.ch

Autorinnen:
Julia Fritsche (julia.fritsche@ost.ch)
Méline Sieber (meline.sieber@ost.ch)

Betreuer: Prof. Beat Stettler, OST; Urs Baumann, OST

Experte: Basile Bluntschli, onway AG

Gegenleser: Stefan Richter, OST

Datum: 9. Juli 2021

1. Abstract

Einleitung: Netzwerke zu verwalten war und ist oft noch mühsame Handarbeit. Doch auch die Netzwerkwelt ist der Automatisierung unterworfen, mit dem Ziel, dass sich Netzwerke ähnlich wie Software programmieren lassen. Zu diesem Wandel gehört auch die Netzwerkverwaltung. Heute sind es häufig händisch gepflegte Textdateien, die den Zustand des Netzwerks abbilden – eine Quelle für Fehler, Inkonsistenzen und Unübersichtlichkeit.

Eine sogenannte "Single Source of Truth" (SSoT) soll im Automatisierungszeitalter Abhilfe schaffen: Ein einziges Inventar, das alle Netzwerkinformationen als strukturierte Daten zentral an einer Stelle verwaltet. Existierende Lösungen versprechen bereits eine solche SSoT. Doch im Arbeitsalltag gibt es selten ein Inventar mit alleinigem Anspruch an die Informationshoheit. Gerade in grossen Netzwerken existieren parallel solche Inventare, die für sich verbindliche Datenquellen sind, und überlappend diese "Single Source of Truth" darstellen.

Ziel: An diesem Punkt setzt die vorliegende Arbeit an: Gibt es mehrere, voneinander unabhängige SSoT-Inventare, ist das Problem der inkonsistenten Daten wieder da. Eine Applikation soll Daten von diesen verschiedenen Inventaren sammeln, auf ein gemeinsames Format bringen, sie vergleichen und Inkonsistenzen beheben. Als Resultat liefert die Software den Zustand des kompletten Netzwerkes, wie es die Inventare abbilden. Ziel ist, bestehende SSoT-Inventare nicht abzulösen, sondern vielmehr das fehlende Gesamtbild zu liefern – eine vergleichbare Lösung auf dem Markt gibt es noch nicht.

Ergebnis: Kraken ist im Kern eine Python-Applikation, die auf Basis des Data-Engineering im Netzwerkbereich Daten zusammenführt und aufbereitet. Sie lässt sich über die Kommandozeile oder als Web-Applikation bedienen. Kraken kann sich über APIs auf zwei Typen von SSoT-Inventaren verbinden, um deren Daten abzufragen. Von einem Inventar-Typ ist es auch möglich, die Daten offline einzulesen. Neben allen Daten kann auch ein Subset bezogen werden (Filterfunktion), beispielsweise nur Geräteinformationen, die zu einem gewissen Standort gehören.

Kraken normalisiert die geholten Daten auf eine eigens entwickelte Datenstruktur, erkennt Differenzen und löst diese auf. Dies geschieht auf zwei Varianten: Entweder automatisch über eine interne Logik von Kraken, oder manuell gesteuert von der Nutzerin über die Web-Applikation. Für den zweiten Fall generiert Kraken zuerst einen Konfliktreport, der alle gefundenen Differenzen speichert. Er kann jederzeit abgerufen werden, um daran weiter zu arbeiten. Sind alle Differenzen geklärt, erzeugt Kraken aus all den Daten eine End-Konfiguration – das Gesamtbild des Netzwerkes.

Für eine künftige Weiterentwicklung muss vor allem die Datenstruktur überarbeitet werden, auf die Kraken die Daten normalisiert: Sie erschwert, neue Systeme an Kraken anzubinden oder die Struktur selbst zu erweitern. Ausserdem holt Kraken zwar Daten von den SSoT-Inventaren, synchronisiert die Daten jedoch nicht zurück. Eine weitere Ergänzung wäre also, dass Kraken die End-Konfiguration auch auf die Inventare zurückschreiben könnte, um dort Fehler zu korrigieren.

Inhaltsverzeichnis

1. Abstract **3**

2. Aufgabenstellung **5**

3. Management Summary **7**
3.1. Ausgangslage 7
3.2. Vorgehen 7
3.3. Ergebnisse 8

4. Technischer Bericht **9**
4.1. Einleitung und Übersicht 9
4.2. Lösungsansatz 13
4.3. Umsetzung und Resultate 14
4.4. Ergebnisdiskussion 17
4.5. Ausblick und Weiterentwicklung 17

A. Literaturverzeichnis **20**

B. Abkürzungsverzeichnis **21**

C. Projektspezifische Begriffe **22**

Tabellenverzeichnis

4.1. Übersicht zur Marktsituation 12

2. Aufgabenstellung

1. Ausgangslage

Im Management von Netzwerken werden wie in jedem anderen IT-Prozess Daten verarbeitet. Diese Daten sind oft in verschiedenen Systemen vorhanden und müssen von den Aktoren zusammengetragen und geprüft werden. Bei der Ausführung des Prozesses durch eine Person wird dies oft implizit und manuell durchgeführt. Die Person weiss durch Erfahrung, welche Daten von welchem System benötigt werden und kann oft potentielle Probleme bei der Datenqualität feststellen. Dieses Wissen wird häufig auch prozessübergreifend in verschiedensten Workflows genutzt.

Bei der Automatisierung dieser Workflows kommen diverse Werkzeuge (Applikationen) zum Einsatz, wobei jedes einzelne dieser Werkzeuge wiederum verschiedene Systeme als Datenquelle nutzt und Modifikationen durchführt. In diesen automatisierten Abläufen fehlt jedoch das (menschliche) Wissen über die Datenqualität und die Systeme selbst. Die vormals manuelle, menschliche Arbeit lässt sich nicht direkt in einen automatisierten Prozess übersetzen.

Im Rahmen eines Forschungsprojekts wurde ein erster Prototyp einer Software namens "Kraken" entwickelt, der diesen automatisierten Prozess abbilden soll: Kraken bezieht Daten zu Netzwerkgeräten von verschiedenen Systemen, normalisiert sie und führt sie zusammen. Ebenfalls wurden erste Schritte in die Richtung einer automatisierten Qualitätskontrolle durchgeführt, insbesondere der Konflikterkennung: Ziel wäre, dass die Software Inkonsistenzen in den Daten erkennt, vermelden und potentiell auflösen kann.

2. Aufgabe

Während der Bachelorarbeit soll dieser Prototyp verbessert und erweitert werden, um zusätzliche Anforderungen zu unterstützen.

2.1 Qualitätskontrolle

Im Kontext einer erhöhten Automatisierung der Netzwerke ist die Datenqualität von grosser Wichtigkeit -- Daten müssen in einem einheitlichen, strukturierten Format vorliegen und konsistent sein, ohne Doppelungen oder Fehler, um Prozesse automatisieren zu können. Leider entspricht die vorhandene Datenbasis oft noch nicht diesen erhöhten Anforderungen, was zu schwerwiegenden Folgefehlern oder Umgehungslösungen führt.

Die wichtigste Qualitätskontrolle ist deswegen die Konsistenzprüfung der Daten in allen angebundenen Systemen.

Konkret bedeutet das, dass teilredundante Daten miteinander verglichen und allfällige Konflikte erkannt werden sollen. Die vorhandene und sehr rudimentäre Konflikterkennung des Prototyps soll verbessert werden: Die Software soll über Konflikte informieren können (Konfliktreport), aber auch manuell oder automatisch diese Konflikte beheben können. In einem weiteren Schritt soll es möglich sein, manuell oder teilautomatisiert Korrekturen an den Quellsystemen selbst vornehmen zu können. Diese Korrekturen in den Quellsystemen sind jedoch nicht Teil der Qualitätskontrolle selbst und stellen nicht den Schwerpunkt der Aufgabe dar.

Werden Daten zusammengeführt, soll das Wissen des Nutzers mit einfliessen, das er über die Datenqualität in den jeweiligen Systemen hat. Es soll möglich sein, dass der Nutzer festlegen kann, ob ein System beim Zusammenführen der Daten präferiert werden soll. Zusätzlich soll der Nutzer Systeme festlegen können, die für spezifische Kategorien von Informationen präferiert werden sollen.

2.2 Filter

Oft werden für Automatisierungsworkflows nicht die Informationen über das gesamte Netzwerk benötigt,
sondern lediglich über ein Gerät oder einen ganzen Standort. Es soll deshalb möglich sein, mithilfe von Filtern nur die Informationen über gewisse Teile des Netzwerks abzufragen, die effektiv benötigt werden.

Ein grundlegendes Problem ist dabei, dass die potentiellen Filterinformationen nicht in jedem System verfügbar sind und somit die Filter nicht identisch an die Quellsysteme übergeben werden können. Es muss also ein Mechanismus entwickelt werden können, dass die Daten von allen Systemen geholt werden, die zu einer spezifischen Filtersuche gehören, selbst wenn nicht alle Systeme den entsprechenden Filter aufweisen.

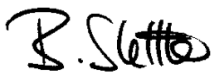
2.3 REST API

Um die Daten für verschiedene Automatisierungsworkflows zur Verfügung zu stellen, soll eine REST API entworfen werden. Diese soll alle grundlegenden Funktionalitäten von Nettowel-Kraken enthalten.

2.4 Show-Case

Es soll eine kleine Beispielapplikation entwicklet werden, über die Kraken benutzt werden kann, um eine sinnvolle und möglichst nachvollziehbare Aktion auszuführen. Das kann beispielsweise eine Kommandozeilen- oder eine Web-Applikation sein.

Rapperswil, den 23.Juni 2021



Prof. B.Stettler

OST
Ostschweizer Fachhochschule

Oberseestrasse 10
8640 Rapperswil
Switzerland

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

3. Management Summary

3.1. Ausgangslage

Mit der fortschreitenden Digitalisierung werden nicht nur Netzwerke, sondern auch deren Verwaltung immer komplexer. Dies führt dazu, dass Informationen zu den Netzwerkgeräten in verschiedenen Inventaren abgelegt sind, welche nicht miteinander kommunizieren und somit teilweise veraltete, duplizierte oder komplett falsche Daten enthalten. Dies kann in grossen Netzwerken zu Fehlern führen, welche nicht nur aufwändig, sondern auch teuer zu beheben sind.

Um diese Differenzen zwischen den verschiedenen Inventaren bzw. Systemen zu finden und anschliessend (manuell) in den umliegenden Systemen zu korrigieren, wurde in der vorliegenden Arbeit "Kraken" entwickelt. Wird Kraken regelmässig benutzt, ist es möglich, die Datenqualität der Inventare zu verbessern und somit die Fehlerquote zu verringern. Als Folge werden Netzwerkingenieurinnen effizienter und können sich auf die wesentlichen Aufgaben ihrer Arbeit konzentrieren.

3.2. Vorgehen

Kraken ist ein Werkzeug für die Netzwerkbetreuung und -instandhaltung, insbesondere für mittlere und grosse Netzwerke. Kraken verbindet sich mit den einzelnen Verwaltungssystemen, die für das Netzwerk- und Gerätemanagement eingesetzt werden und sammelt die gewünschten Daten. Die Applikation vergleicht daraufhin die Daten auf Inkonsistenzen. Die Nutzerin kann die Differenzen, falls solche existieren, entweder manuell beheben oder die automatische Konfliktlösung von Kraken nutzen. Das Resultat besteht aus einer Konfiguration, die als Vorlage für die Korrektur der Daten in den verschiedenen Systemen benutzt werden kann.

Zur Implementation der Kraken-Funktionalitäten wird primär die Programmiersprache Python eingesetzt. Für den effektiven Datenvergleich, aber auch für die Kommunikation mit den verschiedenen Systemen, kommen diverse externe Software-Bibliotheken zum Einsatz. Kraken kann auf zwei verschiedene Arten verwendet werden. Eine Web-Applikation, geschrieben in der Programmiersprache Javascript, ist vorrangig für die manuelle Verwendung über eine menschliche Nutzerin ausgelegt. Ein Kommandozeilen-Programm (CLI), geschrieben in Python, ist für den Einsatz in einem automatisierten Kontext vorgesehen (Maschine-zu-Maschine-Kommunikation).

3.3. Ergebnisse

Die zu Projektbeginn gesteckten funktionellen Ziele wurden erreicht. Diese waren die Konfiguration von Kraken, die Anbindung an die Verwaltungssysteme, die automatisierte und manuelle Konfliktlösung sowie die Anzeige eines Reports, welcher die Differenzen und Konflikte zwischen den Systemen darstellt. Dazu kommt die Web-Applikation, die insbesondere den Prozess der manuellen Konfliktlösung nutzerfreundlich abhandelt und für neue Funktionen einfach erweiterbar ist. Wie im Lösungsansatz bereits erwähnt, gehört auch das Kommandozeilen-Programm zu den Resultaten.

Durch den Eintritt diverser Risiken im Projektverlauf mussten bei der Implementierung von Funktionen, welche zur Erfüllung von nicht-funktionalen Anforderungen beitragen, Kompromisse eingegangen werden. Es ist nötig, das Normalisierungsformat zu überarbeiten, also das Data-Engineering bzw. die effektive Handhabung der Daten. Auch muss die Applikation auf ihre Skalierbarkeit in einem grösseren Netzwerk getestet werden sowie eine Rückmeldung für die Nutzerin eingebaut werden, falls Kraken keine Verbindung zu den gewünschten Systemen aufbauen kann. Grundsätzlich ist Kraken jedoch nach der Implementierung und Erfüllung dieser nicht-funktionalen Anforderungen für kleine Netzwerke einsatzbereit.

In einer Weiterentwicklung von Kraken sollte der Fokus auf den oben erwähnten nicht erreichten Zielen liegen, also primär das Data-Engineering zu überarbeiten. Ebenfalls von hoher Priorität ist die Funktion, die generierten Daten zu den jeweiligen Systemen zurück zu synchronisieren. So wird sichergestellt, dass Konflikte, welche Kraken oder die Nutzerin gelöst hat, in den Systemen nicht mehr bestehen und keine Fehler durch die manuelle Fehlerkorrektur selbst passieren. Ausserdem sollten mehr Systeme an Kraken angebunden werden, um das Werkzeug auch für mittlere und grosse Netzwerke einsetzen zu können.

4. Technischer Bericht

4.1. Einleitung und Übersicht

Ein gängiger Witz über die Art und Weise, wie Netzwerke unterhalten und gewartet werden, lautet: "Was ist der grösste Wechsel im Netzwerkmanagement der letzten 20 Jahre? Antwort: Der Wechsel von Telnet zu SSH".[1] Sprich: Es hat sich nichts geändert. Eine weitere Blüte besagt, dass Netzwerkgeräte (manuell) als einzigartige Schneeflöckchen konfiguriert seien, mit einzigartigen, nicht-standardisierten Einstellungen, und Netzwerk-Ingenieurinnen sich brüsten würden, einmalige Änderungen am Netzwerk anbringen zu können.[2]

Lange Zeit waren Netzwerkgeräte geschlossene Systeme und das Terminal (CLI) war die einzige Schnittstelle dazu. Konfigurationen und Befehle mussten in Skripts integriert, Terminal-Ausgaben von Hand geparst werden. Netzwerke warten, ändern, unterhalten war häufig Handarbeit und somit extrem fehleranfällig und inkonsistent.[3] Doch nach dem Wechsel von Telnet zu SSH ist seit einigen Jahren eine grössere Transformation der Netzwerkwelt im Gange: Software-ähnlich definierte Netzwerke, also Netzwerke, die sich wie Software programmieren und konfigurieren lassen.[4] Die digitale Transformation hat auch die Netzwerkwelt erfasst, weg von der fehleranfälligen Handarbeit, weg vom Handwerk hin zur Fabrik, also zur Netzwerkautomation.

Teil dieses Transformationsprozesses ist, wie Geräteinformationen und -konfigurationen verwaltet werden. In Zeiten der Handarbeit sind dies mitunter sorgfältig gepflegte Excel- oder Textdateien, die beispielsweise festhalten, an welchem Standort welche Geräte wie konfiguriert sind. Auch hier gilt das Problem: Je mehr manuelle Arbeit dahintersteckt, desto schwieriger ist es, konsistente und fehlerlose Daten zum Zustand des Netzwerkes zu erhalten. Die Information zu einem Netzwerkstandort ist beispielsweise über mehrere Dateien oft mehrfach verteilt – doch die Angaben aktuell und konsistent zu halten ist manuell aufwändig und fehleranfällig. Dazu kommt, dass bei händisch verwalteten Dateien das Wissen um deren Aufbau und Organisation oft nur bei der Person liegt, die sich effektiv um diese Dateien kümmert.

Im Zeitalter der Netzwerkautomation soll die sogenannte **"Single Source of Truth" (SSoT)** beitragen, diese Probleme zu beheben: Ein Inventar, das alle Informationen zu einem Netzwerk zentral an einer Stelle verwaltet, in Form von strukturierten Daten. Es kann idealerweise Auskunft zum intendierten und aktuellen Zustand des Netzwerkes geben. Aus diesem Speicher

[1] I.e. Vom Zugriff auf Netzwerkgeräte über ein unsicheres, unverschlüsseltes Protokoll (Telnet) hin zu einer verschlüsselten, authentifizierten Variante (Secure Shell Protocol, SSH) [Ede18, S. 4].

[2] "Today, most network devices are configured as unique snowflakes (having many one-off non-standard configurations), and network engineers take pride in solving transport and application issues with one-off network changes that ultimately make the network not only harder to maintain and manage, but also harder to automate." [Ede18, S. 18]

[3] Denn die CLI ist für Menschen gemacht, nicht für Maschine-zu-Maschine-Kommunikation (z.B. für die Automation): Vgl. [Ede18, 29f.].

[4] Die Definition von SDN (Software-Defined Networks) ist schwer zu fassen und hat sich die letzten Jahre hin gewandelt – zum Begriff und zu den grösseren Umwälzungen der Netzwerkdomäne siehe deswegen Kapitel 1 und 2 in [Ede18].

lassen sich dann nach Bedarf beispielsweise Konfigurationen für einzelne Netzwerkgeräte erzeugen, die wiederum auf die entsprechenden Geräte eingesetzt werden.[5]

Ziel wäre eine einzelne Plattform, eine "Single Source of Truth", welche die manuell verwalteten Datenspeicher und einzelne, dezidierte Werkzeuge ersetzt, die heutzutage schon teil-automatisiert Informationen zu Netzwerkgeräten verwalten (z.B. DNS-Informationen). In den letzten Jahren sind Lösungen auf den Markt gekommen, die versprechen, eine solche Plattform zu sein und eine "Single Source of Truth" herzustellen zu können.

Blue Planet Inventory von Ciena: Ciena ist ein US-Unternehmen, das Netzwerk- und Telekommunikationsausrüstung anbietet. Das Produkt "Blue Planet Inventory" von Ciena[6] verspricht, "Single Source of Truth" eines Netzwerkinventars zu sein, es gibt jedoch wenig Informationen, wie dies genau erzeugt wird. Blue Planet ist kostenpflichtig und proprietär.

IPFabric: IPFabric ist ein tschechisches Unternehmen, deren Plattform namens IP Fabric eine Netzwerk-Topologie kartografieren und zu einer Gesamtsicht zusammenzufügen können soll. Das Produkt ist kostenpflichtig und proprietär.[7] IPFabric soll anhand vordefinierter "Verification Checks" kontinuierlich prüfen, ob mit der gesamten Konfiguration alles in Ordnung ist. IPFabric unterstützt jedoch nur eine gewisse Anzahl von Herstellern und Plattformen[8], da die Plattform direkt mit den Netzwerkgeräten interagiert.

Nautobot von Network to Code: Nautobot[9] ist ein Fork des NetBox-Projektes und wird vom US-Unternehmen "Network to Code" entwickelt. Nautobot ist sehr neu – die Version 1.0 wurde Anfang März 2021 veröffentlicht. Nautobot ist als Rundum-Sorglospaket rings um die Netzwerk-Automation konzipiert. Teil davon ist die sogenannte "Golden Configuration", welche als Plugin zu Nautobot fungiert.[10]

Das Plugin verbindet sich direkt mit Netzwerkgeräten, um deren Konfiguration zu sichern und abzufragen und vergleicht diese mit einer gewünschten Konfiguration. Es soll zudem Konfigurationen anhand von Templates erzeugen können.[11] Die "Golden Configuration" stützt sich stark auf Nornir[12] und ist auf eine Reihe von Herstellern wie Cisco oder Juniper beschränkt.

Nautobot ist noch in einem frühen Entwicklungsstadium. Es ist schwer einzuschätzen, wie gut Nautobot für grosse Netzwerke skaliert, insbesondere wenn tausende Netzwerkgeräte auf ihren gewünschten Zustand hin konfiguriert werden sollen.[13]

---

[5] Es gibt verschiedene Definitionen zur "Single Source of Truth", die hier gegebene basiert auf dieser: "A network source of truth is a repository of data that provides network managers and their network automation pipelines with authoritative insight into the intent of the network and the current state of the network. Intent data can be derived from configuration files, device inventory, cabling information, and other sources." [McG20, S. 2]

[6] `https://www.blueplanet.com/products/inventory.html`, Zugriff 18.3.2021.

[7] `https://ipfabric.io/product/network-verification/`, Zugriff 18.3.2021.

[8] `https://docs.ipfabric.io/matrix/`, Zugriff 18.3.2021.

[9] `https://www.networktocode.com/nautobot/`, Zugriff 18.3.2021. Nautobot wird hier etwas genauer ausgeführt, da es dem Konzept von Kraken am nächsten ist.

[10] `https://github.com/nautobot/nautobot-plugin-golden-config`, Zugriff 18.3.2021. "The golden configuration app is designed to let users know how compliant their network is and if their golden configurations that are generated from the rich data already stored in Nautobot are being implemented properly across the network." [Net21, S. 25]

[11] Siehe Beschreibung auf `https://github.com/nautobot/nautobot-plugin-golden-config`, Zugriff 18.3.2021.

[12] Eine Python-Library, um programmatisch und geräteunabhängig mit Netzwerkgeräten zu interagieren: `https://nornir.readthedocs.io/`, Zugriff 18.3.2021.

[13] Vgl. Hinweis in [McG20, S. 3].

Die eben umrissenen Lösungen versprechen, das *eine* System zu sein, die *eine* "Single Source of Truth", die alles andere ersetzt. Doch die Realität ist vielmehr, dass es mehrere "Single Sources of Truth" gibt: Es gibt im Arbeitsalltag kaum einen einzigen Punkt, der eine solche "Single Source of Truth" darstellt, sondern mehrere, sich überlappende Systeme, die für sich verbindliche Datenquellen sind, und die dann gemeinsam diese "Single Source of Truth" darstellen.[14] Gerade bei grösseren Netzwerken ist dies sinnvoll: Eine einzelne SSoT ist bei kleineren Netzwerken durchaus möglich, skaliert jedoch schlecht für grosse Netzwerke. Das Problem der manuellen Netzwerkverwaltung bleibt aber bestehen: Sobald es mehrere SSoTs gibt, kann dies auch zu Inkonsistenzen zwischen den Systemen führen – Router X fehlt beispielsweise in System A ein Interface, das jedoch in System B für Router X aufgeführt ist. Daten können dupliziert, falsch oder schlicht uneinheitlich vorliegen, dasselbe Problem ist also wieder da wie bei der ursprünglichen Netzwerk-Handarbeit.

An diesem Punkt setzt die vorliegende Arbeit – Kraken – an und ist der wesentliche Unterschied zu den existierenden Lösungen: Kraken soll ein Werkzeug sein, um Informationen zu Netzwerkgeräten aus unterschiedlichen "Single Sources of Truth" zu sammeln, die ebensolche Netzwerkgeräte verwalten. Kraken ist sozusagen der Punkt, an dem alle diese "Single Sources of Truth" zusammenlaufen und liefert dazu den **Gesamtüberblick aller verwalteten Netzwerkgeräte**.

Das heisst: Die bestehenden Systeme, die Netzwerkinformationen verwalten, soll Kraken *nicht* ablösen. Vielmehr sammelt Kraken Informationen von verschiedenen SSoTs, vergleicht deren Daten, weist auf Inkonsistenzen hin, behebt sie gegebenenfalls und generiert aus all den Daten ein Gesamtbild des kompletten Netzwerkes – die sogenannte End-Konfiguration.

Ein weiterer Unterschied: Kraken interagiert nicht direkt mit den Netzwerkgeräten, also den Routern, Switches oder anderen Netzwerkgeräten, um deren Konfigurationen abzufragen, sondern mit den Systemen, die ebendiese Netzwerkgeräte verwalten. Diese Systeme behandelt Kraken Hersteller-neutral: Das System kann eine Datei sein, in der Konfigurationen zu Netzwerkgeräten abgelegt sind, oder eine API, wie sie beispielsweise Netbox bereitstellt – eine Open-Source-Lösung, um Netzwerkgeräte zu verwalten[15]. Ziel ist, aus all diesen System die Informationen zu sammeln, zusammenzuführen und auf einen Nenner zu bringen. Das Resultat ist ein Gesamtbild zum intendierten Zustand eines Netzwerkes.

Tabelle 4.1 liefert eine Übersicht, wie sich Kraken zu den oben evaluierten bestehenden Markt-Lösungen bzw. Plattformen unterscheidet und verwendet dazu folgende Kriterien:

- Normalisierung: Die heterogenen Daten der verschiedenen Systeme liegen in einem strukturierten, gemeinsamen Format vor.
- Die Plattform liefert ein Gesamtbild aller im Netzwerk verwalteten Geräte.
- Konflikterkennung: Die Plattform informiert, falls inkonsistente Daten vorliegen.
- Konfliktlösung: Die Plattform vermag Inkonsistenzen automatisch oder manuell (über Nutzerinteraktion) beheben.
- Über eine REST-API kann mit der Plattform interagiert werden.
- Die Plattform ist quelloffen (open-source).[16]

[14] [McG20, S. 4]

[15] `https://netbox.readthedocs.io/en/stable/`, Zugriff 18.3.2021.

[16] Kraken ist derzeit closed-source – geplant war, die Software der BA unter einer Open-Source-Lizenz zu lizenzieren, doch dies war nicht möglich.

- Die Plattform nutzt verschiedene Netzwerk-Inventare bzw. Verwaltungssysteme, um Informationen zu sammeln.
- Die Plattform nutzt direkt die Netzwerkgeräte, um Informationen zu sammeln.
- Die Plattform ist die einzige SSoT.

	Blue Planet	IPFabric	Nautobot	Kraken
Normalisierung	✓	✓	✓	✓
Gesamtbild	✓	✓	✓	✓
Konflikterkennung	?	✓	✓	✓
Automatische Konfliktlösung	?	?	✗	✓
Manuelle Konfliktlösung	?	?	✗	✓
REST API	?	✓	✗	✓
Open-Source	✗	✗	✓	?
Datenquelle: Verwaltungssysteme	?	✗	✗	✓
Datenquelle: Netzwerkgeräte	?	✓	✓	✗
Plattform ist einzige SSoT	✓	✓	✓	✗

Tabelle 4.1.: Übersicht zur Marktsituation

Kraken ist somit näher am Konzept der tatsächlichen "Single Source of Truth", wie sie bereits charakterisiert wurde[17]: Die Applikation berücksichtigt die "Single Sources of Truth", statt den (fiktiven) Anspruch zu haben, eine "Single Source of Truth" sein zu können. Kraken ist funktioniert also wie ein Mediator, um Datenkonflikte zwischen all diesen Systemen zu lösen.

Mit diesem Ansatz erleichtert Kraken auch die digitale Transformation des Netzwerkbereichs: Selten ist es aufgrund Legacy-Komponenten möglich, direkt ein komplettes Netzwerk zu automatisieren und auf eine einzelne SSoT umzusteigen. Zudem laufen in einem Netzwerk oft alte und neue Komponenten nebeneinander, ohne den Bedarf zu haben, erstere auszutauschen.

Kraken erlaubt es, die von Hand gepflegte, in einer Datei vorliegende Netzwerkkonfiguration zu belassen, aber auch bereits andere Lösungen wie Netbox gleichzeitig zu verwenden, die näher an der Netzwerkautomatisierung sind, und trotzdem daraus ein Gesamtbild des Netzwerks zu schaffen. Das ermöglicht eine schrittweise Automatisierung der Netzwerkverwaltung und ist letztlich näher am Arbeitsalltag einer Netzwerkingenieurin.

[17] [McG20, S. 4]

4.2. Lösungsansatz

Kraken ist im Bereich des **Data-Engineering** angesiedelt, mit einem speziellen Fokus auf Netzwerke. Denn letztlich geht es darum, Daten von Systemen abzufragen (den SSoTs), sie aufzubereiten, potentielle Inkonsistenzen zu erkennen, aufzulösen und daraus ein Gesamtbild des Netzwerkes zu schaffen.[18] Um dieses Data-Engineering zu erreichen sind verschiedene Komponenten nötig:

- Anbindung an Systeme ermöglichen und deren Daten abfragen (alle Daten oder ein Subset)
- Daten auf ein gemeinsames Format bringen (Normalisierung)
- Daten automatisch konsolidieren und Inkonsistenzen selbständig auflösen (automatische Konfliktlösung)
- Daten konsolidieren, aber bei Inkonsistenzen die Nutzerin mit einbeziehen (manuelle Konfliktlösung)
- Kraken innerhalb einer automatisierten Pipeline verwenden (Verwendung über Kommandozeile, z.B. via Script)
- Kraken innerhalb eines Browsers verwenden (REST-API, Web-Applikation)

Konnektoren sollen als Teil von Kraken die Anbindung an Systeme ermöglichen. Diese sollen so konzipiert sein, dass eine Netzwerkingenieurin mit Python-Erfahrung selber weitere Systeme an Kraken anbinden kann. Oft benötigt eine Nutzerin nicht die Information über das gesamte Netzwerk, sondern nur einen Teil, etwa Informationen zu einem bestimmten Standort oder einem Gerätetyp. Kraken soll deswegen nicht nur alle Daten von den Systemen holen können, sondern auch ein definiertes Subset über eine Filterfunktion. So kann die Nutzerin nach Schlüsselwörtern suchen und beispielsweise alle Informationen abfragen, die zum Standort "Rapperswil" gehören.

Zentral im Data-Engineering ist, dass die Daten strukturiert und in einem **einheitlichen Format** vorliegen, weswegen Kraken die heterogenen Daten der Systeme normalisieren muss. Hierbei sind der Aufbau des Formats sowie die Datenstruktur, die Kraken intern verwendet, zu beachten. Ideal wäre ein Format, das möglichst nahe an der Netzwerkdomäne ist – hier bietet sich das OpenConfig-Modell an.[19]

Die **automatische Konfliktlösung** soll ohne Interaktion der Nutzerin Daten vergleichen, zusammenführen und daraus die End-Konfiguration erzeugen. Für dieses Vorgehen sind zwei Anforderungen an die Netzwerkadministratorin nötig, die Kraken konfiguriert:

1. Sie ist informiert über die Qualität der Daten pro System. Beispielsweise liefert System A qualitativ bessere Daten als System B.
2. Geht es um spezifische Kategorien an Informationen, weiss sie, von welchem System welche Daten qualitativ zu bevorzugen sind. Beispielsweise sind die Informationen zu IP-Adressen aus System A vorzuziehen, da sie aktueller sind als solche aus System B.

Somit ist es nötig, **zwei Metriken** zu definieren, damit Kraken auf Basis der Datenqualität eines Systems eine Entscheidung treffen kann. Des Weiteren muss ein Algorithmus entwickelt

[18] Vgl. die O'Reilly-Definition von "Data Engineering" in [Fur21].
[19] Das Projekt verwendet eine Reihe wiederkehrender Begriffe und eigene Definitionen. Diese sind im Abschnitt C erklärt.

werden, wie Kraken die normalisierten Daten überhaupt zusammenführt. Dazu gehört der Vergleich der Daten, aber auch wie Kraken selbständig entscheidet, von welchem System im Falle von Inkonsistenzen welche Daten zu bevorzugen sind. Es muss auch entwickelt werden, was überhaupt als "Inkonsistenz der Daten" gilt, bei der Kraken eine Entscheidung treffen muss. Die Ausgabe dieses Prozesses soll eine End-Konfiguration sein: Die konsolidierten Daten, die den erwünschten Zustand des Netzwerkes abbildet (oder einen Teil davon, falls ein Filter verwendet wurde), nachdem die Inkonsistenzen aufgelöst wurden.

Die **manuelle Konfliktlösung** kommt zum Zug, wenn eine Nutzerin darüber entscheiden soll, wie bei Inkonsistenzen vorzugehen ist. In diesem Fall entscheidet Kraken nicht selber, sondern überlässt der Nutzerin die Wahl, ob beispielsweise für Router X besser die Gerätebezeichnung von System A statt System B gewählt werden soll. Idealerweise soll die manuelle Konfliktlösung innerhalb einer grafischen Oberfläche erfolgen, da hier nicht das Ziel ist, den Ablauf zu automatisieren.

Die konsolidierten Daten sollen schliesslich in einem Format ausgegeben werden, das nahe an der Netzwerkdomäne ist – das bereits erwähnte OpenConfig-Format bietet sich hierbei an. Trotzdem sollte es möglich sein, in einer weiteren Entwicklung von Kraken wählen zu können, wie dieses **Ausgabeformat** zu gestalten ist.

Die automatische und manuelle Konfliktlösung bedingen zwei verschiedene Verwendungsarten. Eine **Kommandozeilen-Applikation (CLI)** soll die automatische Konfliktlösung ermöglichen, so dass diese nahtlos in eine Netzwerk-Automatisierungspipeline integriert werden kann. Für die manuelle (aber auch automatische) Konfliktlösung bietet sich eine **Web-Applikation** an, mit der die Nutzerin interagiert. Eine REST-API[20] ermöglicht zudem die programmatische Einbindung von Kraken in einen grösseren Automatisierungskontext.

4.3. Umsetzung und Resultate

Zu Beginn der vorliegenden Arbeit existierte bereits ein "Proof of Concept" (PoC), also ein Code-Entwurf des Projekts, das am INS (Institute for Networked Solutions, Fachhochschule OST) entwickelt und auf welchem aufgebaut wurde. Die resultierende Applikation ist im Kern eine reine Python-Applikation, die Web-Applikation verwendet Javascript, HTML und CSS.

Über eine YAML-Datei konfiguriert die Nutzerin Kraken: Sie legt dort fest, ob sie Daten automatisch oder manuell zusammenführen möchte. Zusätzlich erfasst sie alle Systeme, von denen Kraken Daten abfragen soll. Dazu gehören die entsprechenden Zugangsdaten wie beispielsweise API-Token oder Passwörter, aber sie definiert auch die Präzedenz: Diese legt die Datenqualität eines Systems im Vergleich zu den anderen Systemen fest – je höher der Wert, desto besser.

Um die Konfigurationsdatei sowie das Merge-Template[21] auf Korrektheit zu überprüfen, wird ein Validator eingesetzt. Dieser Validierungsvorgang prüft mit Hilfe der Bibliothek Cerberus[22] anhand eines vordefinierten Schemas, ob die Kraken-Konfiguration oder das Merge-Template der Nutzerin keine formalen Fehler enthält, also dem korrekten Format entspricht.

[20] Vgl. [Ede18, 28ff.]
[21] Siehe weiter unten in der Umsetzung.
[22] `https://docs.python-cerberus.org/en/stable/`, Zugriff 17.6.2021.

Die **Schnittstelle zu den Systemen** ist eine `AbstractConnector`-Klasse, von der alle konkreten Implementationen ableiten, wenn beispielsweise ein neuer Konnektor in Kraken eingebaut werden soll. Zwei verschiedene Systeme sind nun als Konnektoren-Klassen in Kraken integriert: Die Netbox- sowie die PowerDNS-Instanz. Die Netbox-Daten lassen sich über eine API-Schnittstelle (`NetboxAPIConnector`) oder über Dateien einlesen (`NetboxFileConnector`), die PowerDNS-Daten über eine API (`PowerDnsApiConnector`).

Um ein Subset an Daten von den Systemen zu holen, verfügt Kraken über **Filter**: Eine `AbstractFilter`-Klasse liefert die Basisstruktur, von der alle konkreten Filter ableiten müssen, nach deren Schlüsselworte Kraken suchen kann. Die Wahl für eine abstrakte Basisklasse erfolgte aus zwei Gründen:

1. Der Schlüssel eines Schlüssel-Werte-Paars für die Suche ist je nach System anders. Sucht die Nutzerin nach `ort=Rapperswil`, lautet diese beispielsweise für System A `site=Rapperswil` und für System B `location=Rapperswil`.
2. Nicht alle Systeme können nach allen möglichen Schlüsseln suchen. Beispiel: PowerDNS kann nach DNS-Informationen suchen, Netbox nicht – trotzdem müssen von allen Systemen Daten geholt werden, um die End-Konfiguration zu generieren.

Die implementierten Filter in Kraken geben nun vor, nach welchen Schlüsselbegriffen bzw. Kriterien überhaupt über alle Systeme hinweg gesucht werden kann: Name, Ort, Gerätemodell, IP-Adresse (IPv4 und IPv6) und DNS-Adresse. Soll ein neuer Filter eingeführt werden, leitet dieser von der abstrakten Klasse ab und definiert das Schlüsselwort, das für die Nutzerin sichtbar ist. Jeder Konnektor implementiert diejenigen Filterklassen, nach denen das dahinterliegende System überhaupt suchen kann, und setzt das passende Schlüsselwort, welches das angeschlossene System zur korrekten Suche benötigt.[23] Als Folge ist es nicht möglich, frei nach einem Begriff zu suchen – die Heterogenität der Systeme ist schlicht zu gross.

Der "Multilevel-Filter" löst das Problem, dass nicht alle Systeme nach allen Schlüsselwörtern suchen können, aber trotzdem von allen Systemen die Daten geholt werden müssen. Mit dem Multilevel-Filter erfolgt nun die Suche in zwei Schritten. Im ersten Schritt holt Kraken die Daten von denjenigen Systemen, die über den Filter verfügen. Danach wird aus den Resultaten der Name extrahiert, der auch ein Gerät eindeutig identifiziert – es ist ein Wert, nach dem alle Systeme suchen können und müssen. Im zweiten Schritt holt Kraken anhand des Namens in allen verbleibenden Systemen die restlichen Daten.

Damit die Daten von den verschiedenen Systemen miteinander verglichen werden können, müssen diese in einer einheitlichen Struktur abgebildet werden. Aus den gelieferten Rohdaten werden als Erstes nicht gebrauchte Informationen herausgefiltert und in ein gemeinsames **Normalisierungsformat** gebracht. Diese normalisierten Daten sind die Grundlage für den darauffolgenden Datenvergleich und dem End-Konfigurationsformat.

Ein wichtiger Schritt im Data-Engineering ist, Daten zu bereinigen und aufzubereiten. Das erfolgt in Kraken mittels der **DeepDiff-Bibliothek**, die darauf spezialisiert ist, tief verschachtelte Datenstrukturen zu vergleichen, was bei den normalisierten Daten von Kraken der Fall ist. DeepDiff erkennt Differenzen zwischen Datensätzen und weist sie verschiedene Kategorien zu. Wenn beispielsweise der Eintrag X in System A mit dem in System B verglichen wird, und Eintrag X existiert in B noch nicht, erhält das Resultat die Kategorie `dictionary_item_added`. Wichtig ist anzumerken, dass DeepDiff nur Differenzen findet, aber noch nicht auflöst – das ist die Aufgabe der automatischen und manuellen Konfliktlösung.

[23] Ein Filter hat also zwei Schlüsselwörter: Eines, das die Nutzerin sieht (`consumer_keyword`) und eines, das das System für die Suche versteht (`connector_keyword`).

Die **automatische Konfliktlösung** verwendet zwei Metriken in Kombination mit den gefundenen DeepDiff-Differenzen, um automatisch die Daten zu bereinigen. Ein eigens entwickelter Algorithmus löst anhand zweier Metriken die Differenzen auf:

1. Globale Präzedenz eines Systems: Die Netzwerkadministratorin definiert in der Kraken-Konfiguration (vor der eigentlichen Verwendung) die existierenden Systeme und legt deren Präzedenz fest, und zwar in Form einer Zahl von 0-255.
2. Merge-Template: Eine YAML-Datei bildet die Struktur ab, auf die Kraken die Daten normalisiert. Die Netzwerkadministratorin kann für fast[24] jedes Element in dieser Struktur den Namen eines Systems hinterlegen, falls für dieses Element ein bestimmtes System bevorzugt werden soll. Das Merge-Template überschreibt also die globale Präzedenz.

Bei der Verarbeitung der DeepDiff-Resultate verwendet Kraken primär die globale Präzedenz der Systeme. Nur bei Konflikten kommt das Merge-Template zum Zug: Dann nämlich, wenn sich Eintrag X in System A und B unterscheidet, und einer der beiden Einträge nicht nur aus einem leeren String besteht.[25] Kraken konsultiert in diesem Fall das Merge-Template um zu entscheiden, von welchem System der Eintrag gewählt werden kann. Ist nichts vermerkt, wird erneut der Eintrag mit der höheren Systempräzedenz gewählt.

Als Puzzleteil in der gesamten Netzwerkautomatisierung bietet Kraken eine **CLI** an, so dass die Applikation beispielsweise ohne menschliche Interaktion automatisiert verwendet werden kann. Allerdings ist hierbei nur die automatische Konfliktlösung möglich, da die manuelle Konfliktlösung nur über menschliche Interaktion erfolgen kann. Zusätzlich muss die Nutzerin über die CLI sowohl die Kraken-Konfiguration als auch das Merge-Template gleich mitgeben. Eine gefilterte Suche ist auch implementiert. Um die Leichtgewichtigkeit dieser CLI zu gewährleisten, wurde auf eine Anbindung an die Datenbank verzichtet.

Wenn hingegen viele Unsicherheiten bestehen, bietet die **manuelle Konfliktlösung** eine Alternative, welche über eine Web-Applikation benutzt wird. Anstatt die Differenzen automatisch zu lösen, erstellt Kraken mit diesen Differenzen zuerst einen sogenannten Konfliktreport, der in der Datenbank abgespeichert wird. Die Nutzerin kann die Differenzen in der Web-Applikation einsehen und entscheiden, welche Differenzen von den jeweiligen Systemen in die End-Konfiguration übernommen werden sollen. Um immer einen möglichst aktuellen Stand der End-Konfiguration bereitzustellen, kann sich die Nutzerin diese jederzeit generieren und anzeigen lassen. Die manuelle Konfliktlösung ist nur in der Web-Applikation vorhanden, da die CLI keine benutzerfreundliche Darstellung und Nutzung der Funktion bereitstellen kann.

Die **Web-Applikation** bietet auch die Möglichkeit, die automatisierte Konfliktlösung sowie die Validierung der Konfigurations-Dateien zu benutzen. Für beide Lösungsmöglichkeiten gilt, dass via die implementierten Filter auch nur ein Subset der zu vergleichenden Daten geholt werden kann. Damit deckt die Web-Applikation die beiden Hauptfunktionen des Tools ab. Die Basis dafür legen Vue.js als Javascript-Framework sowie TailwindCSS als CSS-Framework.

[24] Fast: Bei gewissen Elementen in der Hierarchie drohen Inkonsistenzen, werden für Kind-Elemente Daten von einem anderen System verwendet als für das Eltern-Element. In diesem Fall darf das Merge-Template nicht zur Anwendung kommen.

[25] Ein Konflikt ist also `kraken` vs. `Kr4ken`. Kein Konflikt, nur Differenz: `""` vs. `kraken` oder `kraken` vs. `""`. Vgl. auch Abschnitt C zu projektspezifischen Begriffen.

4.4. Ergebnisdiskussion

Ergebnisse Wie bereits in den Resultaten ersichtlich, wurden alle geplanten Hauptfunktionen umgesetzt. Zusammengefasst besteht Kraken aus folgenden Komponenten:

- **Kraken-Core**: Im Kraken-Core sind die Hauptfunktionen integriert: Kraken holt die Daten von den Systemen, vergleicht sie in einem normalisierten Format und löst Inkonsistenzen entweder automatisch oder manuell (von der Nutzerin) auf. Davor wird das Werkzeug über YAML-Dateien konfiguriert, welche von einem Validierungsvorgang auf deren Korrektheit geprüft wird. Diese Funktionen werden in einem Package zusammengefasst, welches von der REST API angesprochen wird. Die CLI ist ebenfalls ein Teil dieses Kraken-Core-Packages.
- **REST API** Um von aussen auf die Funktionalitäten des Kraken-Core zugreifen zu können, ist eine API implementiert. Sie legt Funktionen wie die automatische und manuelle Konfliktlösung, Filtermöglichkeiten für die Datensammlung sowie die Validierung der Konfigurationsdateien frei.
- **Web-Applikation**: Die Web-Applikation dient als grafische Oberfläche, wird insbesondere für die manuelle Konfliktlösung eingesetzt und kommuniziert über die REST API mit dem Kraken-Core. Zusätzlich zu den integrierten Funktionen im Kraken-Core ist es möglich, die End-Konfiguration im JSON-Format zu exportieren.

Die nicht-funktionalen Anforderungen sind grösstenteils umgesetzt und konnten gemessen werden – einige wurden jedoch nicht erfüllt. Zum Beispiel wurde eine Datenbank für die manuelle Konfliktlösung implementiert, welche ursprünglich aufgrund der Performance-Einbusse nicht geplant war. Unter anderem fehlen auch Rückmeldungen für die Nutzerin, falls Kraken keine Verbindung mit den Systemen herstellen kann.

Probleme Ursprünglich war geplant, die Konnektoren und das Mapping der Rohdaten mehrheitlich von der bereits vorhandenen Code-Basis zu übernehmen. Da die Daten jedoch für den Vergleich mit DeepDiff nicht geeignet waren, musste ein Normalisierungsformat konzipiert und umgesetzt werden, was zum Anfang des Projektes viel Zeit gekostet hat. Ebenfalls komplizierter als gedacht war die Implementation der Filter-Funktionen in den jeweiligen Konnektoren: Jedes System besitzt eine eigene API, die angesprochen wird, weswegen auch die Suchfunktionen auf die Systeme zugeschnitten sein müssen. Wie bereits bei den nicht-funktionalen Anforderungen erwähnt, wurde für die manuelle Konfliktlösung eine nicht-geplante Datenbank implementiert. Diese Entscheidung wurde zu Beginn der Construction-Phase getroffen, da eine Persistierung der Daten für diese Funktion unumgänglich ist. Dies hat jedoch ebenfalls zu einem höheren Aufwand geführt.

4.5. Ausblick und Weiterentwicklung

Konnektoren Im Rahmen dieser Arbeit wurden weniger Konnektoren implementiert als ursprünglich angedacht: Hauptgrund waren Probleme mit dem Normalisierungsformat, wie der vorherige Abschnitt bereits ausführte. Deswegen sind im Endprodukt dieser Arbeit nur drei Konnektoren (zwei APIs und ein File-Konnektor) implementiert, auf deren Daten-Grundlage auch das Normalisierungsformat entwickelt wurde. Die Vorlagen für die nicht implementierten Konnektoren sind noch immer im Code abgelegt und wurden in einen "Legacy"-Ordner

verschoben. Diese fehlenden Konnektoren sollten in Zukunft ebenfalls korrekt implementiert werden.

Normalisierungsformat Da weitere Konnektoren auch Daten für neue Attribute und Netzwerkstrukturen liefern, müsste das Normalisierungsformat potentiell mit jedem Konnektor erweitert werden, was für eine effiziente Erweiterung von Kraken nicht optimal ist. Die Suche nach dem idealen Normalisierungsformat lag jedoch nicht im Scope dieser Bachelorarbeit, weswegen dies als eine weiterführende Aufgabe für eine weitere Studien- oder Bachelorarbeit denkbar wäre. Diese Suche würde eine Re-Evaluation der Datenstruktur, aber auch allgemein die Frage der Datenhaltung enthalten.

Filter Eine zukünftige Weiterentwicklung von Kraken müsste sich auch darum kümmern, die Sprache für Suchanfragen in Kraken zu erweitern. Die derzeitige Implementation ermöglicht nur ein Bool'sches AND der Suchanfrage, wünschenswert wäre auch ein Bool'sches OR. In Zukunft sollten also Abfragen wie "Suche alle Daten zu den Geräten, die entweder leaf1 oder leaf2 heissen" möglich sein. Zusätzlich sollte die Suchanfrage so verfeinert werden, dass es möglich ist, nach einem Teilbegriff wie "leaf" zu filtern, um trotzdem die Ergebnisse von "leaf1" und "leaf2" weiterverabeiten zu können. Wichtig wäre auch die Erweiterung der verschiedenen Filter um andere Teilbereiche der Netzwerke wie z.B. nach VLAN, Trägermedium oder Geräte-Hersteller.

Multilevel-Filter Zum Endzeitpunkt des Projektes wird der Namens-Filter als Hilfsfilter angewendet, um sicherzustellen, dass von allen Systemen Daten zurückgesendet werden, selbst wenn der eigentliche Suchfilter nicht überall vorhanden ist. Um dies auf die eigene Netzwerkumgebung zuschneiden zu können, wäre es von Vorteil, einen anderen Hilfsfilter festlegen zu können.

Manuelle Konfliktlösung Der Konfliktreport wird nach der einmaligen Abfrage der Systeme mit allen Daten und Differenzen in der Datenbank abgespeichert. Da die Konflikte und Differenzen jedoch zeitversetzt gelöst werden können – teils Tage oder Wochen danach – besteht die Gefahr, dass die Daten zwischenzeitlich in den Systemen geändert wurden und somit nicht mehr aktuell sind. Das Ziel wäre hier, die manuelle Konfliktlösung so umzuschreiben, dass jederzeit die korrekten Daten vorhanden sind, der Fortschritt der Konfliktlösung aber trotzdem nicht verloren geht.

End-Konfigurationsformat festlegen Zum Endzeitpunkt der Arbeit entspricht das End-Konfigurationsformat dem Normalisierungsformat – dies ist jedoch sehr spezifisch und passt nicht in alle Netzwerkumgebungen. Somit sollte in einer Weiterentwicklung von Kraken das End-Konfigurationsformat abgekoppelt werden und selbst definierbar sein.

Usability-Tests Der Web-Applikation wurden im Zeitraum der Arbeit keine Usability-Tests unterzogen, da sich die Aufgabenstellung besonders auf den Core von Kraken konzentriert. Da die Web-Applikation zum Endzeitpunkt des Projektes primär funktionell ist, sollte die Applikation vor der Weiterentwicklung von Aussenstehenden getestet werden, um besonders die Benutzeroberfläche der manuellen Konfliktlösung möglichst intuitiv und ansprechend darstellen zu können.

Rücksynchronisation Mit der Konfliktlösung ist bereits ein erster Schritt in die Daten-Synchronisation zwischen den Systemen getan. Es können jedoch immer noch Fehler bei der (manuellen) Übertragung der korrekten Konfiguration in die Ursprungssysteme passieren, was wiederum zu neuen Inkonsistenzen führen kann. Eine automatisierte Rücksynchronisation der Daten würde diesen Prozess ablösen und nicht nur Zeit sparen, sondern auch die Korrektheit der Daten erhöhen. Eine mögliche Hürde hier wäre, dass sicherheitsbedingt nicht alle

Systeme einen automatischen Import zulassen, was für eine konsistente Umsetzung Voraussetzung wäre.

Weitere Use-Cases Das Projekt definierte zusätzliche Use-Cases, von denen nahezu alle auf einem geschützten Bereich mit einem Login basieren. Dieser Bereich sollte Funktionen wie Statistiken, Nachverfolgung von Suchanfragen und bereits gelösten Konfigurationen sowie das Abspeichern von Filtern für den späteren Gebrauch zur Verfügung stellen. Diese Funktionen werden auch neue Security- und Datenschutzthemen aufwerfen, die es zu lösen gilt. Diese Use-Cases sind primär für die Nutzung über die Web-Applikation gedacht, wohingegen sich viele der obenstehenden Weiterentwicklungen auf den Kraken-Core beziehen.

A. Literaturverzeichnis

[Ede18] Edelman, Jason, Lowe, Scott S. and Oswalt, Matt. *Network Programmability and Automation. Skills for the Next-Generation Network Engineer*. O'Reilly, 2018.

[Fur21] Furbush, James. *Data engineering: A quick and simple definition*. 2021. URL: https://www.oreilly.com/content/data-engineering-a-quick-and-simple-definition/ (besucht am 12. 06. 2021).

[McG20] McGillicuddy, Shamus. *A Network Source of Truth Promotes Trust in Network Automation*. 2020. URL: https://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/white-paper-sp-ema-trust-in-network-automation.pdf (besucht am 18. 03. 2021).

[Net21] Network to Code, Nautobot. *Nautobot Solution Guide: Source of Truth for Network Automation*. 2021. URL: https://www.networktocode.com/wp-content/uploads/2021/02/Nautobot_Technical_Solution_Guide.pdf (besucht am 18. 03. 2021).

Das Oktopus-Bild auf der Titelseite, das auch als Logo von Kraken verwendet wird, stammt von der Nutzerin user2104819 von Freepik.com.

B. Abkürzungsverzeichnis

API Application Programming Interface. Schnittstelle, um auf ein System zuzugreifen oder eine Software-Bibliothek, um sie in einer eigenen Applikation zu verwenden.

CLI Command-Line-Interface. Kommandozeilen-Programm.

CSS Cascading Style-Sheets. Sprache, um Webseiten zu gestalten.

DNS Domain Name System. Beantwortung von Anfragen zur Namensauflösung, d.h. es führt die Nutzerin mit der Eingabe `www.google.com` im Browser zum richtigen Server.

FQDN Fully Qualified Domain Name. Die vollständige und eindeutige Adresse einer Domäne, endet auf einen Punkt. Beispiel: `myhost.example.com.`

IP-Adresse Internet Protocol Adresse. Eine numerische Adresse für Geräte in einem Netzwerk, welche sie eindeutig identifizierbar macht.

INS Institute for Networked Solutions, Fachhochschule OST.

LoC Lines of Code. Anzahl Codezeilen ohne Kommentare oder Leerzeilen.

PoC Proof of Concept. Code-Entwurf, der eine gewisse Funktion demonstrieren soll.

REST (API) Representational State Transfer (REST), Application Programming Interface (API). Kommunikationsmittel für verteilte Systeme und Applikationen.

SDN Software-Defined Network. Netzwerke wie Software konfigurieren und programmieren.[1]

SSoT Single Source of Truth. Ein einzelnes Inventar bzw. Datenquelle mit verbindlichen Informationen zu Netzwerkgeräten.

YAML YAML Ain't Markup Language. Sprache zur Datenserialisierung, populär für Konfigurationsdateien.

[1] Die Definition von SDN (Software-Defined Networks) ist schwer zu fassen und hat sich die letzten Jahre hin gewandelt – zum Begriff und zu den grösseren Umwälzungen der Netzwerkdomäne siehe deswegen Kapitel 1 und 2 in [Ede18].

C. Projektspezifische Begriffe

Da in Kraken beispielsweise der Begriff "Konfiguration" mehrere Bedeutungen haben kann[1], sind für diese Arbeit folgende Begriffsdefinitionen festgelegt:

Automatische Konfliktlösung Kraken löst automatisch Differenzen und Konflikte anhand der Präzedenz. Für letzteres kann zusätzlich ein Merge-Template verwendet werden.

Consumer Kraken enthält zwei sogenannte Consumer: Eine Web-Applikation und ein Kommandozeilenprogramm (im nachfolgenden Bericht CLI genannt). Ein Consumer leitet Benutzereingaben an Kraken weiter und empfängt die Ergebnisse, um sie der Nutzerin darstellen zu können.

End-Konfiguration Die "Single Source of Truth", also das Ergebnis, welches Kraken nach der Zusammenführung der Daten liefert.

End-Konfigurationsformat Das Format, in welchem Kraken die End-Konfiguration ausgibt. Zum Abgabe-Zeitpunkt ist dieses identisch mit dem Normalisierungsformat.

Konfliktreport Ein Konfliktreport enthält alle Differenzen und Konflikte, welche zwischen den Systemen gefunden wurden.

Konnektor Ein Konnektor stellt die Verbindung zwischen Kraken und einem externen System her. Ebenfalls werden hier die ausgewählten Filter mitgegeben. Ein Konnektor liest die Daten vom externen System aus und bringt diese auf das Normalisierungsformat.

Kraken-Core Zum Kraken-Core gehört das interne Package `kraken`, welches die ganze Logik enthält – namentlich die Konnektoren, Konfiguration, Datenbank, Validierung der YAML-Konfigurationsdateien und die Generierung der End-Konfiguration inklusive den Merge-Vorgängen. Das Package liefert zusätzlich die CLI mit. Nicht dazu gehören die API und die Web-Applikation, welche ausserhalb des Packages, aber im gleichen Projekt und Repository abgelegt sind.

Kraken-Konfiguration YAML-Datei. Enthält Verbindungsinformationen zu allen Systemen (Kraken-Systemeinstellungen) sowie Angabe zur Merge-Strategie (manuell, auto).

Kraken-Systemeinstellung Angaben, wie Kraken zu einem oder mehreren Systemen eine Verbindung aufbaut. Enthält die Präzedenzangabe für die Merge-Strategie.

Manuelle Konfliktlösung Der Nutzerin wird jeder Konflikt und jede Differenz anhand eines Konfliktreports vorgelegt. Sie muss von Hand entscheiden, von welchem System der Eintrag gewählt werden soll.

Merge-Differenz, Differenz In System A und System B ist Eintrag X unterschiedlich. "Differenz" ist im grösstmöglichen Sinne definiert und sagt schlicht aus, dass Eintrag X "anders" ist: Unterschiedliche Datentypen, der Eintrag fehlt komplett, es existieren Werte, doch sie sind anders etc.

Merge-Konflikt, Konflikt Ein Konflikt ist eine Differenz, aber enger definiert und somit Subset einer Differenz: System A und System B haben beide für Eintrag X einen Wert und keiner von beiden ist ein leerer String. Beispiel: `NExus 9301` (System A, das E ist absichtlich gross) vs. `Nexus 9300` (System B) und umgekehrt.

[1] Kraken-eigene Konfiguration, Konfiguration von Umsystemen, Konfigurationen, die Kraken verarbeitet etc.

Merge-Strategie Angabe, wie die Daten zusammenzuführen sind. Es existieren folgende Möglichkeiten:

1. Nutzerin entscheidet bei jedem Konflikt und jeder Differenz (manuelle Konfliktlösung).
2. Kraken entscheidet bei Konflikten (automatische Konfliktlösung):
   a) Anhand des Merge-Templates.
   b) Anhand der Präzedenz, die in der Kraken-Konfiguration hinterlegt ist.

Merge-Template YAML-Datei. Die Struktur basiert auf dem Normalisierungsformat. Auf einer beliebigen Hierarchie-Ebene kann angegeben werden, von welchem System die Information zu präferieren ist.

Netbox Netbox ist eine Web-Applikation mit dem Ziel, Netzwerke zu managen. Es können unter anderem IP-Netzwerke und Geräte verwaltet werden.[2]

Normalisierungsformat Das Format, auf welches die Rohdaten der verschiedenen Systeme gemappt bzw. normalisiert werden. Dieses Format dient als Basis für die Weiterverarbeitung der Daten.

OpenConfig OpenConfig zielt darauf ab, herstellerunabhängige Schemata zu entwickeln, um Netzwerkkonfigurationen und -managementdaten darin abzubilden.[3]

PowerDNS PowerDNS ist ein Unternehmen, welches unter dem gleichem Namen Software für das Management von DNS-Zonen bereitstellt. Der Begriff wird in dieser Arbeit einfachheitshalber für das Produkt "Authoritative Server" benutzt, welches in der Testumgebung im Einsatz ist.[4]

Präzedenz Numerische Angabe, wie wichtig die Information des jeweiligen Systems zu gewichten ist. Je höher, desto wichtiger.

System, Umsystem Ein System bzw. Inventar, das Netzwerkgeräte verwaltet und zu dem Kraken eine Verbindung aufbaut – beispielsweise Netbox, PowerDNS oder eine Datei. Die Schnittstelle wird innerhalb der Kraken-Architektur als Konnektor bezeichnet.

Systemdaten, Daten Informationen, die ein System an Kraken liefern kann.

Validator Der Validator wird dazu benutzt, die Korrektheit von YAML-Dateien, in diesem Fall der Kraken-Konfiguration und des Merge-Templates, zu überprüfen und im Fehlerfall der Netzwerkadministratorin eine entsprechende Rückmeldung zu geben.

[2] `https://netbox.readthedocs.io/en/stable/`, Zugriff 18.6.2021.
[3] Siehe `https://www.openconfig.net`, Zugriff 11.6.2021.
[4] `https://www.powerdns.com/`, Zugriff 18.6.2021.