HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

INSTITUTE FOR
NETWORKED SOLUTIONS

Automatisiertes Troubleshooting

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2017

Autoren: Raphael Jöhl und Nico Vinzens

Betreuer: Beat Stettler

Automatisiertes Troubleshooting

Abstract

Das Troubleshooting von Problemen in Netzwerken ist heutzutage zum grössten Teil immer noch manuelle Arbeit. Der Netzwerk Engineer sammelt dabei Informationen indem er den Zustand der Verbindungen überprüft oder die Konfiguration der Geräte durchforstet.
Dieses altmodische Paradigma soll durch ein System ersetzt werden, welches dank dem Konzept von «Event Driven Automation» das Troubleshooting automatisieren kann.

Zu diesem Zweck wurden in einer Analysephase Tools identifiziert, die ein solches System ermöglichen. Das Hauptaugenmerk lag dabei darauf, dass die Tools untereinander kompatibel sind und so viel vom Netzwerk wie möglich abstrahieren, sodass kein «Screen-Scraping» mehr nötig ist.
Die Tools erfüllen folgende Funktionen: Das Erkennen von Problemen im Netzwerk (Syslog), das Sammeln von Daten zu einem bestehenden Problem (napalm, napalm-logs), das Auslösen eines Workflows, der auf den aktuellen Problemfall zugeschnitten ist (Salt) und das Überprüfen der im Workflow durchgeführten Massnahmen (napalm). Zudem soll zu jedem Problemfall ein Log erstellt und in einer Datenbank (mongoDB) abgespeichert werden. War der Workflow nicht erfolgreich, so wird ein Techniker benachrichtigt (über Slack), der das Problem manuell lösen muss.
Die Hauptarbeit dieser Studienarbeit bestand darin, das Gerüst aufzubauen welches all diese Tools zu einem sinnvollen Ganzen verbindet.

Das Resultat der Arbeit ist OATS (Open Source Automated Troubleshooting System), welches die Kommunikation zwischen den Tools ermöglicht und über Python-Skripts verschiedene Workflows abbildet, welche in der Lage sind Netzwerkprobleme automatisch zu beheben.
OATS wird auch Gegenstand einer weiterführenden Bachelorarbeit sein, welche das automatisierte Troubleshooting noch weiter ausbauen wird. So wäre es zum Beispiel interessant, die benutzten Tools direkt weiter zu entwickeln.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Management Summary

Automatisiertes Troubleshooting

OATS (Open-Source Automated Troubleshooting System)

Diplomanden	Nico Vinzens, Raphael Jöhl
Examinator	Beat Stettler
Themengebiet	Networks, Security & Cloud Infrastructure

Ausgangslage

Viele Arbeitsschritte bei Fehlerbehebungen könnten eigentlich automatisiert werden. So Arbeitet eine nette Stimme bei der Support Hotline aller grossen Telekom Konzerne zuerst die Checkliste ab. Das gleiche hat auch in den anderen IT Bereichen seine Gültigkeit. So sammelt ein Netzwerk Engineer bei einem Support Ticket zuerst meistens nur Informationen. Wo befindet sich das Gerät? In welchem Subnetz ist es? Kann ich es via ICMP erreichen? Solche Arbeiten eignen sich sehr gut zum Automatisieren. So können bei einem Fehlerfall die üblichen Tasks bereits abgearbeitet werden. Sind genügend Informationen vorhanden können automatisch Lösungsversuche gestartet werden oder das Problem temporär umgangen werden, so dass der Techniker am nächsten Tag ausgeschlafen die Bereinigung abschliessen kann.

Aufgabe: Entwickeln eines Systems, welches dank dem Konzept von «Event Driven Automation» das Troubleshooting automatisieren kann. Über Sensoren sollen Workflows angestossen werden, welche wiederum sogenannte Aktoren ansteuern. Es muss nicht eine komplette Eigenentwicklung geben, da es viele Komponenten bereits als Open Source Lösungen gibt. Im gesunden Rahmen sollen diese Projekte durch Integrationen und Anpassungen zu einer Gesamtlösung verschmelzen.

Vorgehen/Technologien

In einer ersten Analysephase wurden diverse Netzwerktechnologien untersucht, mit denen es möglich ist, Informationen in einem Netzwerk zu sammeln. Hauptzweck dieser Phase war es, sich einen Überblick über die Problemdomäne zu verschaffen. Wie sammelt man Informationen in einem Netzwerk? Welche Probleme treten in einem Netzwerk auf? Welche Geräte werden in einem Netzwerk eingesetzt und wie funktionieren diese?

In der zweiten Analysephase wurde aufgrund der Ergebnisse der ersten Analysephase vertieft nach Tools und Frameworks gesucht, die es einem leichter machen können, sich im Netzwerk zurechtzufinden. Dabei wurden folgende Tools identifiziert und anschliessend in OATS zum Einsatz gebracht:

- **Syslog**: Löst Events direkt auf den Netzwerkgeräten aus: agiert als Sensor von OATS, der z.B. merkt wenn ein Gerät unerwartet runterfährt.
- **Napalm-logs:** Verarbeitet die Syslog Events und extrahiert deren Daten. Sendet die Daten weiter an Salt.
- **Salt:** Entscheidet aufgrund der erhaltenen Daten, welcher Workflow ausgeführt werden muss und führt diese aus. Innerhalb der Workflows werden Informationen gesammelt und Befehle an die Netzwerkgeräte gesendet. Das Informationssammeln und Senden der Befehle geschieht mittels der napalm-library.
- **Napalm:** Eine Library, die eine Schnittstelle für einfache und herstellerunabhängige Zugriffe auf Netzwerkgeräte anbietet. Ist direkt in Salt integriert.
- **mongoDB:** Datenbank für das Abspeichern von Daten zu den ausgeführten Workflows und dem zugrundeliegenden Netzwerk (Topologie-Daten).
- **Slack:** Chat-Software zum Benachrichtigen der Techniker, falls ein menschlicher Eingriff nötig ist.

Ergebnisse

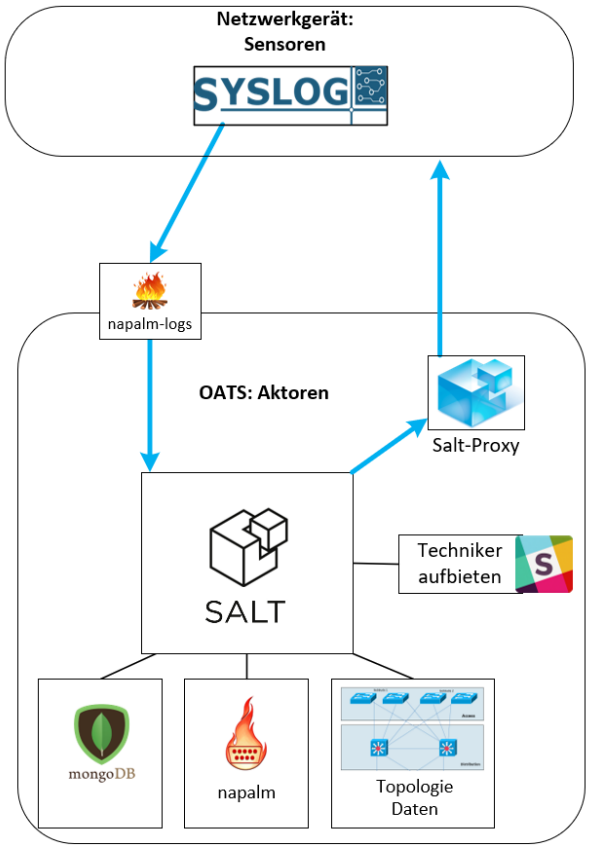
Das Ergebnis ist OATS: ein System mit welchem automatisches Troubleshooting in einem Netzwerk möglich wird. Dafür wurde während der Studienarbeit das Gerüst entwickelt, welches all die oben beschriebenen Tools sinnvoll zusammenführt.
Bis zum Ende der SA sind zwei Troubleshooting Workflows entwickelt worden die je einen Fehler im Netzwerk zuverlässig erkennen und beheben.

Ausblick

OATS ist noch nicht so ausgereift, wie das Projektteam es gerne hätte. Deswegen wird OATS auch in einer weiterführenden Bachelorarbeit weiterentwickelt werden.
Mögliche Themen wären:

- Das Entwickeln von weiteren Workflows
- Das Aggregieren/Korrelieren von Events um die Informationsbeschaffung zu vereinfachen
- Das Einsetzen von OATS in einem produktiven Umfeld
- Refactoring von OATS, damit es möglich ist das System komplett ohne Programmierkenntnisse zu bedienen und zu erweitern

Tool	Zweck
Syslog	Sensor für Netzwerkprobleme, löst Events aus
Napalm-logs	Daten aus Syslog-Events extrahieren
Salt	Auf Events mit Troubleshooting Workflows reagieren
Salt-proxy	Proxy für Netzwerkgeräte
Slack	Chat-Ops
Napalm	Library für Netzwerkzugriffe
mongoDB	Datenspeicherung
Topologie Daten	Daten über das zugrundeliegende Netzwerk



Inhaltsverzeichnis

I. Technischer Bericht 8
1. Ausgangslage & Problembeschreibung 9
2. Lösungskonzept 10
2.1 Erfassung 10
2.2 Identifizierung 11
2.3 Klassifizierung 11
2.4 Behebung 11
2.5 Überprüfung 11
2.6 Reporting/Logging 11
3. Umsetzung 12
3.1 Syslog 12
3.2 Napalm-logs 13
3.3 Salt 13
3.4 Napalm 13
3.5 MongoDB 13
3.6 Slack 13
3.7 Salt-Proxy 14
4. Ergebnisdiskussion 14
5. Ausblick 14
6. Danksagung 14
II. Software-Engineering Dokumente 15
1. Domainanalyse 16
Änderungsgeschichte 16
1.1 Einführung 17
1.1.1 Zweck 17
1.1.2 Gültigkeitsbereich 17
1.2 Technologieanalyse 17
1.2.1 Ping 17
Options 17
Beispiel Ping 18
Format 18
Echo Reply und Request 19
Payload 19

Vor- und Nachteile 19
1.2.2 Traceroute 20
Implementation 20
Beispiel Traceroute 21
Optionen 21
Vor- und Nachteile 21
1.2.3 SNMP – Simple Network Management Protocol 21
Funktionsweise 22
Paketaufbau 22
Vor- und Nachteile 23
1.2.4 NETCONF – Network Configuration Protocol 23
Protokoll-Layer 24
Content-Layer 24
Operations-Layer 24
Messages 24
YANG 24
Vor – und Nachteile 28
1.2.5 Syslog 28
Message-Format 28
Konfiguration 29
Vor- und Nachteile 29
1.2.6 Netflow 29
Anwendung 30
Versionen und Alternativen 31
Vor- und Nachteile 31
1.2.7 Netzwerk-Sniffing 31
Funktionsweise 31
Vor- und Nachteile 31
1.2.8 IP SLA - Internet protocol service level agreement 32
Anwendung 32
Vor- und Nachteile 33
1.2.9 SSH – Secure Shell 33
1.3 Problemanalyse 33
1.3.1 OSI-Modell 33
Layer 1: Physical Layer 33
Layer 2: Data Link Layer 34
Layer 3: Network Layer 34
Layer 4: Transport Layer 34

1.3.2 Probleme 35
Layer 1 Probleme 35
Layer 2 Probleme 35
Layer 3 Probleme 37
Layer 4 Probleme 40
Applikation Layer Probleme 40
1.4 Deviceanalyse 42
1.4.1 Geräte in einem Netzwerk 42
Router 42
Switch 42
Firewall 42
1.4.2 Probleme durch Geräte 42
2. Anforderungen 43
Änderungsgeschichte 43
2.1 Zweck 44
2.1.1 Beschreibung 44
2.1.2 Gültigkeitsbereich 44
2.1.3 Übersicht 44
2.2 Systemumfang 45
2.2.1 Stakeholder 45
2.2.2 Systemfunktionalität 45
2.2.3 Systemumfeld 45
2.2.4 Architekturbeschreibungen 46
Erfassung 46
Identifizierung 46
Klassifizierung 46
Behebung 46
Überprüfung 46
Reporting/Logging 46
2.2.5 Abhängigkeiten 48
Netzwerktopologie 48
2.3 Use Cases 49
2.3.1 Aktoren 49
2.3.2 Use Case Diagramm 49
2.3.3 Use Cases im Brief Format 49
2.4 Anforderungen an das System 52
2.5 Nicht funktionale Anforderungen 54

2.5.1 Modifizierbarkeit 54
2.5.2 Interoperabilität 54
2.5.3 Richtigkeit 54
3. Architektur 55
Änderungsgeschichte 55
3.1 Einführung 56
3.1.1 Zweck 56
3.1.2 Gültigkeitsbereich 57
3.2 Toolanalyse 58
3.2.1 Sensoren 58
Syslog 58
NETCONF 58
NMS –Network Monitoring Systems 58
3.2.2 Aktoren 59
NAPALM 59
NAPALM-Logs 61
Pyangbind 63
3.2.3 Core Tools 63
Ansible 63
Stackstorm 66
Salt 69
Fazit 71
3.2.4 Logging 73
MongoDB 73
PostgreSQL 73
MariaDB 73
Verwendung 73
3.2.5 ZeroMQ 74
3.2.6 Slack 74
3.3 Architektur: Konzept 75
3.3.1 Erfassung 75
3.3.2 Identifizierung 76
3.3.3 Klassifizierung 76
3.3.4 Behebung 77
3.3.5 Überprüfung 77
3.3.6 Reporting/Logging 77
3.4 Architektur: Umsetzung 78

3.4.1 Erfassung 79
3.4.2 Identifizierung 79
3.4.3 Klassifizierung 79
3.4.4 Behebung 79
3.4.5 Überprüfung 79
3.4.6 Reporting/Logging 79
3.5 Abläufe 80
3.6 Logische Architektur 81
3.6.1 Napalm-logs 81
3.6.2 Salt 82
3.6.3 Oatsdbhelpers 83
3.6.4 DB Access 83
3.6.5 Schnittstellen 84
System <> Techniker 84
Innerhalb des Systems 84
Netzwerkgerät-OS <> napalm-logs 84
Salt <> Netzwerkgerät-OS 84
3.7 Deployment 85
Netzwerkgeräte 85
Server 85
3.8 Datenspeicherung 86
3.8.1 Device 86
3.8.2 Connections 86
3.8.3 Cases 87
3.8.4 Technician 87
3.9 Prozesse und Threads 87
3.9.1 oatsclient.py 87
3.9.2 tshoot.py 87
4. Anleitungen 88
Änderungsgeschichte 88
4.1 Einführung 89
4.1.1 Zweck 89
4.1.2 Gültigkeitsbereich 89
4.2 Installationsanleitung 90
4.2.1 Python 90
4.2.2 Pip 90
4.2.3 Salt 90

4.2.4 Napalm 90
4.2.5 Napalm-logs 90
4.2.6 OATS 90
4.2.7 MongoDB 91
Installation 91
Oatsdbhelpers 91
4.3 Konfigurationsanleitung 91
4.3.1 Napalm-Logs 91
4.3.2 Salt-Proxies 92
4.3.3 Netzwerkgeräte 92
4.4 Benutzungsanleitung 93
4.4.1 Salt Environment 93
Salt Master 93
Salt Minion 93
Salt Napalm-Proxies 93
4.4.2 Napalm-Logs 94
Listener 94
Client 94
4.4.3 MongoDB 94
4.4.4 Testing der Workflows 95
4.5 Development 95
4.5.1 Syslog 95
4.5.2 Napalm-Logs 95
4.5.3 Oatsclient 96
Konstruktion der Salt-Events 96
Direktes Weiterleiten der Events 97
Aggregieren von Events 97
Überlegungen für die Zukunft 98
4.5.4 Salt 98
Salt Master File 98
Salt Reactor 98
Salt-Runners / Workflows 99
Überlegungen für die Zukunft 100
5. Systemtests 101
Änderungsgeschichte 101
5.1 Voraussetzungen 102
5.2 Test Spezifikation 102

5.3 Testprotokoll 105
III. Glossar 110
1. Begriffe 110
2. Abkürzungen 111
IV. Bibliographie 113
1. Quellenverzeichnis 113
2. Abbildungsverzeichnis 115
V. Anhänge 116
Anhang A: Aufgabenstellung 117
Anhang B: Projektplan 118
Anhang C: Zeitabrechnung 129
Anhang D: Technische Risiken 131

I. Technischer Bericht

Projekt: Automatisiertes Troubleshooting

Raphael Jöhl
Nico Vinzens

1. Ausgangslage & Problembeschreibung

Viele Arbeitsschritte bei Fehlerbehebungen könnten eigentlich automatisiert werden. So Arbeitet eine nette Stimme bei der Support Hotline aller grossen Telekom Konzerne zuerst die Checkliste ab. Das gleiche hat auch in den anderen IT Bereichen seine Gültigkeit. So sammelt ein Netzwerk Engineer bei einem Support Ticket zuerst meistens nur Informationen. Wo befindet sich das Gerät? In welchem Subnetz ist es? Kann ich es via ICMP erreichen? Solche Arbeiten eignen sich sehr gut zum Automatisieren. So können bei einem Fehlerfall die üblichen Tasks bereits abgearbeitet werden. Sind genügend Informationen vorhanden können automatisch Lösungsversuche gestartet werden oder das Problem temporär umgangen werden, so dass der Techniker am nächsten Tag ausgeschlafen die Bereinigung abschliessen kann.

Aufgabe: Entwickeln eines Systems, welches dank dem Konzept von «Event Driven Automation» das Troubleshooting automatisieren kann. Über Sensoren sollen Workflows angestossen werden, welche wiederum sogenannte Aktoren ansteuern. Es muss nicht eine komplette Eigenentwicklung geben, da es viele Komponenten bereits als Open Source Lösungen gibt. Im gesunden Rahmen sollen diese Projekte durch Integrationen und Anpassungen zu einer Gesamtlösung verschmelzen.

2. Lösungskonzept

Während der Inception Phase wurde folgender Ablaufplan entworfen, um eine erste Vorstellung zu erhalten, wie ein Troubleshooting-Prozess ablaufen sollte:

Abbildung 1: Troubleshooting Workflow

2.1 Erfassung

Das System wird durch ein Event gestartet, welches den Troubleshooting-Prozess in Gang setzt. Events könnten z.B. durch den Kunden (Ticket, Email, Telefon), oder direkt durch ein Monitoring System (z.B. Syslog) ausgelöst werden.

2.2 Identifizierung

Das System identifiziert, von welchen Geräten, das Problem ausgeht und um welche Art von Problem es sich handelt.

2.3 Klassifizierung

Das System teilt die Probleme in 2 unterschiedliche Oberklassen ein:

1. Durch das System selber lösbar
2. Nicht durch das System lösbar

Ist z.B. das betroffene Gerät wegen eines Kabelbruchs nicht mehr am Netz, so soll das System direkt einen Techniker benachrichtigen, der das Problem lösen soll.

2.4 Behebung

Ist das System in der Lage das Problem zu beheben, so wird ein Skript ausgeführt, welches gängige Netzwerk-Troubleshooting Schritte ausführt.
Falls ein Techniker nötig ist, wird ihm ein Bericht zum Problem übermittelt und er muss das Problem manuell lösen.

2.5 Überprüfung

Das System kontrolliert, ob das Problem behoben wurde. Falls es durch das System selbst nicht behoben werden konnte, muss ein Techniker benachrichtigt werden, der das Problem manuell löst.

2.6 Reporting/Logging

Das System erstellt einen Logeintrag, in welchem alle Details des Troubleshooting-Prozesses einsehbar sind. Der Eintrag wird bereits mit der Erfassung des Problems erstellt und dann laufend mit den gesammelten Informationen aus dem Troubleshooting Prozess erweitert.

3. Umsetzung

Mit diesem zugrundeliegenden Konzept wurden Tools identifiziert, welche diese Aufgaben erfüllen können und im weiteren Verlauf der SA zu einem sinnvollen Ganzen zusammengeschlossen:

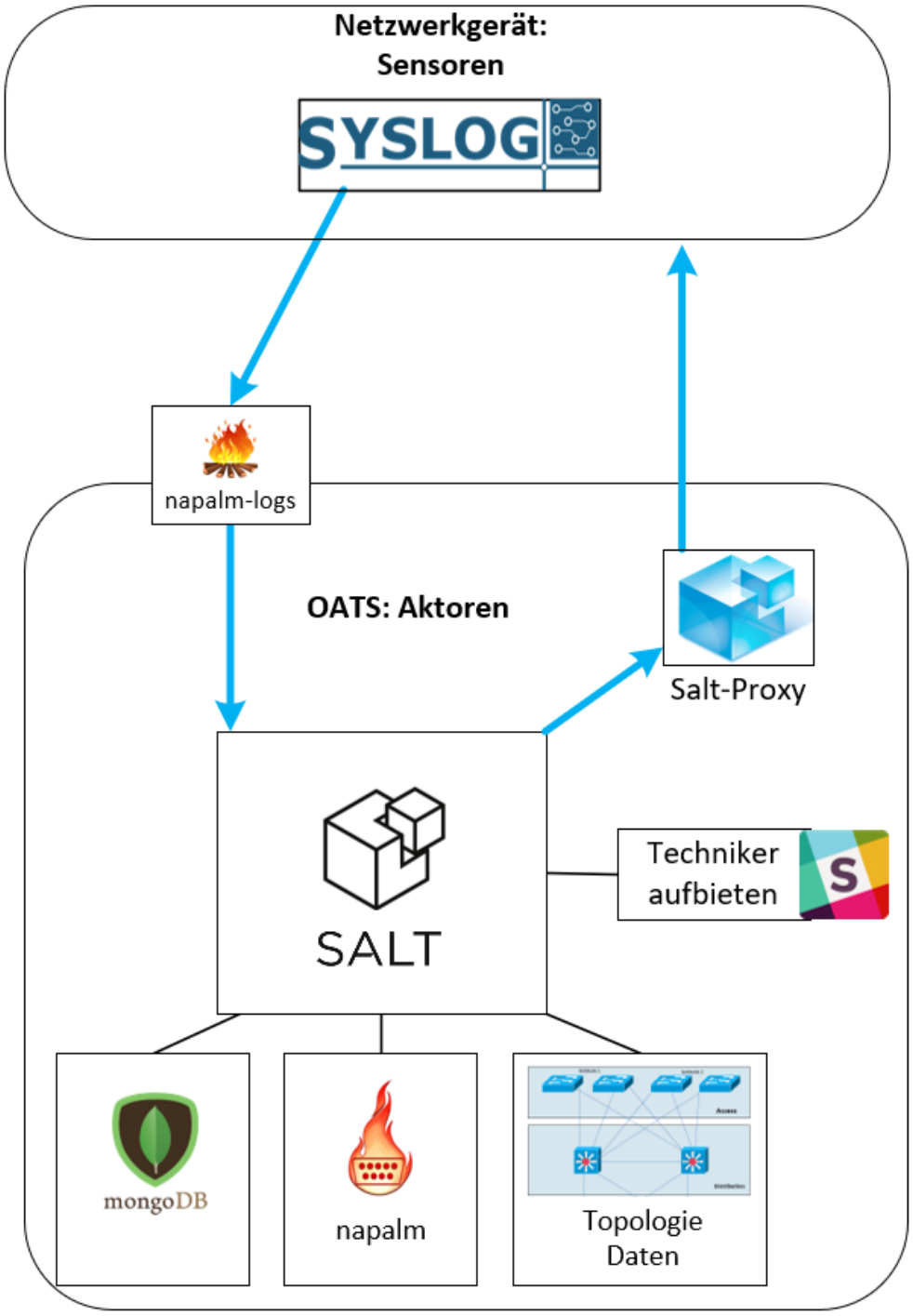


Abbildung 2: Architektur (Barroso, Welcome to the NAPALM Documentation, 2017) (Ulinic, 2017) (MongoDB, 2017) (Slack, 2017) (Saltstack, 2017) (wikibooks, 2017)

3.1 Syslog

Syslog ist ein Protokoll zum zentralen Sammeln von Informationen in einem Netzwerkumfeld und läuft in der Regel direkt auf Netzwerkgeräten. Auf jedem Netzwerkgerät muss genau definiert werden, welche Meldungen geschickt werden und welche nicht. So kann z.B. eingestellt werden,

dass wenn ein Interface auf einem Router runterfährt, automatisch eine Meldung von diesem Router an eine zentrale Stelle geschickt wird.
In diesem Projekt wurde das entwickelte OATS als zentrale Sammelstelle eingestellt. Somit schicken alle Netzwerkgeräte bei vordefinierten Ereignissen Meldungen an OATS.

3.2 Napalm-logs

Da Syslog als Sensor im Netz eingesetzt wird, ist eine offene Frage, wie die plaintext Nachrichten von Syslog in eine im Code nutzbare Form gebracht werden können. Diese Lücke füllt napalm-logs.
Napalm-logs ist eine Open Source Software, die als Daemon läuft und auf Syslog Meldungen horcht und diese in eine herstellerunabhängige Form bringt. Anders als napalm wird napalm-logs nur benutzt um Meldungen zu empfangen.
Die Meldungen werden entweder direkt durch TCP/UDP empfangen oder im Zusammenhang mit einem Messaging-System wie ZeroMQ entgegengenommen und verschickt.
Napalm-logs läuft auf dem gleichen Server wie OATS und liefert Daten, welche aus den Syslog Meldungen extrahiert wurden, sodass diese in OATS weiterverarbeitet werden können.

3.3 Salt

Salt ist eine komplexe Open Source Software (Saltstack ist die Enterprise Version) mit dem Ziel eine ereignisbasierte Netzwerkautomation zu erreichen. OATS bildet eine Art Wrapper um Salt und bedient sich der vielfältigen Funktionen zum Sammeln von Daten und Manipulieren von Netzwerkgeräten.
Salt (bzw. ein Salt-master) ist dafür verantwortlich als Reaktion auf ein von Syslog ausgelöstes und von napalm-logs verarbeitetes Event einen Troubleshooting Workflow auszuführen. Innerhalb dieses Workflows werden per napalm (siehe 3.4) Informationen gesammelt und Befehle auf den Netzwerkgeräten ausgeführt.

3.4 Napalm

Napalm (Network Automation and Programmability Abstraction Layer with Multivendor support) ist eine Open Source Python-Library die Funktionen implementiert, welche es einem erlauben mit unterschiedlichen Netzwerkgeräten zu interagieren. So ermöglicht napalm den Verbindungsaufbau zu Geräten, die Manipulation deren Konfiguration und das Sammeln von Informationen.
Napalm ist direkt in Salt integriert, was die Ausführung der Funktionen einfach macht.

3.5 MongoDB

MongoDB ist eine Verteilte Datenbank, welche hohe Verfügbarkeit, horizontale Skalierung und Geographische Verteilung bietet. MongoDB ist zudem frei verfügbar und Open Source.
Im Rahmen von OATS wird MongoDB verwendet um die Netzwerktopologie abzulegen, welche in den Troubleshooting Workflows benötigt wird. Zudem wird dort für jeden von OATs behandelten Problemfall ein Log erstellt.

3.6 Slack

Der Name Slack ist ein Akronym für “Searchable Log of All Conversation and Knowledge” und die Software ist im Grunde eine Chat App.
OATS verwendet Slack um Techniker zu benachrichtigen, wenn ein Problem nicht behoben werden konnte. Dabei Ist für den Techniker ersichtlich, welche Schritte das System bereits unternommen hat und er kann das Problem eingrenzen.

3.7 Salt-Proxy

Salt-proxies sind eine Komponente von Salt und sind einem Salt-master untergeordnet. Für jedes Netzwerkgerät, welches durch Salt angesteuert werden soll, existiert ein solcher Proxy. Dieser Proxy weiss genau mit welchen Protokollen er mit seinem Netzwerkgerät kommunizieren kann. So kann der Salt-master indirekt mit allen Netzwerkgeräten kommunizieren, ohne dass er für jedes Gerät wissen muss, über welches Protokoll dies möglich ist.

4. Ergebnisdiskussion

Als Resultat dieser Arbeit steht ein System, mit dem aufgezeigt werden konnte, dass automatisches Troubleshooting im Netzwerkumfeld nur mit dem Einsatz von Open Source Technologien möglich ist. Im Rahmen der SA wurden 2 Troubleshooting Workflows umgesetzt, die je einen Netzwerkfehler erkennen und zuverlässig beheben. Die beiden Workflows betreffen folgende Fehler:

- **Interface down:** wenn ein Interface unerwartet runterfährt. Ist nur das Interface runtergefahren, so wird der Workflow das erkennen und das Interface wieder hochfahren. Ist aber ein gesamtes Gerät runtergefahren, so ist OATS machtlos und es wird ein Techniker per Slack benachrichtigt.
- **OSPF neighbor down:** wenn ein OSPF-Prozess auf einem Router einfriert. Dies löst von den umliegenden Geräten mehrere gleichartige Syslog Meldungen aus. OATS erkennt dies und startet als Reaktion darauf den OSPF-Prozess auf dem betroffenen Gerät neu.

5. Ausblick

Da OATS in einer mehrheitlich virtuellen Testumgebung entworfen wurde, wäre OATS einem letzten Test zu unterziehen: der Einsatz in einem produktiven Umfeld mit echten Netzwerkgeräten und echtem Netzwerkverkehr.

Aus der Arbeit wurden einige interessante Erkenntnisse gezogen, bei denen es sich lohnt diese im Rahmen einer Bachelorarbeit zu untersuchen. So befindet sich z.B. napalm-logs noch in den Kinderschuhen und es wäre sehr interessant dieses Tool zu erweitern damit der Troubleshooting Prozess vereinfacht werden kann.
Die Erweiterbarkeit wurde als wichtigste nichtfunktionale Anforderung deklariert. Es wurde auch eine Anleitung zum Erweitern von OATS geliefert, diese setzt aber z.T. Programmierkenntnisse voraus. Ideal wäre es, wenn das Erweitern ohne Programmierkenntnisse möglich wäre. Auch dies wäre ein mögliches Thema für eine weiterführende Arbeit.

6. Danksagung

Wir möchten uns bei Prof. Stettler für die Chance bedanken dieses Projekt durchführen zu dürfen. Diese Arbeit hat uns die Welt der Netzwerkautomation, die uns zu Beginn des Semesters völlig unbekannt war, auf eine sehr lehrreiche Art und Weise näher gebracht. Wir sind uns sicher, dass das während der Arbeit gewonnene Wissen noch lange nützlich sein wird.
Weiter möchten wir uns bei Urs Baumann vom INS bedanken, der sich immer Zeit genommen hat, unsere Fragen zu beantworten. Auch für das Aufsetzen der Testumgebung, die während der gesamten Projektdauer immer verfügbar war, sind wir ihm sehr dankbar.

Der letzte Dank gebührt dem „Network to Code“-Slack Workspace (Chat für Entwickler) und seinen aktiven Mitgliedern. In diesem Workspace tummeln sich viele der Entwickler der Open Source Tools, welche im Rahmen dieser Studienarbeit eingesetzt wurden.
Im speziellem möchten wir hier Mircea Ulinic von Cloudflare danken, der uns bei Fragen zu den Tools Salt und napalm-logs mehr als nur einmal aus der Klemme geholfen hat.

II. Software-Engineering Dokumente

Projekt: Automatisiertes Troubleshooting

Raphael Jöhl
Nico Vinzens

1. Domainanalyse

Änderungsgeschichte

Datum	Version	Änderung	Autor
19.09.2017	1.0	Erstellen des Dokuments	nv
04.10.2017	1.1	Technologieanalyse	nv
05.10.2017	1.2	Technologieanalyse	rj
06.10.2017	1.3	Technologieanalyse, Problemanalyse, Deviceanalyse	rj
07.10.2017	1.4	Problemanalyse	rj
15.12.2017	2.0	Korrektur vor Abgabe	nv

1.1 Einführung

1.1.1 Zweck

Dieses Dokument wurde erstellt, um einen Überblick über die Domain, in der im Rahmen dieses Projekts gearbeitet wird, zu erlangen.

Das Automatisierte Troubleshooting in einem Netzwerk ist ein sehr komplexes Thema, um präzise und sinnvolle Anforderungen an unser System zu erstellen und definieren, analysieren wir in diesem Dokument die Technologien die uns zur Verfügung stehen, Probleme die in einem Netzwerk auftreten und Geräte mit welchen wir in einem Netzwerk arbeiten.
Als Erstes kommt eine Technologieanalyse in der beschrieben wird, mit welchen Tools Informationen über Geräte und das Netzwerk beschafft werden können.
In der Problemanalyse folgt eine Auflistung an möglicher Probleme die in einem System auftreten können und wie diese detektiert werden können. Als Einführung folgt dazu zuerst eine kurze Erläuterung über die vier unteren OSI Layer, auf welche wir uns in den möglichen Problemen konzentrieren möchten.
Danach werden in der Deviceanalyse die möglichen Geräte, mit welchen wir in einem Netzwerk arbeiten, beschrieben und wieso hier Probleme auftreten.

1.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung vom automatisierten Troubleshooting und während der nachfolgenden Bachelorarbeit.

1.2 Technologieanalyse

In diesem Abschnitt werden verschiedene Technologien beschrieben und analysiert, die im Netzwerkumfeld benutzt werden können, um Informationen über den Status des Netzwerks zu erhalten.

1.2.1 Ping

Ping ist ein Protokoll welches es uns erlaubt innerhalb eines IP Netzwerks zu überprüfen ob ein Host verfügbar ist. Ein ping misst die Round Trip Time zwischen zwei Hosts. Ping verwendet Internet Control Message Protocol (ICMP/ICMP6) Echo Request Pakete um beim Ziel eine Anfrage zu platzieren, auf die dieser mit einer ICMP Echo Response antwortet.
(IETF, 2017)

Options

Die Optionen von ping unterscheiden sich je nach Implementation innerhalb des Systems. Die Optionen verfügbar unter Windows 10 sind:
Options:

Option	Beschreibung
-t	Pingt das Ziel solange, bis es gestoppt wird. Um eine Statistik anzuzeigen und weiterzufahren kann man “Control-Break” eingeben, um den ping zu beenden kann man „Control-C“ eingeben.
-a	Wandelt Adressen in Hostnamen um
-n <Anzahl>	Die Anzahl der zu übertragenden echo requests
-l <Grösse>	Die Grösse des Payloads eines Pakets
-f	Setzt das „nicht Fragmentieren“ Flag, nur in IPv4

-i	Spezifiziert die Time to Live
-r <Anzahl>	Zeichnet die Route für die spezifizierte Anzahl Sprünge auf, nur in IPv4
-s <Anzahl>	Gibt die Zeit an für die erreichte Anzahl Sprünge, nur in IPv4
-j <host-list>	Loose Route entlang der angegebenen Hosts, nur in IPv4
-k <host-list>	Strenge Route entlang der angegebenen Hosts, nur in IPv4
-w <timeout>	Wie lange in Millisekunden auf jede Antwort gewartet werden muss
-S <src-addr>	Falls eine alternative Quelle verwendet werden soll
-4	Erzwingt IPv4
-6	Erzwingt IPv6

Beispiel Ping

Hier wurde eine Web Adresse fünf Mal „angepingt“.

```
U:\>ping -n 5 hsr.ch

Pinging hsr.ch [152.96.12.22] with 32 bytes of data:
Reply from 152.96.12.22: bytes=32 time<1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127
Reply from 152.96.12.22: bytes=32 time=1ms TTL=127

Ping statistics for 152.96.12.22:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

Abbildung 3: 3: ping -n5 auf hsr.ch

Format

IP Datagram				
	Bits 0–7	**Bits 8–15**	**Bits 16–23**	**Bits 24–31**
IPv4 Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Identifier		Sequence Number	
ICMP Payload (*optional*)	Payload Data			

IP Datagram						
	Bits 0–3	**Bits 4–7**	**Bits 8–11**	**Bits 12–15**	**Bits 16–23**	**Bits 24–31**
IPv6 Header (40 bytes)	Version	Traffic Class		Flow Label		
	Payload Length				Next Header	Hop Limit
	Source Address					
	Destination Address					
ICMP6 Header (8 bytes)	Type of message		Code		Checksum	
	Header Data					
ICMP6 Payload (*optional*)	Payload Data					

In einem regulären IPv4 Header ist „Protocol“ auf 1 gesetzt für ICMP und „Type of Service“ auf 0.
In einem regulären IPv6 Header ist das Feld „Next Header“ auf 58 gesetzt.

Der ICMP Header beinhaltet das 8 Bit lange „Type of message“ Feld, das 8 Bit lange „Code“ Feld, und das „Checksum“ Feld ist 16 Bit lang und gibt an ob die Summe der Einsen innerhalb der ICMP Nachricht korrekt ist.
Das Header Data Feld welches hier entweder Echo requests oder replies sind, bestehen dadurch aus dem 16 Bit „Identifier“ und der 16 Bit „Sequence Number“.
Der ICMP Payload kann selber definiert werden, jedoch muss der Payload mit den Headern kleiner sein als die Maximum Transmission Unit oder man riskiert eine Fragmentierung.

Echo Reply und Request

Die oben beschriebenen Identifier und Sequenz Nummer Felder dienen dazu bei einem Echo Request die dazu gehörenden Antwort (Echo Reply) zu finden. Die Details des Identifiers sind je nach Implementation unterschiedlich.

Ein Echo Reply ist die Antwort auf einen Echo Request, sie ist für alle Hosts zwingend und muss den genauen erhaltenen Payload angeben.

Payload

Grundsätzlich wird Payload bei Angabe eines solchen mit ASCII Zeichen gefüllt. Der Payload kann ausserdem separat einen Timestamp und eine Sequenznummer enthalten. Dies ermöglicht es einem Ping die Round Trip Zeit stateless zu ermitteln ohne die Zeit der Übertragung jedes Pakets zu ermitteln.

Vor- und Nachteile

Vorteile:

- Einfach und simples Tool um die Verbindung zu einem Gerät zu überprüfen
- Diverse Optionen erlauben es uns genauere Informationen zu erhalten

Nachteile:

- Begrenzte Anwendungsmöglichkeiten

1.2.2 Traceroute

Traceroute ist ein Netzwerk-Diagnose-Tool welches uns erlaubt die Route und Verzögerungen von Paketen über ein IP Netzwerk zu ermitteln. Der Verlauf der Route wird zusammen mit der Round Trip Time der Pakete welche man von jedem nachfolgendem Host erhalten hat gespeichert. Die Summe der durchschnittlichen Zeit zwischen jedem Schritt ergibt die Zeit welche insgesamt benötigt wurde um die Verbindung herzustellen. Traceroute sendet jeweils drei Pakete, wenn diese mehr als zwei Mal verloren gingen ist die Verbindung verloren und die Route kann nicht evaluiert werden. Anders als bei Ping werden jeweils die Zeit für die einzelnen Schritte aufgeschrieben und nicht nur die insgesamt benötigte Zeit.
(Microsoft, 2017), (Linux/Unix, 2017)

Implementation

Die Implementation von traceroute unterscheidet sich bei Unix basierten Systemen und Windows Systemen. Unter Unix werden per default eine Sequenz von UDP oder TCP Paketen gesendet mit der Option ICMP Echo Request. Unter Windows wird traceroute über ICMP Echo Requests gesendet.

Um herauszufinden wie viele Schritte bis zum Ziel nötig sind verwendet traceroute den Time to Live Wert. Erhält ein Router ein Paket reduziert er die TTL um eins, falls die TTL auf null ist, verwirft er das Paket und sendet ein ICMP Time Exceeded Paket zurück. Erhält der Host von welchem der traceroute gestartet wurde eine solche Nachricht erhöht er die TTL des Pakets um eins um den nächsten Schritt des Pfades herauszufinden. Somit kann der Sender Schritt für Schritt eine Liste mit allen Routern und deren Verbindungszeit erstellen. Dies geht so lange weiter bis das Paket das Ziel erreicht hat. Das eigentliche Ziel des traceroutes sendet dann eine ICMP Echo Reply Nachricht und signalisiert dass das Ziel erreicht wurde.

Der Sender hat jeweils eine Zeitgrenze wie lange er auf die Reply eines Schrittes wartet. Wird diese Überschritten zeigt er in der Liste einen Stern. Da das Internet Protocol nicht erzwingt dass die Pakete immer die gleiche Route verwenden, kann es sein, dass die Liste mit den Hosts auf dem Weg des traceroutes unterschiedlich ist wie die schlussendlich überquerten Hosts.

In einem Netzwerk welches sowohl Windows als auch Unix System hat, müssen in einer eventuell vorhandenen Firewall mehrere Protokolle erlaubt sein, ansonsten funktioniert traceroute nicht.

Es gibt unterschiedliche Arten von traceroute, so gibt es zum Beispiel auch tcptraceroute und Layer 4 traceroute. Ein weiteres Werkzeug ist Pathping welche die Funktionalitäten von Ping und traceroute verbindet.

Beispiel Traceroute

```
U:\>tracert google.com

Tracing route to google.com [172.217.19.14]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  router-192.hsr.ch [152.96.192.1]
  2     3 ms     2 ms     1 ms  hsrpr1.hsr.ch [152.96.2.49]
  3     1 ms    22 ms     3 ms  152.96.2.33
  4     1 ms     1 ms     1 ms  swiAG1-10GE-0-0-1-1.switch.ch [130.59.37.13]
  5     1 ms     1 ms     1 ms  swiEZ1-10GE-2-1.switch.ch [130.59.38.2]
  6     2 ms     2 ms     2 ms  swiEZ3-B3.switch.ch [130.59.36.34]
  7     2 ms     2 ms     2 ms  swiZH1-100GE-0-1-0-0.switch.ch [130.59.38.110]
  8     2 ms     2 ms     2 ms  72.14.195.4
  9     2 ms     2 ms     2 ms  209.85.244.127
 10     2 ms     2 ms     2 ms  zrh04s08-in-f14.1e100.net [172.217.19.14]

Trace complete.
```

Abbildung 4: tracert auf google.com

In Abbildung 2 sieht man ein tracert-Beispiel für google.com unter Windows. Rechts sieht man die IP-adressen und falls vorhanden die Namen der traversierten Router.

Optionen

Nachfolgend die zu Verfügung stehenden Optionen für traceroute unter Unix Systemen.

Option	Beschreibung
-4, -6	Zwingt traceroute dazu IPv4 oder IPv6 zu verwenden, per default wird das Protokol automatisch gewählt
-I	Benutzte ICMP ECHO für die Pakete, wie tracert unter Windows
-T	Benutzte TCP SYN für die Pakete
-U	Benutzte UDP Datagramme für die Pakete, dies ist der Default unter Unix
-F	Setzt das „Nicht Fragmentieren“ Bit
-ffirst_<ttl>	Spezifiziert mit welcher TTL gestartet wird, Default ist 1
-mmax_<ttl>	Spezifizert die maximale Nummer der Schritte, Default ist 30
-iinterface	Spezifiziert über welches Interface traceroute gesendet werden soll, per Default wird dieses nach der Routing Table gewählt
-Nsqueries	Spezifiziert die Anzahl der Pakete die gleichzeitig gesendet werden

Vor- und Nachteile

Vorteile:

- Erlaubt es Routen Informationen nachzuvollziehen und zu überprüfen
- Verwendete Technologien werden von allen Geräten unterstützt

Nachteile:

- Begrenzte Anwendungsmöglichkeiten

1.2.3 SNMP – Simple Network Management Protocol

Ein SNMP-Management System beinhaltet (IETF, 2017):

- mehrere Knoten, die alle über eine SNMP-Instanz verfügen, welche einen *command responder* und *notification originator applications* beinhalten (auch Agenten genannt);

- mindestens eine SNMP-Instanz mit einem *command generator* und/oder *notification receiver applications* (auch Manager genannt) und
- ein Management-Protokoll, um Informationen zwischen den SNMP-Entitäten auszutauschen.

SNMP-Manager überwachen und steuern Geräte, auf welchen SNMP-Agenten aufgesetzt sind. Solche Geräte sind typischerweise Server, Router etc.

Funktionsweise

Die Agenten, welche direkt auf den zu überwachenden Geräten laufen, sind in der Lage den Zustand eines Netzwerkgerätes zu erfassen und auch selbst Einstellungen vorzunehmen oder Aktionen auszulösen. Die Agenten kommunizieren mit dem Manager über ein Netzwerk, dazu gibt es sechs verschiedene Datenpakete die gesendet werden können:

Datenpaket	Beschreibung
GET-REQUEST	Zum Anfordern eines Management-Datensatzes
GETNEXT-REQUEST	Um den nachfolgenden Datensatz abzurufen (für Tabellen)
GETBULK	Um eine angegebene Anzahl an Datensätzen gleichzeitig abzurufen
SET-REQUEST	Um einen oder mehrere Datensätze eines Geräts zu verändern
GET-RESPONSE	Antwort auf eines der vorherigen Pakete
TRAP	Nachricht eines Agenten an den Manager, dass ein Ereignis eingetreten ist. Kann benutzt werden um eine Fehlfunktion oder das nicht erfolgreiche Ausführen eines SET-REQUESTS zu melden.

GET-Pakete werden vom Manager an einen Agenten geschickt, um Informationen über das jeweilige Gerät anzufordern.
Mit einem SET-REQUEST-Paket kann ein Manager Werte bei einem Agenten verändern. Die erfolgreiche Änderung bestätigt der Agent mit einem GET-RESPONSE-Paket.
Stellt ein Agent bei der Überwachung eines Systems einen Fehler fest, so kann er dies dem Manager mithilfe eines Trap-Pakets unaufgefordert melden. Trap-Pakete werden vom Manager nicht bestätigt.
Zum Übermitteln der Pakete wird das Transportprotokoll UDP verwendet. Der Agent empfängt SNMP-Requests auf Port 161, während der Manager Trap-Pakete auf Port 162 empfängt.
Die Datenbasis des jeweiligen Geräts ist in der MIB (Management Information Base) hinterlegt.

Paketaufbau

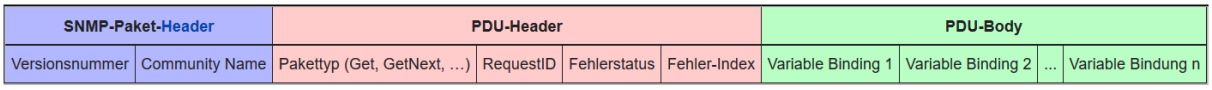


Abbildung 5: SNMP-Paket Aufbau (Wikipedia, 2017)

SNMP-Paket-Header

Versionsnummer: SNMPv1, SNMPv2 oder SNMPv3
Community Name: Zum Erteilen von Rechten werden Communities zugewiesen (z.B. private & public)

PDU-Header (Nicht-Trap-Pakete)

Pakettyp: z.B. GET-REQUEST, GETBULK etc.
Request-ID: Um Antwortpakete den vorausgegangenen Anfragen zuordnen zu können
Fehlerstatus: wird bei Antwortpaketen verwendet um mitzuteilen, weshalb eine Anfrage nicht bearbeitet werden konnte.
Fehlerindex: gibt im Fehlerfall an, beim wievielten Datensatz der Fehler aufgetreten ist

PDU-Header (Trap-Pakete)

PDU-Header					
Pakettyp (Trap)	OID des Gerätes, das den Trap generiert hat	IP-Adresse des Absenders	allgemeine TrapID	firmenspezifische TrapID	Zeitpunkt des Auftretens des Trap-Ereignisses

Abbildung 6: Trap-PDU-Header

OID des Geräts: herstellerspezifische OID (Object Identifier), gibt an um was für ein Gerät es sich handelt

Absender-IP: IP-Adresse des Absenders

Allgemeine Trap-ID:

Bezeichnung	Trap-ID
coldStart	**0**
warmStart	**1**
linkDown	**2**
linkUp	**3**
authenticationFailure	**4**
egpNeighborLoss	**5**
enterpriseSpecific	**6**

Bei ID 6 (enterpriseSpecific), wird die ID des firmenspezifischen Traps im Feld danach übertragen. Um zeitliches Ordnen zu ermöglichen, wird zusätzlich noch die Zeitangabe übermittelt, bei der das Ereignis ausgelöst wurde.

PDU-Body

Beinhaltet die eigentlichen Werte. Jeder Wert wird in einer sogenannten Variable Binding übertragen. Zu solch einer Variable Binding gehören jeweils dessen OID und der Wert selber.

Vor- und Nachteile

Vorteile:

- SNMP ist ein offener Standard, d.h. herstellerunabhängig
- Die meisten Netzwerkgeräte unterstützen SNMP

Nachteile:

- Begrenzte Funktionalität: Cisco-Geräte unterstützen zum Beispiel nur ca. 90 verschiedene Traps (ggü. 6500 bei syslog) (Cisco, 2017)

1.2.4 NETCONF – Network Configuration Protocol

Das NETCONF-Protokoll bietet einen Mechanismus um Netzwerkgeräte zu managen, Konfigurationsdaten abzurufen und um neue Konfigurationsdaten hochzuladen und zu manipulieren. Das Protokoll erlaubt es einem Gerät eine API zu diesen Zwecken anzubieten. Applikationen können diese API benutzen um auf Gerätekonfigurationen zuzugreifen und sie zu ändern.
NETCONF wurde entworfen, um das Verwalten von Netzwerkgeräten per SSH zu abstrahieren. Es greift selbst auch per SSH auf die Geräte zu, allerdings sind die Daten strukturiert.
Um dies zu erreichen benutzt NETCONF eine XML-basierte Datenencodierung für die Konfigurationsdaten und auch die Protokoll-Nachrichten.
Das NETCONF-Protokoll arbeitet auf einem RPC-Layer, d.h. ein Host kann extern definierte Methodenaufrufe ausüben.
(IETF, 2017)

Protokoll-Layer

Das NETCONF-Protokoll ist konzeptuell in 4 Layer aufgeteilt:

- Content
- Operations
- Messages
- Secure Transport

Content-Layer

Der Inhalt von NETCONF-Operations ist XML und bezieht sich grösstenteils auf Netzwerk-Management. Mittlerweile werden auch das JSON-Format und YANG unterstützt.

Operations-Layer

Das Grundprotokoll von NETCONF erlaubt folgende Operationen:

Operation	Beschreibung
<get>	Ruft die aktuell laufende Konfiguration und den Gerätestatus ab
<get-config>	Ruft eine bestimmte Konfiguration ab
<edit-config>	Manipuliert eine Konfiguration
<copy-config>	Kopiert eine Konfiguration an einen anderen Ort
<delete-config>	Löscht eine Konfiguration
<lock>	Blockiert die Konfiguration eines Geräts
<unlock>	Gibt die Konfiguration eines Geräts frei
<close-session>	Terminiert eine laufende NETCONF-Session (kontrolliert)
<kill-session>	Erzwingt die Terminierung einer NETCONF-Session

Die Grundoperationen von NETCONF sind erweiterbar und werden direkt per Server und Client ausgehandelt.
Später wurde noch eine Möglichkeit zum *Subscriben* von Events hinzugefügt. Dies kann mittels des <create-subscription> Befehls erreicht werden, welcher das Erstellen von *Subscriptions* in Echtzeit erlaubt. Damit werden Benachrichtigungen asynchron mittels des <notification> Konstrukts verschickt.

Messages

Die NETCONF-Messages bieten einen transportunabhängigen Weg um *RPC invocations (<rpc> messages)*, *RPC results (<rpc-reply> messages)* und *even notifications (<notification> messages)* zu kodieren.

YANG

YANG ist eine Datenmodellierungssprache und wird benutzt um Konfigurations- und Statusdaten zu beschreiben. YANG ist nicht objekt-orientiert, sondern ist baumstrukturiert, d.h. die Konfigurationsdaten sind aufgebaut wie ein Baum und die Daten können komplexe Typen wie Listen oder Vereinigungsmengen beinhalten.
(Barroso, YANG for Dummies, 2017)

YANG Terminologie (RFC 6020)

(IETF, 2017)

augment	Adds new schema nodes to a previously defined schema node
container	An interior data node that exists in at most one instance in the data tree. A container has no value, but rather a set of child nodes.
data model	A data model describes how data is represented and accessed.
data node	A node in the schema tree that can be instantiated in a data tree. One of container, leaf, leaf-list, list, and anyxml.
data tree	The instantiated tree of configuration and state data on a device.
derived type	A type that is derived from a built-in type (such as uint32), or another derived type.
device deviation	A failure of the device to implement the module faithfully.
grouping	A reusable set of schema nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it. The grouping statement is not a data definition statement and, as such, does not define any nodes in the schema tree.
Identifier	Used to identify different kinds of YANG items by name.
Leaf	A data node that exists in at most one instance in the data tree. A leaf has a value but no child nodes.
leaf-list	Like the leaf node but defines a set of uniquely identifiable nodes rather than a single node. Each node has a value but no child nodes.
list	An interior data node that may exist in multiple instances in the data tree. A list has no value, but rather a set of child nodes.
module	A YANG module defines a hierarchy of nodes that can be used for NETCONF-based operations. With its definitions and the definitions it imports or includes from elsewhere, a module is self-contained and “compilable”.
state-data	The additional data on a system that is not configuration data such as read-only status information and collected statistics [RFC4741].

YANG Beispiel

Folgendes Beispiel wurde von napalm-automation.net/yang-for-dummies/ entnommen:

```
// module name
module napalm-star-wars {

    // boilerplate
    yang-version "1";
    namespace "https://napalm-yang.readthedocs.io/napalm-star-wars";

    prefix "napalm-star-wars";

    // identity to unequivocally identify the faction an individual belongs to
    identity AFFILIATION {
      description "To which group someone belongs to";
    }

    identity EMPIRE {
      base AFFILIATION;
      description "Affiliated to the empire";
    }

    identity REBEL_ALLIANCE {
      base AFFILIATION;
      description "Affiliated to the rebel alliance";
    }

    // new type to enforce correctness of the data
    typedef age {
      type uint16 {
        range 1..2000;
      }
    }

    // this grouping will all the personal data we will assign to individuals
    grouping personal-data {
        leaf name {
            type string;
        }
        leaf age {
            type age;
        }
        leaf affiliation {
            type identityref {
                base napalm-star-wars:AFFILIATION;
            }
        }
    }

    // this is the root object defined by the model
    container universe {
        list individual {
            // identify each individual by using the name as key
            key "name";

            // each individual will have the elements defined in the grouping
            uses personal-data;
        }
    }
}
```

Abbildung 7: YANG Modelldefinition

Hier wird anhand eines Beispiels aufgezeigt, wie man mit YANG Personen im Star-Wars Universum beschreiben kann:

- <Identitiy AFFILIATION> ist eine erfundene *identity* um Personen eindeutig einer Fraktion zuteilen zu können. Es wurden zwei Typen bestimmt: EMPIRE & REBELS. Alle Personen die mit diesem Modell dargestellt werden, müssen der einen oder der anderen Fraktion angehören.
- <age> beschreibt das Alter der Personen. Uint16 ist ein selber definierter Typ, um dem Alter eine Einheit geben zu können.
- <grouping personal-data> teilt dem Personenmodell die *leafs* zu, also Felder, die jede Person haben muss. Hier sind das die oben beschrieben Felder <age> und <identity>.
- <container universe> stellt den Container dar, in dem alle Modelle beinhaltet sein müssen (→die „Wurzel“ des Baumobjekts). Jedes Modell innerhalb des Containers verfügt über einen Namen und über die Felder die in <grouping personal data> definiert wurden.

Mit diesem Modell kann man nun Personen des Star-Wars Universums beschreiben:

```
{
    "universe": {
        "individual": {
            "Obi-Wan Kenobi": {
                "affiliation": "REBEL_ALLIANCE",
                "age": 57,
                "name": "Obi-Wan Kenobi"
            },
            "Luke Skywalker": {
                "affiliation": "REBEL_ALLIANCE",
                "age": 19,
                "name": "Luke Skywalker"
            },
            "Darth Vader": {
                "affiliation": "EMPIRE",
                "age": 42,
                "name": "Darth Vader"
            },
            "Yoda": {
                "affiliation": "REBEL_ALLIANCE",
                "age": 896,
                "name": "Yoda"
            }
        }
    }
}
```

Abbildung 8: Beispiel eines YANG Datensatzes

Es ist auch möglich andere Modelle zu importieren und selber zu erweitern:

```
// We import the old model
import napalm-star-wars { prefix napalm-star-wars; }

// New identity based off the old AFFILIATION
identity MERCENARY {
    base napalm-star-wars:AFFILIATION;
    description "Friend for money";
}
```

Abbildung 9: Importieren und Erweitern eines bestehenden Modells

Vor – und Nachteile

Vorteile:

- Wurde für Automation ausgelegt (Subscription <> Notifications)
- Modellgetrieben und transaktionsbasiert (mittels YANG & RPC)

Nachteile:

- Relativ neu, deswegen noch nicht so weit verbreitet

1.2.5 Syslog

Protokoll zum zentralen Sammeln von Informationen in einem Netzwerkumfeld. Auszug aus dem Abstract des RFC 5424 (IETF, 2017):
„This document describes the syslog protocol, which is used to convey event notification messages. This protocol utilizes a layered architecture, which allows the use of any number of transport protocols for transmission of syslog messages. It also provides a message format that allows vendor specific extensions to be provided in a structured way. “
Ein Server dient als zentrale Sammelstelle der Syslog Meldungen. Somit kann ein eventbasierter Troubleshooting-Prozess ausgelöst werden.

Message-Format

Eine Syslog Meldung beinhaltet einen *facility code* und einen *severety level*. Die Syslog Software fügt der Meldung noch zusätzliche Informationen wie eine Prozess-ID, ein Zeitstempel und Hostname oder IP des Geräts, hinzu.
Der Facility Code beschreibt den Programmtyp, der die Meldung erstellt hat. Es existieren folgende *facilities*:

Facility Code	Keyword	Description
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally by syslogd
6	lpr	line printer subsystem
7	news	network news system
8	uucp	UUCP subsystem
9		clock daemon
10	authpriv	security/authorization messages
11	ftp	FTP daemon
12	-	NTP subsystem
13	-	log audit
14	-	log alert
15	cron	scheduling daemon
16-23	local0-7	local use 0-7

Der Severity Level zeigt die Schwere des aufgetretenen Fehlers an. Es existieren folgende *severity levels*:

Value	Severity	Keyword	Description
0	Emergency	emerg	System is unusable
1	Alert	alert	Action must be taken immediately
2	Critical	crit	Critical conditions, eg. hard device errors

3	Error	err	Error conditions
4	Warning	warning	Warning conditions
5	Notice	notice	Normal but significant conditions
6	Informational	info	Informational messages
7	Debug	debug	Debug-level messages

Die Bedeutung der verschiedenen Severity Levels kann von der Applikation, die das Syslog Protokoll benutzt, selbst bestimmt werden.

Konfiguration

Syslog wird über eine zentrale Konfigurationsdatei gesteuert und heisst meistens */etc/syslog.conf*. In der Datei wird beschrieben, wie Meldungen zu schreiben sind und wie diese über das Netzwerk übertragen werden (wikibooks, 2017). Die Datei hat folgende Form:

```
#important kernel-level warnings
Kern.warning            /dev/tty10
```

Links steht ein *facility-severity* Paar, welches angibt was für eine Message erzeugt werden soll. Rechts steht, wohin die Messages geloggt werden sollen.

Will man eine zentrale Sammelstelle für Syslog Meldungen definieren so geht das folgendermassen:

```
#central logging server gets a copy of all the warnings
*.warn                  @192.168.1.1
```

So werden alle auf diesem Gerät generierten warnings an die IP-Adresse 192.168.1.1 kopiert und geschickt.

Vor- und Nachteile

Vorteile:

- Grösserer Funktionsumfang als z.B. SNMP-Traps (bis zu 6000 unterschiedliche Meldungen bei Cisco)
- So können mittels Syslog sehr viele Probleme die in einem Netzwerk auftauchen, unterschieden werden

Nachteile:

- Syslog Meldungen sind plaintext-basiert, d.h. haben keine vorbestimmte Form (können aber durch verschiedene Tools aber in Objekte/Modelle umgewandelt werden)
- Unterstützt nur PUSH Nachrichten, Syslog Konfigurationen können nicht durch das Syslog Protokoll selbst remote geändert werden

1.2.6 Netflow

Netflow ist ein Feature welches die Möglichkeit bietet Network Traffic zu sammeln sobald er ein Interface betritt oder verlässt. Die gesammelten Informationen werden in UDP Datagrammen exportiert. Ursprünglich war Netflow eine Cisco Technik, sie wurde aber von vielen Herstellern übernommen. Netflow ist wie SNMP ein Messverfahren, d.h. es werden Informationen über den Verkehr gesammelt aber keine Änderungen vorgenommen. Router und Switches welche Netflow unterstützen können auf allen Interfaces auf denen Netflow aktiviert ist IP Traffic Informationen sammeln.
(InterProjektWiki, 2017), (IETF, 2017)

Je nach Implementation des Herstellers unterscheiden sich die Informationen die in einem Network Flow enthalten sind. Unter Cisco beinhaltet ein Packet folgendes:

- IP Source Adresse
- IP Destination Adresse
- Source Port
- Destination Port
- Layer 3 Protocol
- Type/Class of Service
- Router/Switch Interface

Pakete die diese Informationen gemeinsam haben können als Flow zusammengefasst werden. Diese Flow Aufzeichnung beinhaltet weitere Informationen wie

- Input Interface Index von SNMP, ifIndex falls IF-MIB verwendet wurde
- Output Interface Index oder null falls das Paket gedroppt wurde
- Timestamps des Starts und Ende des Flows
- Anzahl Bytes und Pakete in einem Flow
- Layer 3 Header
- Source & destination IP addresses
- ICMP Type und Code
- IP Protokoll
- Type of Service (ToS) value
- Source und destination Portnummern für TCP, UDP, SCTP
- Für TCP Flows die Gemeinsamkeiten aller TCP Flags
- Layer 3 Routing Informationen, IP Adresse des next hop auf der Route

Die Analyse dieser Flow Daten dient dazu sich ein Bild über den traffic Flow und Volumen eines Netzwerks zu erstellen.

Anwendung

Ein Netflow Setup besteht meistens aus drei Komponenten welche an ein Netzwerk angeschlossen werden.

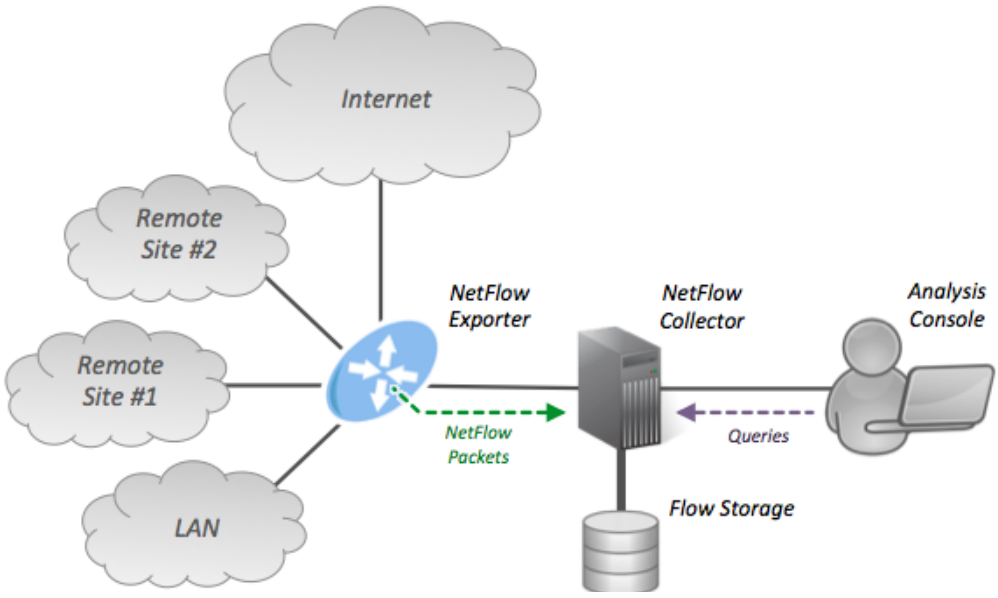


Abbildung 10: NetFlow Funktionsweise (Wikipedia, 2017)

Flow Exporter	Sammelt Pakete aus einem Netzwerk und bündelt diese in Flows, welche dann zum Collector exportiert werden.
Flow Collector	Die Exportierten Flow Pakete werden vom Collector erhalten und gespeichert.
Analysis application	Um die erhaltenen Flow Daten analysieren zu können braucht es eine Basis mit der man Informationen vergleichen kann und daraus Schlüsse ziehen kann.

Versionen und Alternativen

Netflow wurde ursprünglich von Cisco erstellt und weiterentwickelt. Diverse weitere Hersteller haben ihr eigenes Flow Monitoring entwickelt. Die weitverbreitetste Alternative ist sFlow. Welche jedoch einige Unterschiede zu Netflow aufweist, so erlaubt sFlow das exportieren von Paket Daten Chunks und Interface Counters.
Grundsätzlich unterstützen die meisten Netzwerk Geräte Netflow Versionen 5 und 9, jedoch gibt es auch hier Unterschiede.

Vor- und Nachteile

Vorteile:

- Eine grosse Anzahl Pakete können zusammengefasst werden und mit Einbezug des Kontexts analysiert werden

Nachteile:

- zusätzliche Processing-Power nötig für jedes Paket
- Viele verschiedene Versionen und Support dieser sind nicht für alle möglichen Geräte gewährleistet

1.2.7 Netzwerk-Sniffing

Ein Netzwerk-Sniffer überwacht den Datenfluss innerhalb eines Netzwerks in Echtzeit. Sniffer können in Software oder Hardware umgesetzt sein.
(Mitchell, 2017)

Funktionsweise

Sniffer erstellen Kopien von Paketen, ohne dass die originalen Pakete umgeleitet oder geändert werden. Dies erlaubt es einer Applikation oder einem Benutzer den Inhalt des Pakets genau zu untersuchen. Sofern die Pakete nicht verschlüsselt sind, kann so der gesamte Netzwerk-Protokoll-Stack des Pakets, von L2 bis L7(OSI), überprüft werden. Beim TCP-Protokoll kann so z.B. eine genaue Statistik erstellt werden, wie oft eine Verbindung neu aufgebaut werden muss und wie viele Pakete unterwegs gedroppt werden.
So existieren diverse Tools, mit denen in Echtzeit Pakete analysiert werden. Sniffer Pro z.B. kann so den Durchsatz, die Auslastung, Errors/s messen und im Problemfall Alarme (→events) auslösen.

Vor- und Nachteile

Vorteile:

- Erlaubt es Fehler zu entdecken, die sonst unerkannt bleiben
- Erlaubt Durchsatzmessungen (→Erkennung langsamer Verbindungen)

Nachteile:

- Es ist schwierig in Echtzeit den Datenverkehr zu analysieren und in kurzer Zeit darauf zu schliessen wo der Fehler liegt.
- Packet-Inspection basiert meistens auf der Suche nach gewissen Stichworten (z.b /etc/shadow), dies ist nützlich bei der Überwachung eines Netzes, aber nur wenig geeignet um Troubleshooting-Prozesse auszulösen.

1.2.8 IP SLA - Internet protocol service level agreement

Internet Protocol Service Level Agreement oder kurz IP SLA ist ein Protokoll von Cisco IOS, welches es erlaubt Informationen über Network Performance in Echtzeit zu sammeln. Dieses Monitoring wird über ein aktives Verfahren erstellt, d.h. Ein IP SLA Router hat die Fähigkeit, Traffic zu generieren und diesen direkt zu überprüfen.
(Cisco, 2017),

Die SLAs sammeln Daten über Antwortzeit, Latenz, Jitter, Paketverluste, Connectivity, Server oder Webseiten Antwortzeit und Downtime, Pfad und Paket Sequenzierung. Mit diesen Werten ist es mögliche eine grundsätzliche Information über die Netzwerk Performance zu sammeln. Zusätzlich ist es so möglich die Quality of Service überprüfen und man sieht wo die Performance zu niedrig ist. Innerhalb eines Network Management Systems ist es zusätzlich möglich eine Visualisierung der Grenzwerte und Überschreitungen dieser zu erhalten.

IP SLA benötigt zwei Geräte in einem zu überprüfenden Netzwerk, den IP SLA Router welcher Traffic erzeugt, der zweite ist der Responder, dieser kann von einem beliebigen Hersteller sein.

Anwendung

Hier eine Liste mit Messungen die via IP SLA erstellt werden können sowie die verwendeten Technologien dazu.

Cisco IOS IP SLAs Operation	Messungen
UDP Jitter	Misst round-trip Verzögerung, one-way Verzögerung, one-way Jitter, one-way Paketverlust und Connectivity-Testing von Netzwerken welche UDP traffic besitzen wie etwa Voice Channels.
ICMP Path Jitter	Misst hop-by-hop Jitter, Paketverlust, and Verzögerung
UDP Jitter für VoIP	Misst round-trip Verzögerung, one-way Verzögerung, one-way Jitter, and one-way Paketverlust von VoIP traffic.
UDP Echo	Misst round-trip Verzögerung von UDP traffic.
ICMP Echo	Misst round-trip Verzögerung für den ganzen Pfad.
ICMP Path Echo	Misst round-trip Verzögerung and hop-by-hop round-trip Verzögerung.
HTTP	Misst round-trip time um eine Webseite abzurufen.
TCP Connect	Misst die Zeit die benötigt wird eine TCP Verbindung herzustellen.
FTP	Misst round-trip time um ein File zu senden.
Dynamic Host Configuration Protocol (DHCP)	Misst round-trip time die benötigt wird eine IP Adresse von einem DHCP Server zu erhalten
Domain Name System (DNS)	Misst DNS lookup Zeit.
Data Link Switching Plus (DLSw+)	Misst peer tunnel Antwortzeit.
Frame Relay	Misst circuit Verfügbarkeit, round-trip Verzögerung, and frame delivery Rate. Diese Operation hat keine SNMP Unterstützung.

Vor- und Nachteile

Vorteil:

- Überprüfen des Netzwerks von einem zentralen Punkt.
- Viele Konfigurationsmöglichkeiten.
- Erreichbar durch SNMP

Nachteil:

- IP SLA ist eines Funktion von Cisco Geräten, mindestens ein Gerät im zu überprüfenden Netzwerk muss ein Cisco Gerät sein und IP SLA fähig sein.
- Das Monitoring funktioniert über aktiv generierten Traffic der zusätzlich das Netzwerk belastet.
- Die Überprüfungen müssen sehr genau spezifiziert werden.

1.2.9 SSH – Secure Shell

SSH ist ein Netzwerk Protokoll welches die Funktionalität bietet eine sichere Verbindung über ein unsicheres Netzwerk zu erstellen und somit sichere Netzwerk Operationen auszuführen, hierbei wird ein Client –Server Ansatz verwendet.
In einem Netzwerk biete es uns die Möglichkeit remote Command-Line Funktionen auszuführen. So können von einem zentralen Punkt Gerätekonfigurationen überprüft und manipuliert werden.

1.3 Problemanalyse

In einem Netzwerk können diverse Probleme auftauchen, im Rahmen dieser Analyse wollen wir uns auf die Probleme im Layer eins bis vier gemäss dem OSI Modell konzentrieren. Hier eine Erläuterung der vier relevanten Layer.
(Microsoft, 2017), (Rivard, 2017)

1.3.1 OSI-Modell

	Layer	Daten	Beschreibung
Host-Layer	4. Transport	Segment (TCP) Datagram (UDP)	Verlässliche Übertragung von Daten Segmenten zwischen zwei Punkten eines Netzwerks, inklusive Segmentierung, Bestätigung und Bündelung
Media-Layer	3. Netzwerk	Paket	Strukturierung und Verwaltung eines Netzwerks inclusive Adressierung, Routing und traffic Kontrolle
	2. Data Link	Frame	Verlässliche Übermittlung von Data Frames zwischen zwei Physisch verbundenen Punkten
	1. Physical	Bit	Übermittlung von raw bit Streams über eine Physisches Medium

Layer 1: Physical Layer

Der Physische Layer spezifiziert die Elektrische und Physischen Eigenschaften der Datenverbindung. Er definiert die Verbindung eines Geräts und des physischen Übermittlungskanals, z.B. ein Kupferkabel ein Glasfaserkabel oder eine Radiofrequenz Verbindung. Dazu gehören ausserdem das Layout der einzelnen Pins, die Stromspannung, der Wechselstromwiderstand, diverse Kabelspezifikationen, Signal Timing und weiter Charakteristiken welche für verbundenen Geräte benötigt werden.

Es ist verantwortlich für die Übermittlung und den Empfang von unstrukturierten Rohdaten über ein physisches Medium. Übertragungsmodi wie Simplex, Halbduplex und Fullduplex werden hier definiert.

Der physische Layer bietet die Grundlage für alle darüber liegenden Layer. Funktioniert der Layer nicht, so ist kein Datenverkehr möglich
Auf dem physischen Layer gibt es die low-level Netzwerkgeräte, wie etwa Hubs, Netzwerkadapter, Glasfaserkonvertierer und Repeater. Der physische Layer interessiert sich nicht für Protokolle der höheren Layer.

Layer 2: Data Link Layer

Der Data Link Layer bietet Punkt zu Punkt Datentransfer, eine Verbindung zwischen zwei direkt verbundenen Punkten. Er detektiert und korrigiert eventuell vorkommende Fehler aus dem physischen Layer. Dieser Layer definiert Protokolle um Verbindungen zwischen zwei physischen Verbundenen Punkten aufzubauen und zu beenden. Diese Protokolle können auch Flow Control durchführen.

Der IEEE 802 Standard welcher sich mit Netzwerkstandards für Layer 1 und 2 auseinandersetzt Definiert zwei Sublayer im Data Link Layer:

Der Media Access Control, MAC, Layer, dieser ist verantwortlich für die Kontrolle wie zwei Geräte im Netzwerk eine Verbindung zu einem Medium aufbauen und dessen Erlaubnis Daten zu übermitteln.

Der Logical Link Control, LLC, Layer ist verantwortlich für die Identifizierung und Verkapselungsfunktionen, sowie die Fehlerüberprüfung und Frame-Synchronisation.

Layer 3: Network Layer

Der Netzwerk Layer bietet die Funktionalität und Prozeduren um Datensequenzen von variabler Länge, auch Datagramme genannt, von einem Gerät zu einem anderen Gerät zu übermitteln, auch wenn diese in unterschiedlichen Netzwerken liegen.
Ein Netzwerk ist hier eine Verbindung, ein Netz, von verschiedenen Geräten, die alle eine eigene Adresse haben und es erlaubt den verbunden Geräten Mitteilungen zu senden, solange man deren Adresse kennt. Das Netzwerk übernimmt das Senden der Nachricht über einen Pfad. Falls die Nachricht zu gross ist um einzeln übermittelt zu werden, bietet der Netzwerk Layer die Möglichkeit diese in Fragmente aufzuteilen, einzeln zu übermitteln und an einem anderen Punkt wieder zusammen zu führen. Ausserdem besteht die Möglichkeit Übermittlungsfehler zu signalisieren.

Die Nachrichtenübermittlung über den Netzwerk Layer muss nicht zwingend zuverlässig sein. Ein Netzwerk Protokoll kann diese Funktionalität bieten, muss es aber nicht.

Zum Netzwerk Layer gehören Protokolle welche Routing übernehmen, Gruppenverwaltung für Multicast, Netzwerk Layer Informationen, Fehler anzeigen, sowie Netzwerk Layer Adressverwaltung.

Layer 4: Transport Layer

Der Transport Layer bietet Funktionalitäten und Prozeduren um Daten Sequenzen von variabler Länge von einem Ursprungs- zu einem Ziel-Host zu übermitteln und die Quality of Service Funktionalität beizubehalten, die Hosts können über ein oder mehrere Netzwerke verteilt sein.

Der Transport Layer kontrolliert die Zuverlässigkeit einer beliebigen Verbindung, über Flow Control, Segmentierung und Fehler Kontrolle. Protokolle können Status und Verbindungs- orientiert sein. Segmente können verfolgt und bestätigt werden, dadurch kann man diese erneut übermitteln falls sie nicht ankommen. Der Transport Layer kann einzelne Nachrichten welche er von höheren Layer erhält in Pakete bündeln.

Die Hauptsächlich verwendeten Protokolle im Transport Layer sind UDP und TCP, welche unterschiedliche Eigenschaften und Verwendungszwecke haben.

1.3.2 Probleme

Da wir nun wissen welche Funktionen uns auf jedem Layer zur Verfügung stehen können wir eine Liste mit möglichen Problemen, welche wir in einem Netzwerk antreffen könnten, erstellen.

Layer 1 Probleme

Power

- Geräte können ausgeschaltet werden oder ihre Power geht aus.
- **Detektieren:** Ein Interface ist nicht mehr verfügbar.

Disconnect – Interface

- Ein mit einem Interface verbundenes Gerät wird plötzlich entfernt.
- **Detektieren:** Ein Interface ist nicht mehr verfügbar.

Bitfehler

- Einzelne Bits können fehlerhaft sein, was zu grösseren Fehler oder sogar Paketverlust führen kann.
- **Detektieren:** Der CRC ist fehlerhaft.

Kabel Probleme

- Ein Kabel ist beschädigt, weshalb eine Verbindung unzuverlässig wird.
- **Detektieren:** Der CRC ist bei der gleichen Verbindung oft fehlerhaft.

Runts

- Ein Paket ist kleiner als die 64 Byte, wegen einer schlechten Verkabelung oder einem fehlerhaftem elektrischen Interface. Auch Überbleibsel einer Paketkollision können als runt enden.
- **Detektieren:** Cisco Geräte haben unter dem „show interface“ Befehl einen Zähler für Runts.

Temperature, power circuit overload

- Physische Fehler die auftreten können wenn mit elektrischen Geräten gearbeitet wird.
- **Detektieren:** Volt- und Temperaturmessungen

Layer 2 Probleme

Giants

Ein Paket welches mehr als 1500 Bytes Payload hat.
Detektieren: Falsche und unterschiedliche MTU-Einstellungen der Geräte

CRC Fehler

- Der “cyclic redundancy value” auf einem Switch oder Router entspricht nicht dem berechneten Wert.
- **Detektieren:** Geräte haben einen Zähler für wie viele CRC Fehler bemerkt werden.

Collisions

- Pakete auf einem Fullduplex oder Halbduplex Interface können Kollidieren, kann zu runts führen.
- Late collisions, sind eine Art der Kollision welche nach den ersten 64 Byte des Frames erfolgen.

- **Detektieren**: Kollisionszähler auf den Geräten überprüfen.

STP Fehler
Das Spanning Tree Protokoll ist zuständig für die Inter-Switch Kommunikation.

- Obwohl ein Switch in einen Arbeitsfähigen Modus wechseln müsste bleibt er in einem blockierenden Zustand,“blocking“ state.
- **Detektieren:** Abrufen der Zustandsinformation eines Switches.
- Nachdem ein weiterer Switch an das Netz angeschlossen wurde, kommt es zu einem Wechsel in der Root Bridge, dies kann zu einer Massenhaften Sendung von BPDU Pakten führen.
- **Detektieren:** Unerwartet Hoher Traffic, Priorität der Switches in den STP Einstellungen
- Da STP Parallele Verbindungen blockiert um Loops zu verhindern, ist der effektive Durchsatz eines Netzwerks kleiner als der eigentlich mögliche.
- **Detektieren:** Durchsatzmessungen
- Grosse Layer 2 Netzwerke haben oft ineffiziente oder Falsche Pfade welche aus der STP device priority resultieren.
- **Detektieren:** Einträge in der STP Priorität Liste überprüfen, Durchsatzmessungen.
- Obwohl STP Loops verhindern sollte kommt es zu einem Loop im Netzwerk.
- **Detektieren:** Unerwartet Hoher Traffic, dasselbe Paket wird mehrmals gesendet.

Flapping MAC-Adresse (Morell, 2017)

- Ein Switch erhält auf unterschiedlichen Interfaces Pakete von der gleichen MAC-Adresse.
- STP funktioniert nicht korrekt auf einem VLAN.
- **Detektieren:** MAC-Adressen Tabelle eines Switches auslesen, viele Switches loggen wenn sie ein Flapping detektieren.

Broadcast Sturm durch DHCP Discovery

- In einem Netz mit einem DHCP Server sendet dieser eine exzessive Menge an DHCP Pakete.
- **Detektieren:** Geräte erhalten DHCP Pakete die sie nicht angefragt haben.

Duplex Mismatch

- Zwei miteinander verbundene Geräte haben unterschiedliche Duplex Einstellungen, was zu Paketverlusten und Kollisionen führt.
- **Detektieren:** Überprüfen der Duplexeinstellung auf den betroffenen Geräten.

MAC Addressing Errors

- Doppelte MAC-Adressen in der MAC-Address Table eines Switches
- Ein Switch fügt eine MAC-Adresse eines Geräts nicht in seine MAC-Adressen Tabelle hinzu.
- **Detektieren:** Überprüfen der MAC-Adress Tabelle

Ungültige ARP Einträge

- Ein Gerät fügt ungültige Einträge in seine ARP Tabelle ein, falls diese für längere Zeit bestehen blieben, zB. Wenige Minuten, kann man in dieser Zeit den betreffenden Host nicht erreichen.
- **Detektieren:** Gedroppte Pakete wegen ungültiger Adresse, ARP Tabellen Einträge.

Geräte im gleichen VLAN können nicht erreicht werden

- Switchports sind nicht im korrekten VLAN eingetragen.
- Auf einem Trunk Port ist VLAN nicht erlaubt.
- **Detektieren:** Pakete werden beim Switch gedroppt, Interface/Port Einstellungen überprüfen.

Gerät kann andere Geräte im Netz nicht erreichen

- Die Verbindung zwischen zwei Switchen funktioniert nicht, da sie sich in einem anderen Switchport Modus befinden (access/trunk).
- **Detektieren:** Anzeigen der Switchport Einstellungen auf beiden Geräten.
- Die Verbindung zwischen zwei Switchen funktioniert nicht, da sie andere Protokolle für die Channel Aggregation verwenden, zB. PAgP und LACP.
- **Detektieren:** Anzeigen der Switchport Einstellungen auf beiden Geräten.

- Die Verbindung zwischen einem Switch und einem Router funktioniert nicht, da sie andere Protokolle für ihre Trunk Encapsulation verwenden.
- **Detektieren:** Anzeigen der Interface Trunk Einstellungen auf beiden Geräten.

Layer 3 Switch Probleme

- Ein Layer 3 Switch muss auf den Ports die mit dem Layer 3 Netzwerk interagieren Routing aktivieren, hat er aber nicht.
- **Detektieren:** Anzeigen der Switchporteinstellungen auf dem spezifischen Port.
- Ein Layer 3 Switch hat das IP Routing standardmässig deaktiviert, müsste dies aber aktivieren um mit dem Layer 3 Netz zu interagieren.
- **Detektieren:** Anzeigen der IP Interface Einstellungen auf dem Switch.

Port Security Violation

- Ein Gerät hat keine Verbindung ins Netz, da er vom Switch blockiert wird, dieser hat eine Port Security Violation gemeldet.
- **Detektieren:** Die erlaubte statische MAC Adresse in den Port Security Einstellungen ist fehlerhaft.

CDP Probleme

- Fehlerhafte Einträge in der CDP Neighbor Tabelle.
- **Detektieren**: Überprüfen der CDP Neighbor Tabelle
- Geräte die CDP nicht unterstützen fluten erhaltene CDP Pakete über das Netzwerk.
- **Detektieren:** Netzwerk wird mit CDP Paketen überflutet, hoher Traffic.

Layer 3 Probleme

Allgemeine Routing Probleme

- Suboptimale Routen, das verwendete Routing Protokoll wählt ungünstige oder suboptimale Routen für die Pakete, was zu einer allgemeinen Verlangsamung des Netzes führt.
- **Detektieren**: Pakete brauchen länger als erwartet.
- Routing Loops, Innerhalb des Routing gibt es einen Loop.
- **Detektieren**: Unerwartet Hoher Traffic dasselbe Paket wird mehrmals gesendet.

NAT

- Ein PC im Netz mit NAT hat kein Zugriff auf eine Internetseite.
- **Detektieren:** Der PC hat einen falschen Standard Gateway.
- Ein PC im Netz mit NAT hat kein Zugriff auf eine Internetseite, Falsche Inside und Outside Interfaces eingestellt. Die IP Adresse des PC liegt ausserhalb der Access List Range welche der Router eingestellt hat.
- **Detektieren:** Verifizieren der NAT Einstellungen und der Access Liste auf dem Router.

IPv6 Probleme

- Ein PC hat keinen Zugriff auf eine IPv6 Adresse im Internet, hat aber Zugriff auf den Router.
- **Detektieren:** Verifizieren ob Router Advertisement auf dem Router unterdrückt ist.
- Ein PC hat keinen Zugriff auf eine IPv6 Adresse im Internet, auch keinen Zugriff auf den Router. Es wird SLAAC verwendet.
- **Detektieren:** Überprüfen der Interface Einstellungen auf dem Router, SLAAC funktioniert nur wenn der IP Prefix auf /64 gesetzt ist.

OSPF

- Der OSPF-Prozess eines Routers friert ein.
- **Detektieren:** Auf den OSPF-Neighbors laufen die dead timer aus.
- In Abbildung 11 wird der Cisco-Troubleshooting-Prozess für OSPF dargestellt:

(Cisco, 2017)

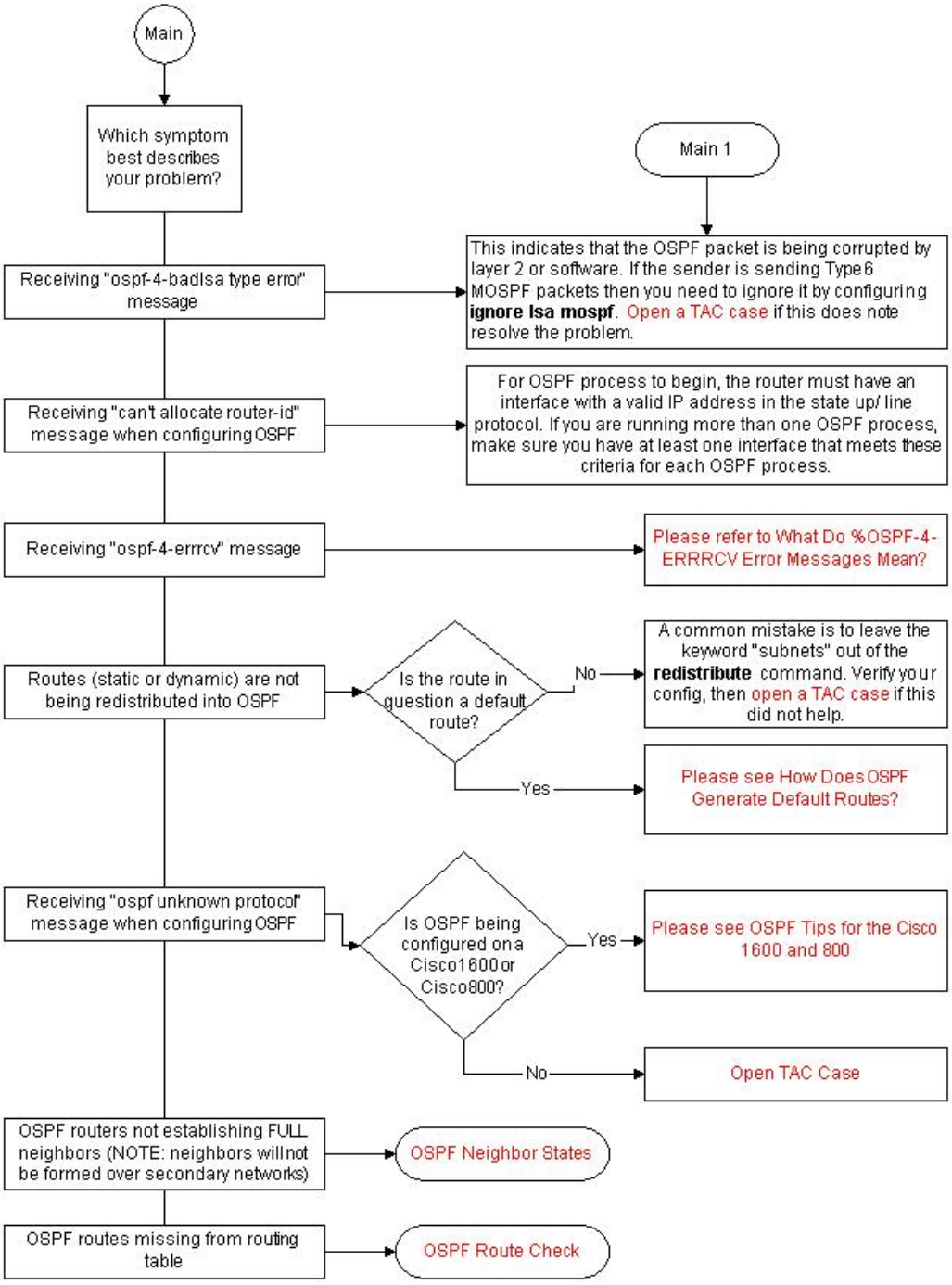


Abbildung 11: OSPF Troubleshooting-Prozess

EIGRP

- In Abbildung 12 wird der Cisco-Troubleshooting-Prozess für EIGRP dargestellt:

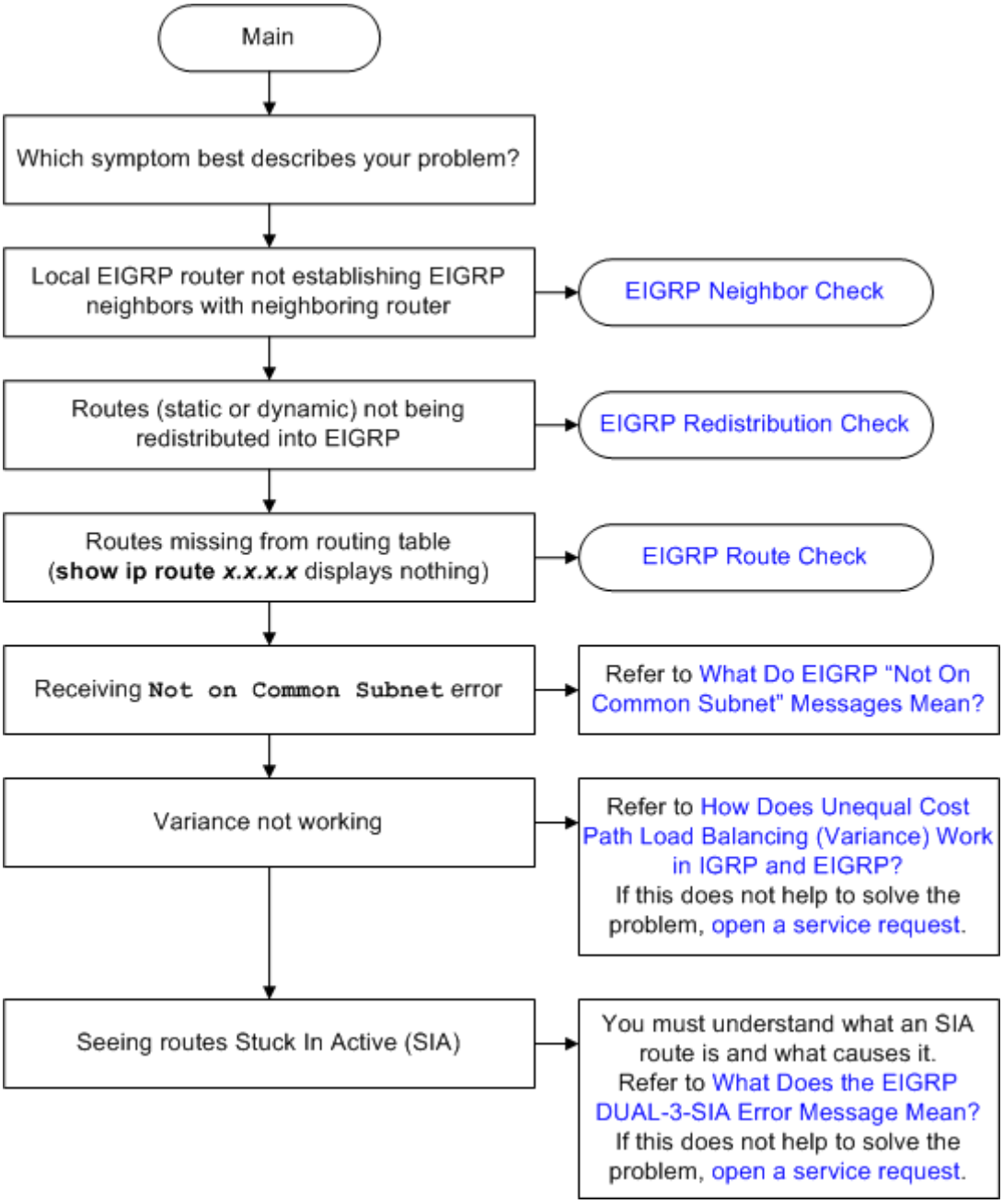


Abbildung 12: EIGRP Troubleshooting Prozess

Asymmetrisches Routing

- Ein Paket nimmt für eine beliebige Route zu einem Zielhost, jedoch eine andere Route für den Rückweg. Dies kann dazu führen das diese Paket eine Firewall nicht durchqueren kann.
- **Detektieren:** Nur Ein Weg des Traffic bleibt erhalten, überprüfen der Geräte die Pakete wegen Rückweg pfaden droppen könnten.

Link Saturation

- Wenn ein Link mehr Daten übertragen soll als er eigentlich kann, 110Mbps auf einem 100Mbps Link, platziert er die übrigen Pakete in einem Buffer, welcher er leert sobald die Sättigung zu Ende ist. Falls dies jedoch zu lange andauert und er den Buffer füllt, ist er gezwungen die Pakete zu droppen.
- **Detektieren:** Eine Erhöhung der Latenz, mehr Jitter und Erhöhung der gedroppten Pakete.

Layer 4 Probleme

Blockierte Ports

- Die Layer vier Protokolle TCP und UDP benötigen für ihre Kommunikation Ports, weshalb eine der häufigsten Probleme auf diesem Layer blockierte Ports sind, dies kann von einer Firewall oder direkt auf einer Access Liste eines Netzwerk Geräts sein.
- **Detektieren**: Überprüfen der Liste mit blockierten Ports, nachdem Pakete nicht ankommen.

Fragmentierung von grossen Datagrammen

- Wenn Datenstreams für die unteren Layer in Pakete umgewandelt werden, kann dies zu einigen Problemen führen.
- Falls bei TCP ein Datagramm in kleinere Fragmente aufgeteilt wird und eines dieser nicht ankommt müssen alle erneut gesendet werden. Dies kann zu Verlangsamungen im Netz führen.
- **Detektieren:** Langsames Netz, hohe Packet drop-rate.
- Falls mehrere fragmentierte Pakete in falscher Reihenfolge an eine Firewall kommen kann es sein dass die Firewall die Pakete droppt da ihr das initiale Paket fehlt.
- **Detektieren**: Firewall droppt Pakete, Verlangsamung des Netzwerks.

Applikation Layer Probleme

HSRP Probleme

- Traffic läuft durch den falschen Switch, da die Priorität beim HSRP falsch ist, ist der Switch der aktiv sein sollte im Standby.
- **Detektieren:** Anzeigen der HSRP Prioritätseinstellungen auf dem Switch Interface.
- Traffic läuft durch den falschen Switch, der Router mit der höheren Priorität der aktiv sein müsste hat preemption deaktiviert.
- **Detektieren:** Anzeigen Standby Optionen auf dem Switch.
- HSRP-fail-over beim runterfahren des aktiven Switches funktioniert nicht korrekt.
- **Detektieren:** Anzeigen Standby Optionen auf dem Switch zeigt auch die Priorität decrement Option, falls der Switch failed muss diese so gewählt sein das die Priorität kleiner wird als die des Standby Switches.

Telnet

- Ein PC hat keinen Telnet Zugriff auf einen anderen PC, sonst funktionieren aber alle Verbindungen.
- **Detektieren:** Verifizieren der Access Liste auf dem Router, ob dieser Telnet erlaubt.

DHCP

- Ein PC hat keinen Zugriff auf den DHCP Server der hinter einem Router liegt. Da dieser keine IP Helper-Adresse gewählt hat.
- **Detektieren:** Verifizieren der Adressen Informationen auf dem Router.

RIP

- Ein PC hat keinen Zugriff auf einen anderen Bereich des Netzwerks, es wird RIP verwendet. Ein Interface des Routers ist im passiven Modus, müsste aber aktiv sein. Was dazu führt das er zwar RIP Informationen erhält aber keine senden.
- **Detektieren:** Verifizieren der Router RIP Einstellungen. Überprüfen der der gesendeten RIP Informationen.

BGP

- BGP läuft über TCP und ist ein Applikations Layer Protokoll, wegen seiner Rolle im Routing haben wir es hier trotzdem erwähnt. In Abbildung 13 wird der Troubleshooting Guide von Cisco für BGP dargestellt:

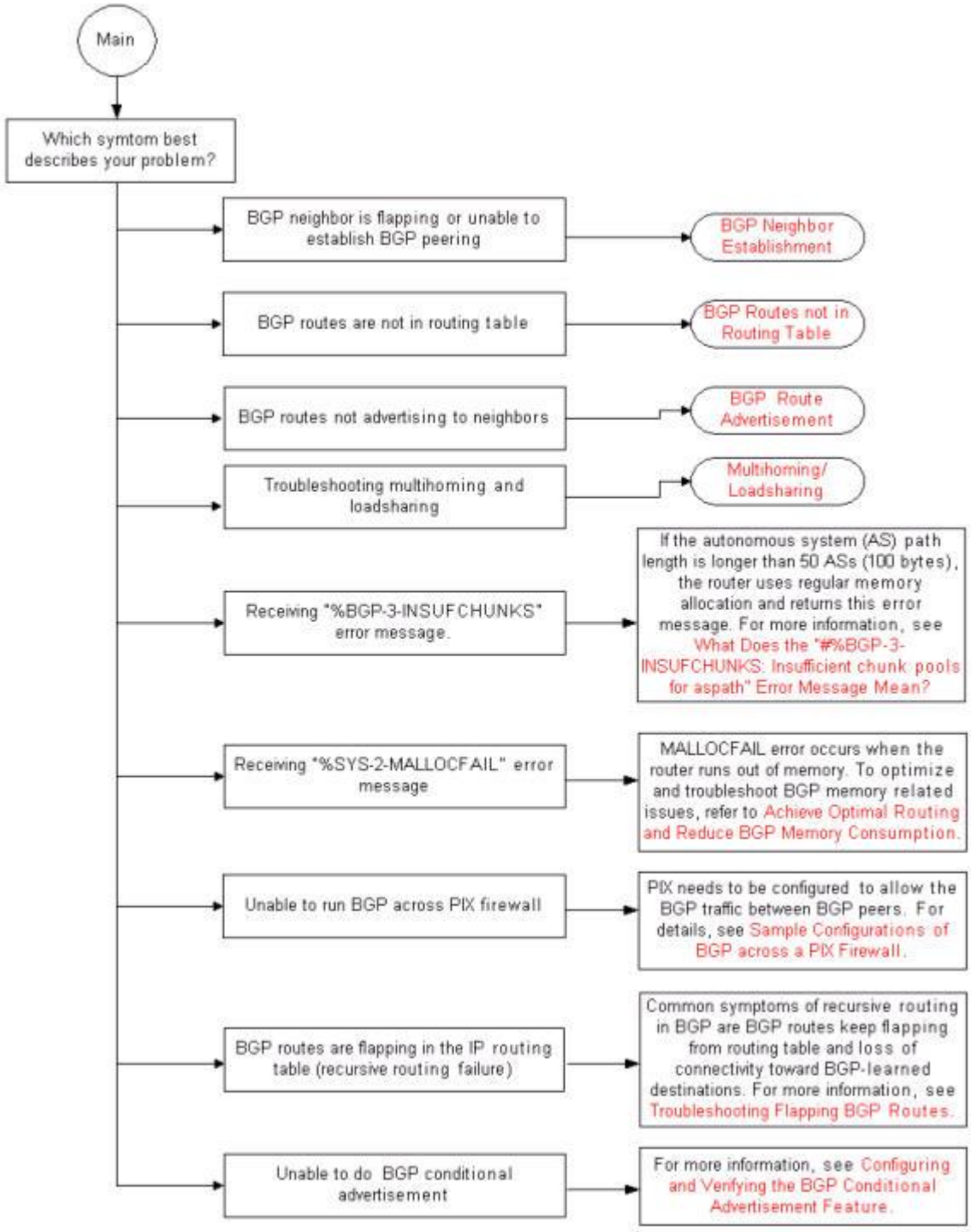


Abbildung 13: BGP Troubleshooting-Prozess

1.4 Deviceanalyse

Ein Netzwerk besteht in den meisten Fällen aus einer Heterogenen Mischung von Geräten. Diese unterscheiden sich sowohl in Funktionalität, ihren Rollen und die von Ihnen unterstützen Protokollen und Software.

1.4.1 Geräte in einem Netzwerk

Router

Ist ein Netzwerkgerät welches Netzwerkpakete zwischen verschiedenen Netzen weiterleiten kann. Die häufigsten Anwendungsfälle sind die Internetanbindung, die Kopplung mehrere Standorte oder direkt mehrere lokale Netzwerksegmente zu koppeln
Aus Sicht der Netzwerkschichten arbeiten sie auf der Schicht drei, sie treffen Entscheidungen über die Weiterleitung basierend auf IP Adressen. Eine weiter Funktion ist das übersetzen zwischen privaten und öffentlichen IP Adressen, NAT/PAT, einige Router können auch Firewall Funktionen übernehmen.

Switch

Ein Switch ist ein Netzwerkgerät welches verschiedene Geräte zu einem Netzwerk verbindet. Switches verwenden die Hardware-Adresse, auch MAC-Adresse genannt, zur Adressierung der Pakete. Sie arbeiten auf dem Layer 2 des OSI Modell.
Es gibt auch Switches welche auf dem Layer 3 arbeiten können und somit Routing Funktionen übernehmen und IP Adressen verwenden. Solche Switches nennt man auch Layer 3 Switches.

Firewall

Eine Firewall ist ein Sicherungssystem welches ein Netzwerk vor unerwünschten Zugriffen schützt. Die Firewall Software beschränkt Netzzugriffe basierend auf Absender oder Ziel und dem genutzten Dienst. Entscheidungen wer und was durchgelassen wird, entscheidet sie basierend auf vordefinierten Regeln. Eine Firewall erkennt keine Angriffe sondern setzt vordefinierte Regeln in der Netzwerkkommunikation um.

1.4.2 Probleme durch Geräte

Je nach Version des Geräts und der darauf installierten Software können sie nicht alle die gleichen Zusatzsysteme und Funktionen unterstützen. Eigenschaften die Einfluss haben auf die Unterstützung von Funktionen sind Hersteller, Version des Gerätes und auch die Version des Betriebssystems auf dem Gerät.
Grundsätzlich kann man auf alle Geräte über eine Konsole zugreifen einige bieten auch Support für eine API Schnittstelle an.

2. Anforderungen

Änderungsgeschichte

Datum	Version	Änderung	Autor
28.09.2017	1.0	Erstellung des Dokuments	Nv
30.09.2017	1.1	Überarbeiten Anforderungen Erstellen NFR	Nv
02.10.2017	1.2	Priorisierung der Anforderungen	nv, rj
16.10.2017	1.3	Updaten Use Cases	rj
16.10.2017	1.4	Überarbeitung nach der Construction	rj

2.1 Zweck

2.1.1 Beschreibung

Dieses Dokument beschreibt die Anforderungen an das automatische Troubleshooting System, welches im Rahmen der Studienarbeit von Nico Vinzens und Raphael Jöhl im FS2017 erarbeitet wird.

2.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung vom automatisierten Troubleshooting und während der nachfolgenden Bachelorarbeit.

2.1.3 Übersicht

In Kapitel 2.2 wird der Umfang des Systems erläutert. Darin werden unter anderem die Systemfunktionalität, das Systemumfeld und die Architektur beschrieben.
Im 3. Kapitel folgen die Anforderungen an das System in Form einer Tabelle. In der Tabelle sind die Anforderungen nummeriert, niedergeschrieben, sie werden gewichtet und es werden Abhängigkeiten innerhalb der Anforderungen aufgezeigt.
Das letzte Kapitel dokumentiert die nichtfunktionalen Anforderungen an das System.

2.2 Systemumfang

2.2.1 Stakeholder

- Betreuer der Arbeit: Sie bewerten den Ausgang der Arbeit und evaluieren das Endprodukt.
- Das Projektteam: Entwirft das System und ist für das Endprodukt verantwortlich.
- Systems-Engineers: potenzielle Benutzer des Systems, z.B. Systems Engineer eines Campus-Netzes eines KMUs

2.2.2 Systemfunktionalität

Das System dient zum automatischen Troubleshooting von Problemen die innerhalb eines Netzwerkes vorkommen.

2.2.3 Systemumfeld

Das System wird im Rahmen eines Campus-Netzes eingesetzt. Das Netz verfügt über je einen Access-, Distribution- und Core-Layer. Auf dem Core und Distribution Layer wird OSPF als Routing Protokoll eingesetzt.

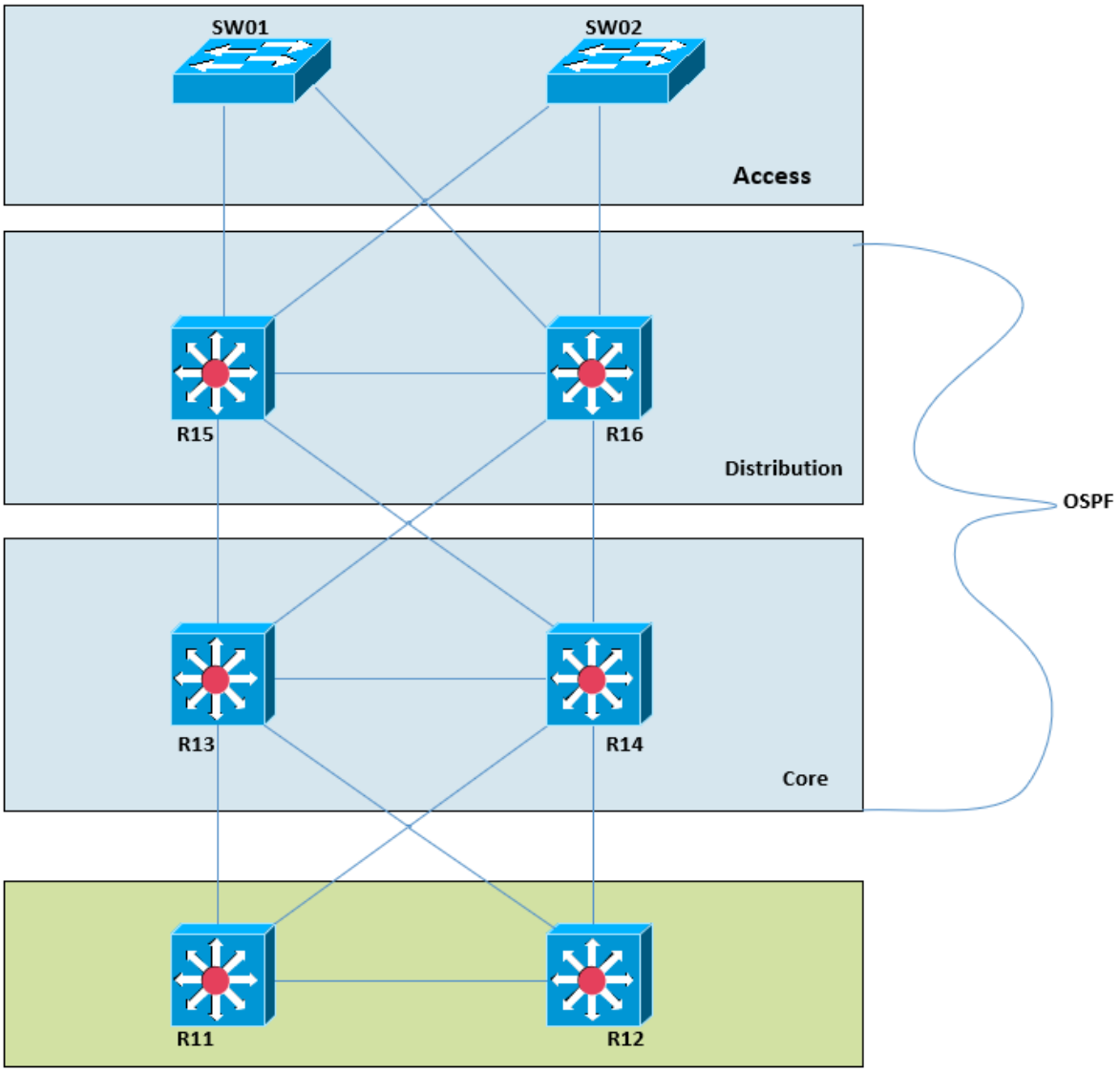


Abbildung 14: Campus-Netz

2.2.4 Architekturbeschreibungen

Das System soll aus einem oder mehreren bereits bestehenden Tools entworfen werden. Das Projektteam soll dabei die Tools sinnvoll miteinander verbinden bzw. die Schnittstellen zum System entwerfen. Auf diese Tools wird im Kapitel Architektur weiter eingegangen.
Das System kann grob in 6 Domänen (siehe Abbildung 15) unterteilt werden:

Erfassung

Das System wird durch ein Event gestartet, welches den Troubleshooting-Prozess in Gang setzt. Events könnten z.B. durch den Kunden (Ticket, Email, Telefon), oder direkt durch ein Monitoring System (z.B. Syslog) ausgelöst werden.
Das Tool soll dabei Berichte erstellen, die unabhängig der Herkunft des Events eine einheitliche Form haben, sowie den Troubleshooting Prozess starten.

Identifizierung

Das System identifiziert, von welchen Geräten, das Problem ausgeht und um welche Art von Problem es sich handelt. Sind die Geräte identifiziert, muss ermittelt werden, auf welchem Netzwerk-Layer das Problem besteht.

Klassifizierung

Das System teilt die Probleme in 2 unterschiedliche Oberklassen ein:

1. Durch das System selber lösbar
2. Nicht durch das System lösbar

Ist z.B. das betroffene Gerät wegen eines Kabelbruchs nicht mehr am Netz, so soll das System direkt einen Techniker benachrichtigen, der das Problem lösen soll.

Behebung

Ist das System in der Lage das Problem zu beheben, so wird ein Skript ausgeführt, welches gängige Netzwerk-Troubleshooting Schritte ausführt. Wir können uns dabei 2 unterschiedliche Arten vorstellen, wie dies ablaufen wird:

- Je nach Problemklasse wird ein anderes Skript ausgeführt
- Es wird für jedes Problem das gleiche Skript ausgeführt

Falls ein Techniker nötig ist, wird ihm ein Bericht zum Problem übermittelt und er muss das Problem manuell lösen.

Überprüfung

Das System kontrolliert, ob das Problem behoben wurde. Falls es durch das System selbst nicht behoben werden konnte, muss ein Techniker benachrichtigt werden, der das Problem manuell löst.

Reporting/Logging

Das System erstellt einen Logeintrag, in welchem alle Details des Troubleshooting-Prozesses einsehbar sind. Der Eintrag wird bereits mit der Erfassung des Problems erstellt und dann laufend mit den gesammelten Informationen aus dem Troubleshooting Prozess erweitert.
Wurde das Problem durch einen Techniker gelöst, erstellt dieser den Logeintrag manuell.
Der Auslöser des Events, soll am Ende des Prozesses informiert werden.

Abbildung 15: Problemdomänen als Flowchart

2.2.5 Abhängigkeiten

Netzwerktopologie

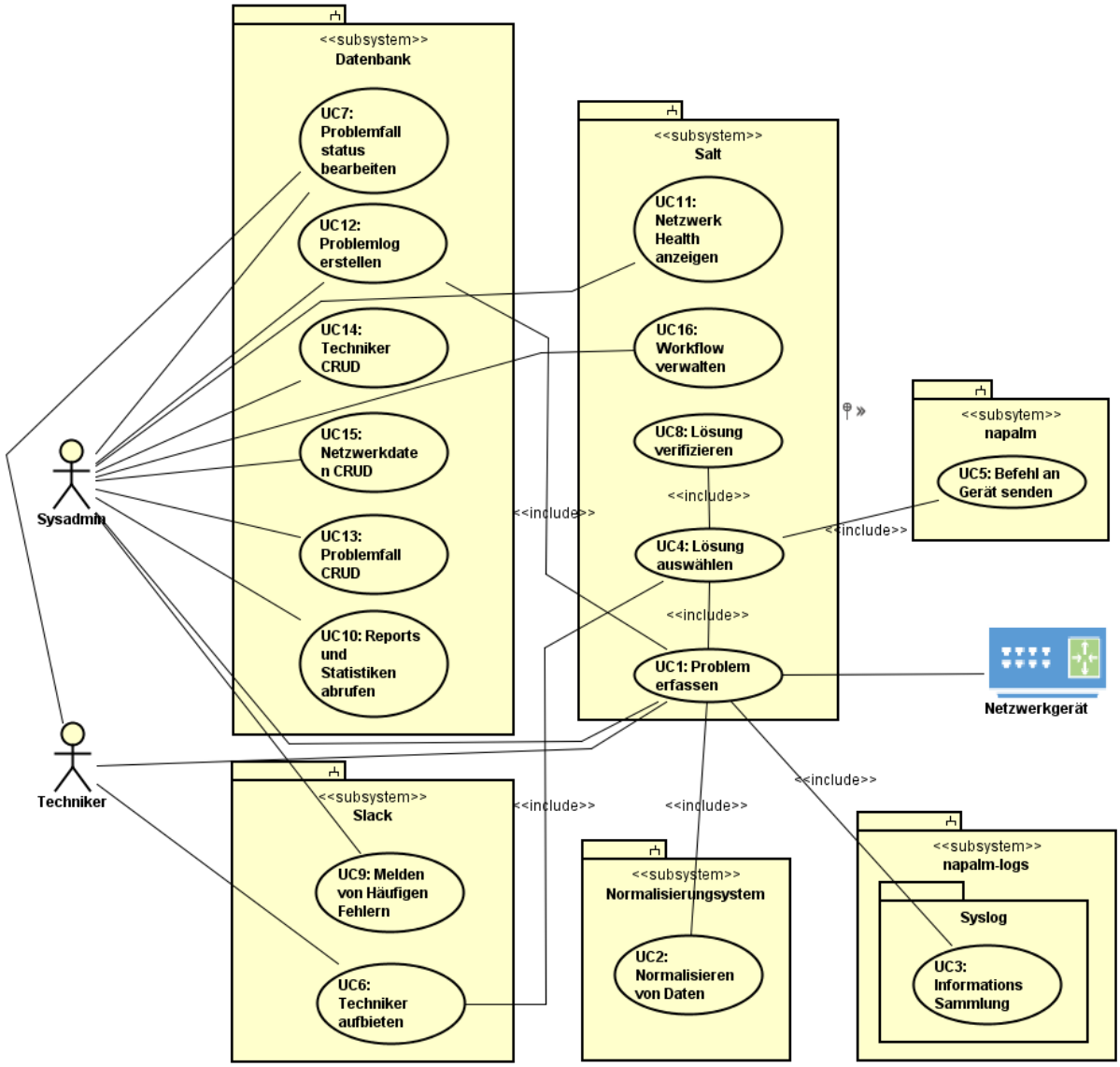
Die Netzwerktopologie inkl. Vorhandene Geräte, IP-Adressen und Interfaces müssen bekannt sein. Die Topologie ist idealerweise in einer Datenbank abgelegt und kann jederzeit vom System abgefragt werden.

2.3 Use Cases

2.3.1 Aktoren

Aktoren	Beschreibung, Ziele
Systemadministrator	Der Systemadministrator ist verantwortlich dafür dass das Automated Troubleshooting System funktioniert und angeschlossene Netzwerk einwandfrei ist.
Techniker	Der Techniker wird aufgerufen um aufgetretene Fehler im Netzwerk zu beheben, er kann auch Systemadministrator sein.

2.3.2 Use Case Diagramm



2.3.3 Use Cases im Brief Format

UC1: Problemfall erfassen

Das System erhält von Syslog oder napalm Informationen zu einem Problem im Netzwerk, es erstellt dazu einen neuen Problemfall und beginnt mit dem Troubleshooting Prozess. Alternative kann ein

Systemadministrator oder ein Techniker manuell ein Problemfall erstellen, etwa um eine Überprüfung des Systems durchzuführen.

UC2: Normalisieren von Daten

Damit mehrere Quellen zum Erhalten von Daten verwendet werden können müssen diese normalisiert werden. Zum Normalisieren von NETCONF Daten bietet sich Pyangbind an.

UC3: Informationen von Geräten sammeln

Das System muss in der Lage sein Informationen von einem Gerät im Netzwerk abzufragen und diese dem richtigen Problemfall zuzuordnen oder einen neuen Problemfall erstellen. Falls ein Problem nicht genau identifiziert werden konnte muss das System von einem Gerät auf Anfrage noch weitere Daten erhalten. Dazu wird hier napalm und napalm-logs verwendet.

UC4: Lösung auswählen

Aufgrund der gesammelten Informationen entscheidet sich das System für eine Lösung um das ermittelte Problem zu beheben. Dazu werden die Salt-reactors und das Salt-engine verwendet. Falls das System das Problem nicht selber lösen kann und ein Techniker aufgeboten wird, muss das System zudem entscheiden wie dringend sich dieser dem Problem annehmen muss.

UC5: Befehl an Geräte senden

Während dem Troubleshooting Prozess muss es dem System möglich sein Befehlen an ein Gerät, welches im Netzwerk liegt, zu senden. Beispiele dazu sind System und Konfigurationsdaten anfordern um das Problem zu identifizieren oder ein ausführbares Skript senden welches das Problem beheben sollte. Diese Kommunikation zu den Netzwerkgeräten wird über napalm geregelt.

UC6: Techniker aufbieten

Falls es dem System nicht möglich ist ein aufgetauchtes Problem automatisch zu Identifizieren oder zu lösen muss ein Techniker aufgeboten werden, um sich diesem anzunehmen. Die Entscheidung für eine Mitteilung des Technikers kommt von Salt, die eigentliche Nachricht wird von Slack gesendet.

Warten bis Techniker verfügbar

Es kann vorkommen dass kein Techniker verfügbar ist und einer aufgeboten werden muss der zurzeit nicht bei der Arbeit ist, z.B. bei einem Pikettdienst, falls das Problem keine hohe Priorität vom System erhalten hat, könnte man warten bis wieder ein Techniker verfügbar ist.

UC7: Problemfall Status bearbeiten

Ein Techniker oder ein Systemadministrator der an einem Problemfall arbeitet, muss in der Lage sein Informationen zu diesem zu bearbeiten, etwa falls er das Problem abgeschlossen hat.

UC8: Lösung verifizieren

Nachdem eine Lösung für ein Problem ausgeführt wurde, muss das System überprüfen ob das Problem nun gelöst wurde und somit den Fall abschliessen kann. Die Verifikation wird von Salt ausgeführt, wobei auf Informationen der betroffenen Geräte zugegriffen werden muss und somit wie bei Use Case 3 und 5 napalm und napalm-logs zur Kommunikation eingesetzt wird.

UC9: Melden von häufigen Fehlern

Falls oft dieselben Fehler auftreten, kann dies auf ein grundlegendes Problem im Netzwerk oder in dessen Aufbau hinweisen, das System muss in der Lage sein diese zu identifizieren und einen Systemadministrator zu benachrichtigen. Detektier werden solche Häufigkeiten über die geloggten

Daten aus der Datenbank oder direkt im Salt. Der Systemadministrator wird über Slack auf dies aufmerksam gemacht.

UC10: Reports und Statistiken abrufen

Ein Systemadministrator möchte, aus Informationen von aufgetretenen Problemfällen, Statistiken und Reports zu einzelnen Geräten oder dem kompletten Netzwerk abrufen. Dazu soll er direkt eine Datenbank Abfrage machen können.

UC11: Network-Health anzeigen

Um schnell eine Übersicht über den Zustand des Netzwerks zu erhalten kann ein Techniker den Network Health Counter anzeigen lassen, welcher Ihm Informationen über den Zustand des Netzwerks gibt. Dies kann man direkt über Salt lösen.

UC12: Problemlog erstellen

Sobald ein Troubleshooting-Prozess durch ein Event ausgelöst wird, wird in der DB ein Problemfall mit allen relevanten Details erstellt. Der Problemfall wird während des Troubleshooting-Prozesses aktualisiert und am Ende als abgeschlossen markiert.

UC13: Techniker CRUD

Ein Systemadministrator erstellt Techniker und verwaltet diese, damit sie bei einem Problemfall aufgeboten werden können.

UC14: Problemfall CRUD

Ein Systemadministrator kann einzelne Informationen zu Problemfällen lesen und bearbeiten. Er kann die Problemfälle auch selber erstellen oder löschen.

UC15: Netzwerkdaten CRUD

Ein Systemadministrator unterhält die relevanten Netzwerkdaten für das System in einem externen File welches in Salt eingelesen wird.

UC16: Workflow verwalten CRUD

Falls neue Probleme auftauchen oder weitere Probleme automatisch gelöst werden sollen, muss ein Systemadministrator dazu einen neuen Workflow im System erstellen. Existierende Workflows sollen einsehbar sein und wenn nötig soll es auch möglich seine diese zu editieren. Falls ein Workflow nichtmehr benötigt wird, etwa weil das betreffende Protokoll nichtmehr verwendet wird, soll man einen Workflow auch löschen können.

2.4 Anforderungen an das System

Nr.	Anforderung	Priorität	Abh.
1	Das System dient als zentrale Sammelstelle für Problemfälle.	1	-
2	Das System kann Problemfälle verarbeiten.	1	1
3	Das System leitet aus Events ab, was für ein Problem besteht und wandelt die Meldung in einen Problemfall um.	1	2
4	Der Problemfall beinhaltet Informationen zum betroffenen Gerät und der Art des aufgetretenen Problems.	1	3
5	Das System unterteilt die Problemfälle in unterschiedliche Klassen, abhängig von der Art des aufgetretenen Problems.	1	4
6	Das System ergreift aufgrund eines Problemfalls Massnahmen zur Fehlerbehebung.	1	5
7	Das System erkennt, wenn es einen Problemfall nicht selbst behandeln kann und informiert einen Techniker.	1	6
8	Das System überprüft die getroffenen Massnahmen.	1	6
9	Das System informiert bei nicht erfolgreicher Problemfallbehandlung einen Techniker.	1	8
10	Das System erstellt nach der Problemfallbehandlung einen Logeintrag	1	6,8
11	Ein Logeintrag beinhaltet alle Informationen eines Problemfalls und die getroffenen Massnahmen zur Problemfallbehandlung.	1	10
12	Das System erkennt, wenn im Netzwerk übermässig viele Kollisionen stattfinden, ermittelt den Grund dafür und ergreift passende Massnahmen.	3	3
13	Das System erkennt, dass das Problem im OSPF Protokoll besteht und behebt diesen.	1	3
14	Das System erkennt, dass das Problem im BGP Protokoll besteht und behebt diesen.	3	3
15	Das System erkennt, dass das Problem im STP Protokoll besteht und behebt diesen.	2	3
16	Das System erkennt, dass das Problem im EIGRP Protokoll besteht und behebt diesen.	3	3
17	Das System erkennt, dass das Problem im CDP Protokoll besteht und behebt diesen.	3	3
18	Das System erkennt eine Fehlerhafte VLAN Konfiguration und behebt diese.	2	3
19	Das System erkennt dass das Problem in der Link-Aggregation besteht und behebt dieses.	2	3
20	Das System erkennt, dass ein Gerät nicht mehr am Netz angeschlossen ist und informiert einen Techniker.	2	3
21	Das System erkennt wenn ein Link aufgrund eines physischen Problems nicht mehr verfügbar ist und informiert einen Techniker.	3	3
22	Das System erkennt, wenn ein Interface runterfährt und ermittelt den Grund dafür.	1	3
23	Das System erkennt, wenn viele CRC-Fehler auf einem Link vorkommen, ermittelt den Grund dafür und ergreift passende Massnahmen.	3	3
24	Das System erkennt Flapping MAC-Adressen, ermittelt den Grund dafür und ergreift passende Massnahmen.	3	3
25	Das System erkennt langsame Links und behebt diese (Streaming Telemetry).	3	3
26	Das System erlaubt das Erstellen von Thresholds für langsame Links.	3	23

27	Das System erkennt Routing-Loops und behebt diese.	2	3
28	Das System erkennt suboptimales Routing und behebt dieses.	3	3
29	Das System erkennt wenn 2 Events den gleichen Problemfall beschreiben und erstellt nur einen Problemfall für diese Events.	3	2

(Cisco, 2015) (Cisco Learning Network, 2017) (ExamCollection, 2017)

2.5 Nicht funktionale Anforderungen

Die folgenden NFRs wurden der ISO 9126 Spezifikation entnommen (ISO, 2017).

2.5.1 Modifizierbarkeit

Das System kann einfach mit weiteren Funktionen ergänzt werden. So kann, wenn das System im produktiven Umfeld eingesetzt wird, einfach durch Module ergänzt werden, die die häufigsten Fehler im Netz automatisch beheben können.
Dies wird durch eine Development-Anleitung sichergestellt (siehe Kapitel 4.5).

2.5.2 Interoperabilität

- Das System ist unabhängig von der im Netzwerk eingesetzten Hardware.
- Das System ist unabhängig von den im Netzwerk eingesetzten Networking-Protokollen.

Es muss dabei nicht alle Protokolle und Geräte unterstützen, soll von deren Existenz aber nicht negativ beeinflusst werden.
Dies ist durch die Verwendung von Salt, napalm und napalm-logs sichergestellt (siehe Kapitel 3).

2.5.3 Richtigkeit

Eingriffe des Systems im Netzwerk belassen das System im schlimmsten Fall (→ Das System kann den Fehler nicht selbst beheben) im gleichen Zustand wie vor dem Eingriff.
Napalm garantiert, dass die Zugriffe atomar und idempotent sind, somit ist auch dies sichergestellt.

3. Architektur

Änderungsgeschichte

Datum	Version	Änderung	Autor
11.10.2017	1.0	Erstellung des Dokuments	rj, nv
14.10.2017	1.1	Stackstorm, Ansible, Logging	rj
15.10.2017	1.2	Salt, NAPALM, NAPALM-logs Pyangbind, ZeroMQ, Fazit etc.	nv
16.10.2017	1.3	Allgemeine Bearbeitung vor Abgabe	nv, rj
08.12.2017	2.0	Überarbeitung nach der Construction	rj
13.12.2017	2.1	Ergänzung nach Construction	nv

3.1 Einführung

3.1.1 Zweck

Abbildung 16: Problemlösungs-Workflow

Dieses Flowchart wurde zu Beginn des Projekts erstellt, um die Anforderungen an das System sichtbar zu machen. Nun gilt es für die einzelnen Anforderungen geeignet Tools auszuwählen, damit der gesamte Funktionsumfang abgedeckt ist. Des Weiteren ist es wichtig, dass die Architektur als Ganzes stimmt, d.h. dass die einzelnen Aspekte des Gesamtsystems ohne grössere Probleme miteinander zusammenarbeiten können.
Zu diesem Zweck werden in Kapitel 3.3 die einzelnen Tools verglichen bzw. beschrieben. In Kapitel 3.4 folgt eine Übersicht über die Architektur und eine Beschreibung, wie die einzelnen Tools die in Abbildung 17 dargelegten Arbeitsschritte übernehmen können.

3.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung vom automatisierten Troubleshooting und während der nachfolgenden Bachelorarbeit.

3.2 Toolanalyse

3.2.1 Sensoren

Die Sensoren des Systems sammeln Informationen im Netz und lösen Events aus, sobald gewisse Ereignisse eintreten (z.B. ein Interface fährt runter). Nachfolgend werden die Sensoren des Systems kurz beschrieben, welche in der Domainanalyse genauer analysiert wurden.

Syslog

Syslog wird auf jedem Netzwerkgerät manuell eingerichtet. Es muss genau definiert werden, welche Meldungen geschickt werden und welche nicht. Syslog schickt dann die definierten Meldungen an OATS. Da die Syslog Meldungen keine definierte Form haben, müssen sie vom System selber in Modellform transformiert werden.

NETCONF

Der Subscription <> Notification Mechanismus von NETCONF erlaubt es, dass auf gewisse Events mit einer asynchronen Nachricht an eine zentrale Sammelstelle reagiert wird. Anders als bei Syslog sind die Daten schon in XML, JSON oder YANG modelliert. In erster Linie sollen die Sensoren mit Syslog umgesetzt werden NETCONF wird hier nur als mögliche Erweiterung des Systems erwähnt.

NMS –Network Monitoring Systems

Der Ansatz von „Streaming-Telemetry“ wird im Netzwerkumfeld immer wichtiger. Ein NMS sammelt kontinuierlich Daten im Netz und kontrolliert so den Zustand des Netzes. Weicht ein Wert (z.B. Durchsatz auf einem Link) signifikant von den im Netz sonst üblichen Werten ab, so wird ein Event ausgelöst und an das AT-System übermittelt. Wie auch NETCONF sind NMS als eine mögliche Erweiterung des Systems gedacht.

3.2.2 Aktoren

NAPALM

Napalm (Network Automation and Programmability Abstraction Layer with Multivendor support) ist eine Open Source Python-Library die Funktionen implementiert, welche es einem erlauben mit unterschiedlichen Netzwerkgeräten zu interagieren. So ermöglicht napalm den Verbindungsaufbau zu Geräten, die Manipulation deren Konfiguration und das Sammeln von Informationen.
Im Rahmen des Projekts wird napalm vor allem als API zum Sammeln von Informationen benutzt werden, um Entscheidungen über den weiteren Verlauf des Troubleshootings treffen zu können (Barroso, Welcome to the NAPALM Documentation, 2017).

Unterstützte Betriebssysteme

Napalm unterstützt folgende Betriebsssteme:

- Arista EOS
- Cisco IOS
- Cisco IOS-XR
- Cisco NX-OS
- Fortinet Fortios
- IBM
- Juniper JunOS
- Mikrotik RouterOS
- Palo Alto NOS
- Pluribus
- Vyos

Somit kann eine Vielzahl von Geräten in einem Netzwerkumfeld angesprochen werden, was die Flexibilität unseres Systems um ein Vielfaches erhöht.

Unterstützte Methoden

	EOS	FORTIOS	IOS	IOSXR	JUNOS	NXOS
connection_tests	{}	{}	{}	{}	{}	{}
get_arp_table	✓	✗	✓	✓	✓	✓
get_bgp_config	✓	✗	✗	✓	✓	✗
get_bgp_neighbors	✓	✓	✓	✓	✓	✓
get_bgp_neighbors_detail	✓	✗	✗	✓	✓	✗
get_config	✓	✓	✓	✓	✓	✓
get_environment	✓	✓	✓	✓	✓	✗
get_facts	✓	✓	✓	✓	✓	✓
get_firewall_policies	✗	✓	✗	✗	✗	✗
get_interfaces	✓	✓	✓	✓	✓	✓
get_interfaces_counters	✓	✓	✓	✓	✓	✗
get_interfaces_ip	✓	✗	✓	✓	✓	✓
get_lldp_neighbors	✓	✗	✓	✓	✓	✓
get_lldp_neighbors_detail	✓	✗	✓	✓	☠	✓
get_mac_address_table	✓	✗	✓	✓	☠	✓
get_network_instances	✓	✗	✗	✗	✓	✗
get_ntp_peers	✗	✗	✗	✓	✓	✓
get_ntp_servers	✓	✗	✓	✓	✓	✓
get_ntp_stats	✓	✗	✓	✓	✓	✓
get_optics	✓	✗	✓	✗	✓	✗
get_probes_config	✗	✗	✗	✓	✓	✗
get_probes_results	✗	✗	✗	✓	✓	✗
get_route_to	✓	✗	✗	✓	✓	✗
get_snmp_information	✓	✗	✓	✓	✓	✓
get_users	✓	✗	✗	✓	✓	✓
is_alive	✓	✓	✓	✓	✓	✓
ping	✓	✗	✓	✗	✓	✗
post_connection_tests	{}	{}	{}	{}	{}	{}
pre_connection_tests	{}	{}	{}	{}	{}	{}
traceroute	✓	✗	✓	✓	✓	✓

Abbildung 17: Auszug der NAPALM Supportmatrix

Wie man in der napalm-Supportmatrix sieht, werden viele in unserer Arbeit nützliche Funktionen out of the box unterstützt. Sollte eine Funktion fehlen, ist es möglich neue Funktionen selber zu schreiben und hinzuzufügen.

Funktionsweise

Hier der Auszug eines `get_interfaces` calls:


```
{
    "Ethernet2": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1502731278.4344141,
        "is_up": true,
        "mac_address": "08:00:27:3D:83:34",
        "speed": 0
    },
    "Management1": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1502731294.598835,
        "is_up": true,
        "mac_address": "08:00:27:7D:44:C1",
        "speed": 1000
    },
    "Ethernet1": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1502731278.4342606,
        "is_up": true,
        "mac_address": "08:00:27:E6:4C:E9",
        "speed": 0
    }
}
```


Abbildung 18: get_interfaces call

Die Ergebnisse des Calls werden als JSON geliefert, sodass diese ohne Probleme im Code weiterverarbeitet werden können.

NAPALM-Logs

Da Syslog als Sensor im Netz eingesetzt wird, ist eine offene Frage, wie die plaintext Nachrichten von Syslog in eine im Code nutzbare Form gebracht werden können. Diese Lücke füllt napalm-logs (Ulinic, 2017).

Napalm-logs ist eine Open Source Software, die als Daemon läuft und auf Syslog Meldungen horcht und diese in eine herstellerunabhängige Form bringt. Anders als napalm wird napalm-logs nur benutzt um Nachrichten zu empfangen.
Die Nachrichten werden entweder direkt durch TCP/UDP empfangen oder im Zusammenhang mit einem Messaging-System wie ZeroMQ entgegengenommen und verschickt.

Funktionsweise:
Folgende 2 Abbildungen stellen 2 Syslog-Meldung von 2 unterschiedlichen Geräteherstellern dar:

```
<149>Jun 21 14:03:12  vmx01 rpd[2902]: BGP_PREFIX_THRESH_EXCEEDED: 192.
168.140.254 (External AS 4230): Configured maximum prefix-limit thresho
ld(140) exceeded for inet4-unicast nlri: 141 (instance master)
```

Abbildung 21: Syslog Meldung eines Juniper-Devices

```
<149>2647599: xrv01 RP/0/RSP1/CPU0:Mar 28 15:08:30.941 UTC: bgp[1051]:
%ROUTING-BGP-5-MAXPFX : No. of IPv4 Unicast prefixes received from 192.
168.140.254 has reached 94106, max 12500
```

Abbildung 19: Syslog-Meldung eines Cisco-Devices

Wie man sieht haben beide Nachrichten völlig unterschiedliche Formen, obwohl sie eine Meldung für dasselbe Problem repräsentieren.
Napalm-logs nimmt diese Nachrichten und bringt sie in dieselbe Form (siehe rechts).

- **error:** von napalm-logs vordefinierter eindeutiger Identifier (UID)
- **facility & severity:** geben Schweregrad und Ort des Fehlers an (siehe auch Domainanalyse)
- **host:** Hostname des Netzwerkgeräts
- **ip:** IP-Adresse des Geräts
- **os:** Betriebssystem des Geräts
- **timestamp:** UTC timestamp zum Zeitpunkt der Generierung der Nachricht
- **yang_message:** YANG-Modell welches benutzt wurde um die Nachricht zu formen (siehe auch Domainanalyse)


```
{
  "error": "BGP_PREFIX_THRESH_EXCEEDED",
  "facility": 18,
  "host": "vmx01",
  "ip": "127.0.0.1",
  "os": "junos",
  "severity": 5,
  "timestamp": 1498053792,
  "yang_message": {
      "bgp": {
          "neighbors": {
              "neighbor": {
                  "192.168.140.254": {
                      "afi_safis": {
                          "afi_safi": {
                              "inet4": {
                                  "ipv4_unicast": {
                                      "prefix_limit": {
                                          "state": {
                                              "max_prefixes": 140
                                          }
                                      }
                                  },
                                  "state": {
                                      "prefixes": {
                                          "received": 141
                                      }
                                  }
                              }
                          }
                      },
                      "state": {
                          "peer_as": "4230"
                      }
                  }
              }
          }
      }
  },
  "yang_model": "openconfig-bgp"
}
```


Abbildung 20: Resultierende Meldung

Pyangbind

Pyangbind ist ein Pyang-Plugin, welches ein YANG-Datenmodell entgegennimmt und daraus Python Klassen generiert.
Pyangbind ermöglicht:

- Das Kreieren von neuen YANG-Dateninstanzen direkt in Python.
- Das Laden von Dateninstanzen von externen Quellen (z.B. napalm-logs).
- Das Serialisieren von Python-Objekten in ein Format welches gespeichert oder über das Netzwerk verschickt werden kann. Die untersützten Formate sind zurzeit OpenConfig JSON oder IETF JSON.

Pyangbind wurde als Bestandteil von OpenConfig entwickelt, d.h. es wurde entwickelt um gut mit den OpenConfig YANG-Modellen (welche auch napalm-logs benutzt) zusammenzuarbeiten.

3.2.3 Core Tools

Für den Core unserer Software haben wir drei Tools genauer betrachtet und diese verglichen, damit wir schlussendlich dasjenige verwenden welches unseren Anforderungen am besten entspricht. Die Tools sollen in der Lage sein erhaltene Events zu interpretieren und weiterzuverarbeiten, basierend auf vordefinierten Regeln (Shaw, 2017).

Ansible

Ansible ist das erste Tool welches wir genauer betrachtet haben. Um Ansible zu verwenden muss man es bloss auf einem Linux Gerät installieren und man kann loslegen. Dies funktioniert da Ansible keine Daemons und keine Datenbanken verwendet. Damit Ansible weiss mit welchen Geräten es kommunizieren soll reicht es aus ein statisches File mit Adressen zu hinterlegen, man kann aber auch ein dynamisches Discovery Modul wie etwa Amazon EC2 oder Openstack verwenden damit man relevante VMs basierend auf API calls finden kann. Sobald man ein Geräteinventar erstellt hat kann man damit beginnen Variablen und Parameter zu definieren welche von Ansible Playbooks verwendet werden können. Der Zugriff auf Ansible selbst läuft über die Kommandozeile.

Abbildung 22: Ansible

Playbooks sind Sequenzen von Ansible Modulen welchen man auf einem Host ausführen möchte, geschrieben sind sie in YAML. Ansible spricht Server über SSH an da dies bei allen Endgeräten vorhanden ist. Sobald Ansible verbunden ist wird der Python Code übertragen und ausgeführt, nach dem ausführen wird er wieder gelöscht.

Architektur

Die Ansible-Applikation läuft auf dem Gerät auf welchem man es installiert hat, falls etwas auf einem anderen Host ausgeführt werden muss, wird eine Verbindung erstellt und nach dem ausführen wieder beendet. Es existiert also keine Server-Client Architektur, sondern man muss Aufgaben auf dem Hauptgerät parallelisieren.
Ansible gibt es auch in der Enterprise „Tower“-Variante, dies erlaubtes einem Ansible auf mehrere Server zu verbinden und somit stark zu skalieren.
Da Ansible nur auf einem Gerät läuft ist es sehr einfach auf eine neuere Version zu upgraden, ohne dass auf einem entfernten Gerät Kompatibilitätsprobleme auftauchen.

Enterprise Version

Mit der Verwendung der Enterprise Version kann man anstelle der Kommandozeile zusätzlich auch ein Web Interface verwenden. Dies erlaubt es einem ein Notifikation System anzubinden sowie Aufträge selber zu planen.

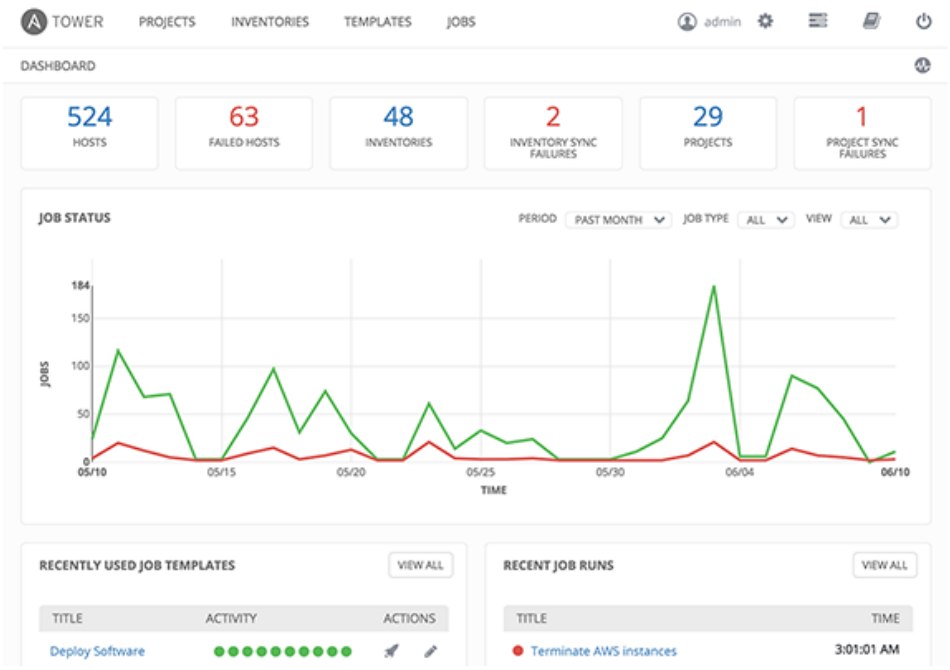


Abbildung 23: Ansible Tower

Mit dem Interface kann man das aktivieren von Playbooks direkt automatisieren und die gewünschten Ziel VMs aus der aus der Inventarliste auswählen. Diese Zusatzfunktionen kommen aber mit einem Preis.

SELF-SUPPORT	STANDARD	PREMIUM
PERFECT FOR SMALL DEPLOYMENTS	FOR ENTERPRISE IT OPERATIONS	FOR MISSION-CRITICAL DEVOPS
Support not included Maintenance & upgrades Limited features	**8x5 support** Maintenance & upgrades Enterprise features	**24x7 support** Maintenance & upgrades Enterprise features
Up to 100 nodes $5,000/year	**Up to 100 nodes** $10,000/year $13,000/year with Ansible Engine	**Up to 100 nodes** $14,000/year $17,500/year with Ansible Engine Ansible Networking support available
Up to 250 nodes $10,000/year	**Over 100 nodes** Contact us for pricing	**Over 100 nodes** Contact us for pricing
Over 250 nodes Standard & Premium only	View production support service level	View production support service level

Abbildung 24: Ansible Support

Ansible Support

Ansible wird von allen wichtigsten Netzwerkgerätehändler und Plattformen unterstützt. Ansible erlaubt es einem die Konfiguration eines Netzwerk Stacks von einem System aus zu automatisieren durch das Ausführen von plattformspezifischen Modulen und Playbooks. Es erlaubt einem ausserdem das existierende Netzwerk zu testen und zu validieren, der Informationsgewinnungsprozess kann selber implementiert werden um Informationen über eine existierende Konfiguration zu gewinnen. Ansible unterstützt die Programmierbaren Cisco Plattformen, Arista, Juniper aber auch andere Anbieter. Die Unterstützung kommt von den Händlern selbst und nicht von der Community.

Vor- und Nachteile

Vorteile:

- Schneller und einfacher Start durch minimalen Installationsprozess
- Verschiedene Dokumentationen und Module sind von der Community bereits vorhanden
- Netzwerk Module von den Anbietern selber Verfügbar

Nachteile:

- Kein System um Events zu triggern oder auf diese zu reagieren
- Viele Funktionen nur in der Premium Version verfügbar

Stackstorm

Abbildung 25: Stackstorm

Stackstorm ist eine erweiterbare Regel- und Ausführungs-Engine, es erlaubt diverse Konfigurationen und Einstellungen. Dies erlaubt es eine sehr flexible auf die jeweiligen Anforderungen angepasste Lösung zu erstellen. Man kann Regeln erstellen welche es erlauben auf spezifische Events zu reagieren und Befehle oder komplexe Workflows zu deployen. Stackstorm bietet ausserdem einen Service für ChatOps, was es erlaubt mit Chat Plattformen wie etwa Slack direkt auf Funktionen zuzugreifen.

Architektur

Die Architektur von Stackstorm besteht aus sechs Hauptteilen, welche hier kurz beschrieben werden.

Sensoren sind Python Plugins welche Events erhalten oder aktiv Ausschau halten. Falls ein Event in einem externen System vorkommt werden diese von den Stackstorm Sensoren registriert und ein Trigger wird ins System eingegeben.
Ein Trigger ist die Repräsentation eines externen Events, es gibt zwei Kategorien, generische Triggers wie etwa ein ablaufender Timer und integrierte Triggers wie etwa ein Monitoring Alarm oder ein JIRA Issue Update. Neue Trigger können über die Sensoren definiert werden.
Sobald ein Trigger im System erscheint wird eine Action ausgeführt, hier gibt es drei Kategorien, generische Actions wie ein SSH Aufruf oder ein REST call, die integrierten Actions wie Openstack oder Puppet und schliesslich die komplett benutzerdefinierten Actions. Hier werden ebenfalls Python Plugins oder einfache Scripts verwendet. Actions können entweder manuell über die Kommandozeile oder die API gestartet werden oder basierend auf einer Regel.
Rules sind dazu da die vorkommenden Trigger einer Aktion oder einem Workflow zuzuordnen, basierend auf erfüllten Kriterien wie Payload oder Tags.
Workflows sind zusammensetzte Aktionen in komplexe Operationen, hier werden Reihenfolge, Kriterien und Datenübertragungen definiert. Oft gibt es für Automations-Operationen mehrere Schritte und dementsprechend auch mehrere Aktionen. Ein Workflow kann wie eine einfache Action aus der Bibliothek manuell gestartet werden oder sie werden durch eine Regel gestartet.
Packs können sowohl Trigger und Aktionen aber auch Regeln und Workflows beinhalten, sie erlauben ein einfaches Managen und Teilen von benutzerdefiniertem Stackstorm-Inhalt. Die Anzahl an Benutzerdefinierten Packs welche über die Stackstorm Community erhältlich sind wächst stetig.
Ausgeführte Aktionen können manuell oder automatisiert in einem sogenannten Audit Trail gespeichert werden. Dort werden die kompletten Daten, einer Aktion, über Kontext und Resultat erfasst.

Stackstorm verwendet eine Anzahl von verschiedenen Services wie etwa ein Rabbit Message Queue und eine MongoDB zur Datenspeicherung.

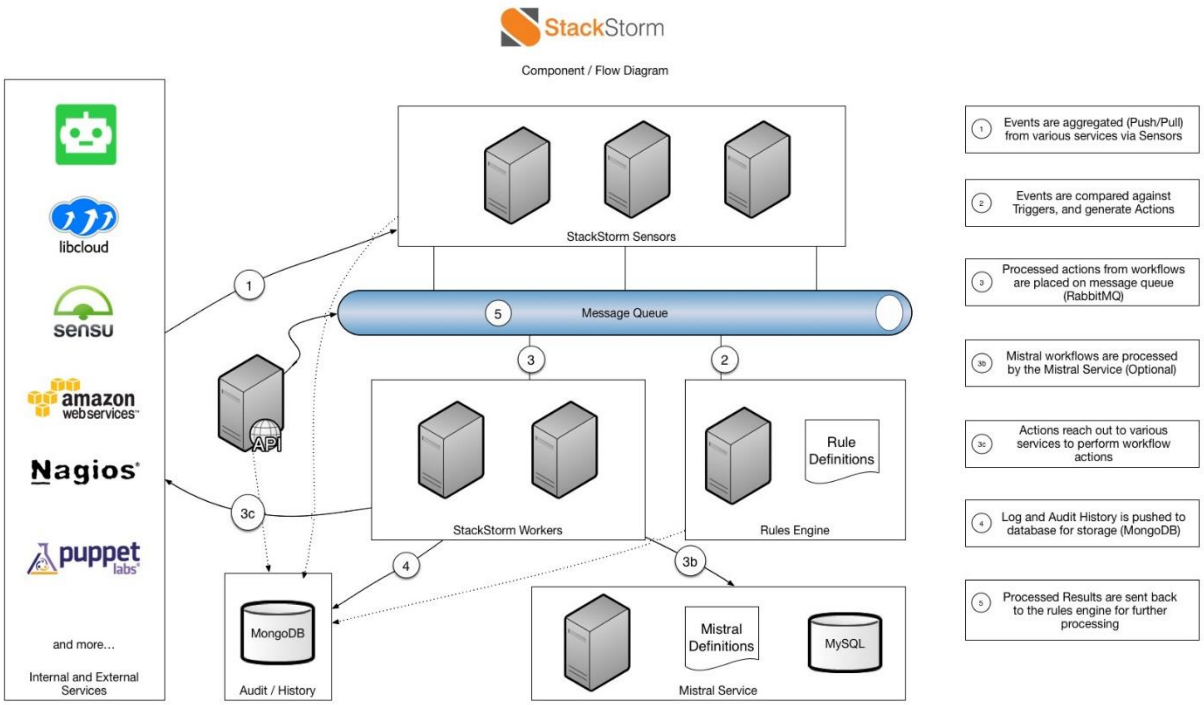


Abbildung 26: Stackstorm Infrakstruktur

Stackstorm wurde nicht dazu erstellt um Endpunkte zu konfigurieren oder mit ihnen zu kommunizieren, dazu bietet es Pakete für Salt, Puppet oder Ansible an. Dies erlaubt es Funktionen und Vorteile von anderen Plattformen zu nutzen und mit Stackstorm zu kombinieren. Logs und die Audit History werden über MongoDB in einer skalierbaren Umgebung abgelegt.
Die Angebotenen Services sind als Http- oder Restful-API konzipiert, man kann einen einzelnen Server oder mehrere verwenden, je nach Anforderungen.

Stackstorm gibt es in seiner Originalfassung ohne Erweiterungen, es muss also alles zusätzlich installiert werden. Dies macht Stackstorm an sich schlanker und die Abhängigkeiten minimal. Wenn mittels Stackstorm entwickelt wird kann man selber erstellte Pakete zusammen in eine einzelne Version bündeln. Jedes Paket von Stackstorm hat seine eigenen Requirements-Files, es erlaubt ausserdem beim Upgrade von Paketen die Konfigurationen zu behalten.
Die Dokumentation liegt ausserhalb des eigentlichen Modul Codes was beim Einsatz von fremdentwickelten Paketen zu Schwierigkeiten führen kann, da diese eventuell nicht sauber beschrieben sind.
Stackstorm ist ein Apache-2 lizensiertes Open Source Software Produkt, gehostet wird es auf Github

Stackstorm Support

Brocade VLX und SDX bieten sehr guten Support für Stackstorm, durch einen verbinden des Supports mit der napalm library, eine Cross-Plattform Abstraktion Python Paket, bieten sie auch Support für Cisco, Juniper, Arista und weitere Anbieter. Man kann zudem die napalm Integration und Routen und Interfaces sowie BGP Peering zu konfigurieren, es werden dazu noch weitere Features geboten.

Vor- und Nachteile

Vorteile:

- Gratis Web Interface ist einfach zu bedienen auch mit sehr wenig Python oder Programmierkenntnissen.
- Direkte Einbindung von ChatOps, Unterstützt diverse Chat Plattformen wie etwa Slack
- Sehr mächtige Regel Engine

Nachteile:

- Weniger Extensions im Vergleich zu anderen Tools, Systeme und API werden eventuell nicht unterstützt
- Workflows sind sehr oft trial and error da der YAQL query syntax nicht sehr gut dokumentiert ist
- Stackstorm benötigt zur Kommunikation mit Geräten noch weitere Plugins

Salt

Im Gegensatz zu den beiden anderen Tools wurde Salt (das Open Source Produkt von Saltstack) als verteiltes System konzipiert. Mittels sogenannten «minions» werden auf (Remote-)Objekten Befehle und Abfragen durchgeführt. Die minions können dabei einzeln oder aufgrund von Selektionskriterien angesprochen werden.

Abbildung 27: Saltstack

Salt ist in der Lage Remote-Objekte direkt zu konfigurieren. Dies passiert mittels sogenannten «states». Mittels den states wird sichergestellt, dass bestimmte Pakete oder Services installiert und im Betrieb sind.

Salt besteht unter anderem aus folgenden Komponenten:

- **master:** Der Server auf dem die Corefunktionalität ausgeführt wird und mit den minions kommuniziert.
- **minions:** Objekte die eine verkleinerte Version von Salt ausführen um die lokale Ausführung von Befehlen und Kommunikation mit dem master zu ermöglichen. Minions werden mittels «grains» oder «pillars» angesprochen die weiter unten erklärt werden.
- **engines:** Salt-Engines sind extern laufende Prozesse auf denen z.B. auf Events gehorcht werden kann.

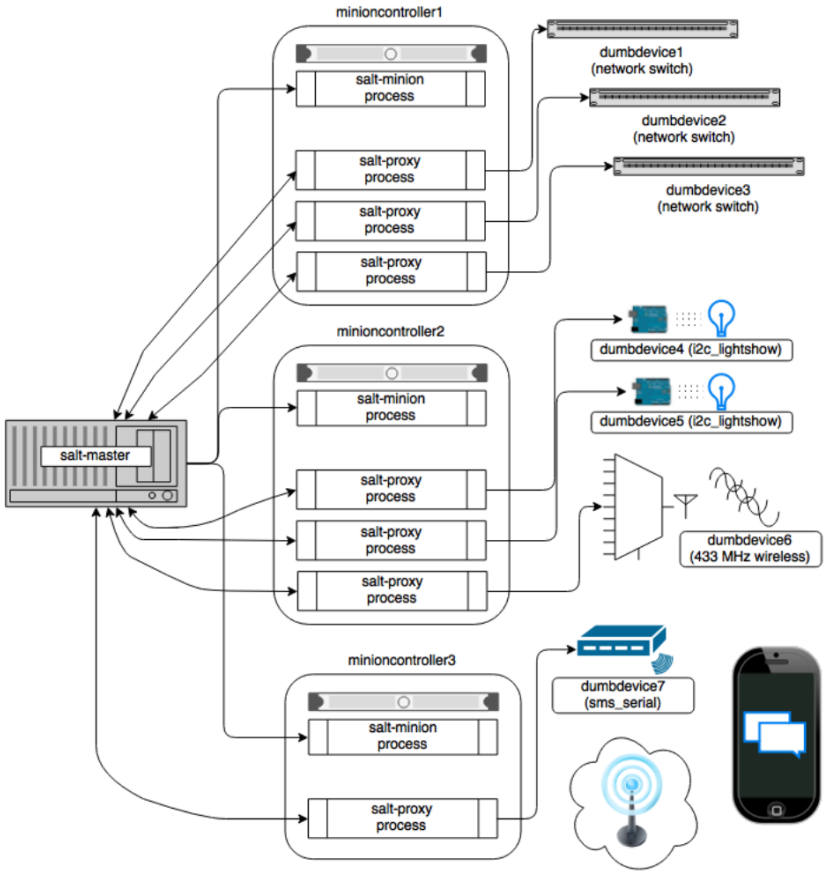


Abbildung 28: Salt Funktionsweise

- **states / formulas:** Dateien die YAML oder Templating-Daten enthalten um die minions zu konfigurieren.
- **grains**: Grains bieten ein Interface um «Körner» von Informationen von Geräten zu sammeln (grain = Korn). So können Informationen über das OS, IP-Adresse, domain name etc. des darunterliegenden Systems gewonnen werden.
- **pillars**: Pillars bieten ein globales Interface, um Werte an die minions zu verteilen. Sie können in mehreren Formaten gespeichert werden (JSON, YAML und andere) und können lokal oder remote liegen.
- **mine**: Die Saltmine sammelt Daten der minions und speichert sie auf dem master. Der Unterschied zu den grains besteht darin, dass die Daten aktueller sind, da grains nicht oft aktualisiert werden. Die mines können so in einem regelmässigen Intervall Daten von den Minions abfragen, um einen aktuellen Status über das gesamte Netzwerk zu erlangen.

Architektur

Salt liegt eine «hub and spoke»-Architektur zugrunde. Es wäre auch ein Deployment mit mehreren masters möglich. Ein master kann aber auf mehrere Tausend Knoten skaliert werden, sodass ein master in den meisten Fällen ausreichend ist.
Der master enthält state-Dateien, welche als Baumstruktur aufgesetzt sind. So können einfach verschiedene Servergruppen angesprochen und konfiguriert werden.

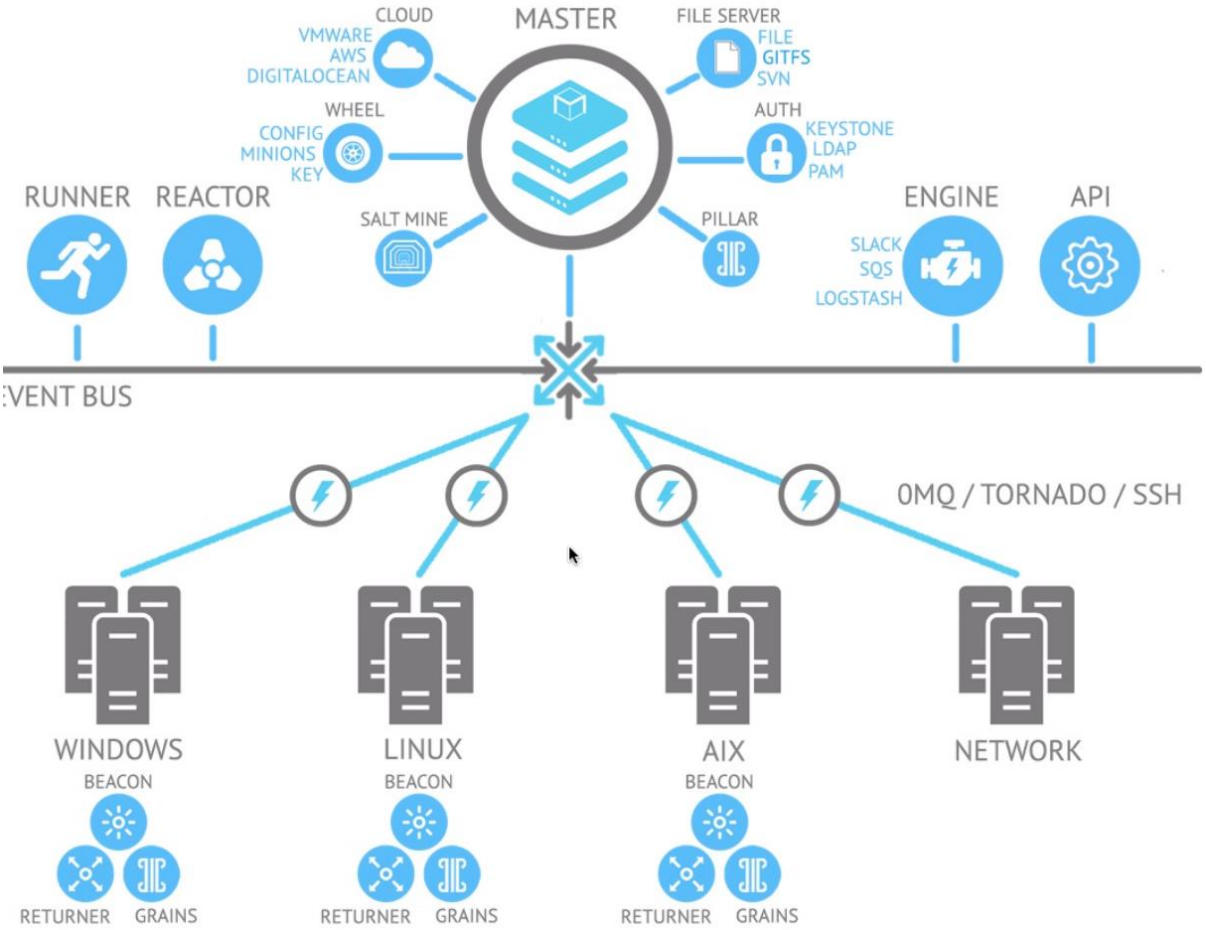


Abbildung 29: Salt Architektur

Das Event-basierte System von Salt benutzt «beacons», ähnlich zum Sensor- und Trigger-System von Stackstorm. Die beacons leiten Events in den Nachrichtenbus, welche anschliessend vom «reactor» auf dem master verarbeitet werden. Der reactor enthält eine Regel-Engine mit welcher ein state oder ein direkter Befehl ausgeführt werden kann. Da die beacons auf den minions laufen, ist die Übermittlung von Events an den master sehr einfach. Nur Salt ist in der Lage dies so zu tun, da Ansible und Stackstorm agentless (→keine minions) sind.
Die Regel-Engine vom reactor ist verglichen mit demjenigen von Stackstorm eher begrenzt. Vor kurzem wurde allerdings der «thorium complex reactor» veröffentlicht (allerdings noch in experimentellem Status), der in der Lage ist komplexere Abläufe zu definieren. Sollte Salt im Rahmen des Projekts eingesetzt werden, müsste ermittelt werden ob sich der einsatz des thorium reactor lohnen würde.

Erweiterbarkeit

Dies ist eine der Stärken von Salt. Der einfachste Weg zum Erweitern von Salt ist es, neue state (wie ein Objekt konfiguriert werden soll) oder execution (Code um mit einem System zu interagieren) modules zu schreiben.
Salt-Packs sind Pakete die in Salt integriert werden können. So existieren z.B. Packs für napalm und napalm-logs (und viele andere). Auch können eigene Packs entwickelt und veröffentlich werden.

Salt Support

Zurzeit werden folgende Netzwerk-Betriebssysteme unterstützt:

- JunOS (Juniper)
- NXOS (Cisco)
- Cisco NSO (Cisco NETCONF)
- NAPALM

Dadurch dass napalm unterstützt wird, werden indirekt auch alle Funktionen unterstützt, die napalm anbietet. Dies erweitert den Funktionsumfang beträchtlich.

Vor- und Nachteile

Vorteile:

- Ermöglicht agent- und auch agentless-Deployment
- Sehr gut dokumentiert (inkl. Einfacher Tutorials mit vorbereiteter Testumgebung)
- Einfache Event-Übermittlung dank der verteilten minions
- Die Regel-Engine ermöglicht die Ausführung von Workflows
- Sehr hohe Performance da als Eventbus ZeroMQ benutzt wird
- Durch die gute Erweiterbarkeit einfach an andere APIs und Datenbanken anhängbar

Nachteile:

- Weniger gute Regel-Engine als Stackstorm (wenn nicht thorium reactor benutzt wird)
- Bei Benutzung von thorium reactor: Noch nicht ausführlich getestet und dokumentiert
- Höherer Verwaltungsaufwand durch die verteilten minions

Fazit

Ansible ist das Tool, mit dem am schnellsten etwas umgesetzt werden könnte. Allerdings ist es das einzige Tool, das keine Event-Driven Architektur unterstützt. Im Grunde genommen ist Ansible nichts anderes als eine Skriptsprache. In der Enterprise Version wäre Ansible wohl ein wenig mächtiger, doch wir wollen uns im Projekt auf Open Source Lösungen konzentrieren. Deswegen haben wir uns gegen Ansible entschieden.

Die Entscheidung bei Salt vs. Stackstorm ist uns um einiges schwieriger gefallen. In einer perfekten Welt, würden wir uns noch eine Woche Zeit nehmen, um eine Testumgebung aufzusetzen und beide Tools darauf ausgiebig vergleichen. Da uns diese Zeit fehlt, müssen wir zu diesem Zeitpunkt eine Entscheidung treffen.
Für Stackstorm spricht die mächtigere Regel-Engine zum Umsetzen der Workflows. Bei Salt ist die verteilte master – minion Architektur ein klarer Vorteil. Diese würde die Kommunikation zwischen unserem System und den Endgeräten massiv vereinfachen. Auch scheint der thorium reactor in etwa gleich mächtig zu sein wie die Stackstorm-Engine.
Beide Tools unterstützen napalm was eine gute API Grundlage für beide Systeme bietet, allerdings ist Salt durch das Modul-System einfacher erweiterbar (eine unserer nichtfunktionalen Anforderungen). Salt hat im Gegensatz zu Stackstorm kein integriertes Nachrichten System und auch kein GUI (in der Open Source Version). Als Nachrichten System kann man mit Salt aber problemlos Slack anhängen.
Um einen besseren Einblick in die beiden Umgebungen zu erhalten, haben wir zusätzlich die Testversionen angeschaut. Auf der Salt Website konnte eine einfache Testumgebung mit einem Master und 2 minions runtergeladen werden. So konnte an einem Beispiel direkt die Funktionsweise

ausprobiert werden. Bei Stackstorm wurde zwar auch ein Tutorial angeboten, allerdings ohne Testumgebung, was den Nutzen des Tutorials doch schmälert.
Der Hauptgrund der gegen Stackstorm spricht ist, dass es nicht in der Lage ist Gerätekonfigurationen selbst zu managen. Für diesen Punkt müsste auf ein zusätzliches Tool (Salt, Puppet) zurückgegriffen werden.

	Salt	**Stackstorm**
Aufbau und Architektur	Master-minion, stark skalierbar, agentenbasiert	Erweiterbare Regeln- und Ausführungs-Engine
Punkte (Max 10)	8	8
Kommunikation zu Endpunkten	Über minions oder proxies	Nicht dafür entworfen, benötigt zusätzliches Tool
Punkte (Max 10)	10	3
Erweiterbarkeit	Erweiterbar durch Module, Abhängigkeiten nicht sauber, sehr umfassend	Erweiterbar durch Packs, sauber individuell installiert, weniger umfassend
Punkte (Max 10)	8	8
Benötigtes Know-How und verfügbare Quellen	Mittlere Länge zum Einrichten viel verfügbare Dokumentation	Mittlere Länge zum Einrichten wenig verfügbare Dokumentation
Punkte (Max 10)	9	6
ChatOps	Muss selbst hinzugefügt werden	Bereits Integriert
Punkte (Max 5)	2	5
Tutorial	Angeboten mit Testumgebung	Angeboten ohne Testumgebung
Punkte (Max 5)	5	2
Benutzeroberfläche	Kein UI	Gratis default WebUI
Punkte (Max 5)	0	5
Total (Max. 55)	**42**	**37**

Salt liegt in der Punkten Anzahl leicht vorne, jedoch ist zu beachten, dass wir uns hier auf externe Quellen basieren und unsere eigene Erfahrungen nur auf Artikeln und den kurzen Tutorials basieren. Die Punkte sind zudem unsere eigene Interpretation der verfügbaren Informationen.
Unsere Hauptpunkte sind jeweils mit zehn Punkten gewertet, in diesen alleine schliesst Salt deutlich besser ab, vor allem das die Kommunikation zu den Endpunkten direkt stattfinden kann, bei Stackstorm muss hier eine weiteres Tool hinzugenommen werden, was zusätzliche Arbeit und Zeit kosten würde.

In Betrachtung dieser Punkte sind wir zum Schluss gekommen, dass das Herzstück unseres Systems mit Salt umgesetzt werden soll.

3.2.4 Logging

Behandelte und abgeschlossene Fälle und Probleme sollen aufbewahrt werden um eventuell später analysiert zu werden. Damit man alle Daten an einem Ort hat bietet es sich an diese direkt in einer Datenbank abzuspeichern.
In unserem Projekt wollen wir zwar ein Logging einbinden jedoch nicht das Hauptaugenmerk darauf setzen. Die wichtigsten Kriterien für uns waren hier Open Source Software und eine einfache Einbindung in das Tool welches die Core Funktionalität unserer Software bietet. Unsere Datenbank muss ausserdem über Python angesprochen werden können.

MongoDB

Eine weitverbreitete NOSQL Datenbank Lösung bietet MongoDB, Stackstorm hat diese bereits direkt integriert und Salt bietet eine Einbindung für diese an. Da Ansible an sich kein Log erstellt müsste hier MongoDB manuell integriert werden (MongoDB, 2017).

MongoDB ist eine Dokumentbasierte Datenbank die skalierfähig und flexibel ist. Sie bietet Indexierungs- und Abfrage-Features an. Daten werden in einem Dokumentenformat welches JSON sehr ähnlich ist gespeichert. Das heisst dass die Felder in jedem Dokument nach Wunsch variieren können und die Datenstruktur nachträglich noch geändert werden kann. Das Dokumentenmodell bildet Objekte aus dem Applikationscode auf die Dokumente ab.
MongoDB ist eine Verteilte Datenbank, welche hohe Verfügbarkeit, horizontale Skalierung und Geographische Verteilung bietet. MongoDB ist zudem frei verfügbar und Open Source.

PostgreSQL

Postgres ist ein Open Source objektrelationales Datenbank Management System.
Features und Funktionen von Postgres sind Multi-Version Concurrency Control, Asynchrone Replikation, nested Transaktionen, Hot Backups, ein Query Planer und Optimierer. Postgres bietet hohe Skalierbarkeit in Bezug auf Datenmenge und Benutzer. Unter anderem läuft es auf
Postgres bietet über ein Framework die Möglichkeit eigene Datentypen zu definieren und zu erstellen, sowie Funktionen und Operationen um für die ein passendes Verhalten zu implementieren. Auch gibt es bereits Erweiterungen für diverse Netzwerkadresstypen, was für uns besonders interessant ist.

MariaDB

MariaDB ist eine relationale Datenbank mit einer erweiterbaren Architektur um verschiedene Use Cases zu unterstützen, gelöst wird dies über ein Storage Engine System und mit zusätzlichen Plugins. MariaDB ist Open Source und die Standarddatenbank auf diversen Linux Distributionen. Es bietet die Möglichkeit Daten über JSON Dokumente, aber auch per SQL Interface zu speichern und abzurufen. MariaDB gilt als schnell und robust sowie sehr gut skalierbar. Einbindung einer Datenbank über Python funktioniert mit dem MySQL Modul.

Verwendung

Alle Datenbanklösungen sind mit unseren Anforderungen kompatibel, da wir hier aber nicht zu viel Zeit mit der Integrierung der Datenbank an die restlichen Tools verlieren wollen, haben wir uns entschieden MongoDB zu verwenden, da Salt und auch Stackstorm bereits Support und Dokumentation für eine Einbindung an die MongoDB bietet.

3.2.5 ZeroMQ

Da ZeroMQ der Default für das Transportieren von Meldungen in Salt ist, ist es nur logisch dass ZeroMQ auch im Umfeld des Systems eingesetzt wird. ZeroMQ ist eine auf hohe Leistung ausgelegte, asynchrone messaging-library für verteilte Software Systeme. ZeroMQ bietet eine Message-queue aber im Gegensatz zu anderen ähnlichen Systemen, benötigt es keinen message-broker. Die ZeroMQ-API wurde so entworfen, dass sie ähnlich wie Berkeley-Sockets funktioniert(siehe Abbildung 31) (Wikipedia, 2017).

```
import zmq
import time
context = zmq.Context()

subscriber = context.socket (zmq.SUB)
subscriber.connect ("tcp://192.168.55.112:5556")
subscriber.connect ("tcp://192.168.55.201:7721")
subscriber.setsockopt (zmq.SUBSCRIBE, "NASDAQ")

publisher = context.socket (zmq.PUB)
publisher.bind ("ipc://nasdaq-feed")

while True:
    message = subscriber.recv()
    publisher.send (message)
```

Abbildung 30: ZeroMQ Code Beispiel

ZeroMQ unterstützt folgende grundlegende Messaging-Patterns:

- **Request-reply:**
  Verbindet ein Set von Publishern mit einem Set von Services
- **Publish-subscribe:**
  Verbindet ein Set von Publishern mit einem Set von Subscribern.
- **Push-pull (pipeline)**
  Verbindet Knoten in einem *fan-out/fan-in* pattern. Erlaubt mehrere Schritte und auch Loops.
- **Exclusive Pair:**
  Verbindet direkt zwei Sockets miteinander.

Für den Use Case in unserem System scheint vor allem der publish-subscribe Mechanismus nützlich. So sollte es möglich sein den Event-Bus sauber umzusetzen.

3.2.6 Slack

Falls ein Problem auftaucht welches nur von einem Techniker vor Ort gelöst werden kann, muss man diesen aufbieten. Diese Mitteilungen werden wir anfangs über Slack aus senden, mit der Option später noch weitere Mitteilungsoptionen anzubieten (Slack, 2017).

Der Name Slack ist ein Akronym für “Searchable Log of All Conversation and Knowledge” und die Software ist im Grunde eine Chat App. Slacks Features beinhalten, persistente Chatrooms auch Channels genannt, welche nach Thema organisiert sind, zusätzlich gibt es auch private Gruppen und Direktnachrichten. Der Inhalt von Slack kann durchsucht werden und beinhaltet Files, Konversationen und Benutzer. Slack bietet zudem eine Integration von diversen Software und Service von Drittanbietern, sowie Support für Community erstellte Integrationen von eigenen Projekten.

Slack ist für uns besonders interessant da es neben dem Mitteilungsdienst zusätzlich noch die Option bietet, einfache Workflows über Nutzereingaben oder Buttons zu starten. Zudem kann man Slack mit einem Modul einfach in eine Salt Installation einbinden.

3.3 Architektur: Konzept

Nun da alle Tools identifiziert wurden, stellt sich die Frage, wie diese miteinander arbeiten. Zu diesem Zweck wurde folgende Systemübersicht erstellt:

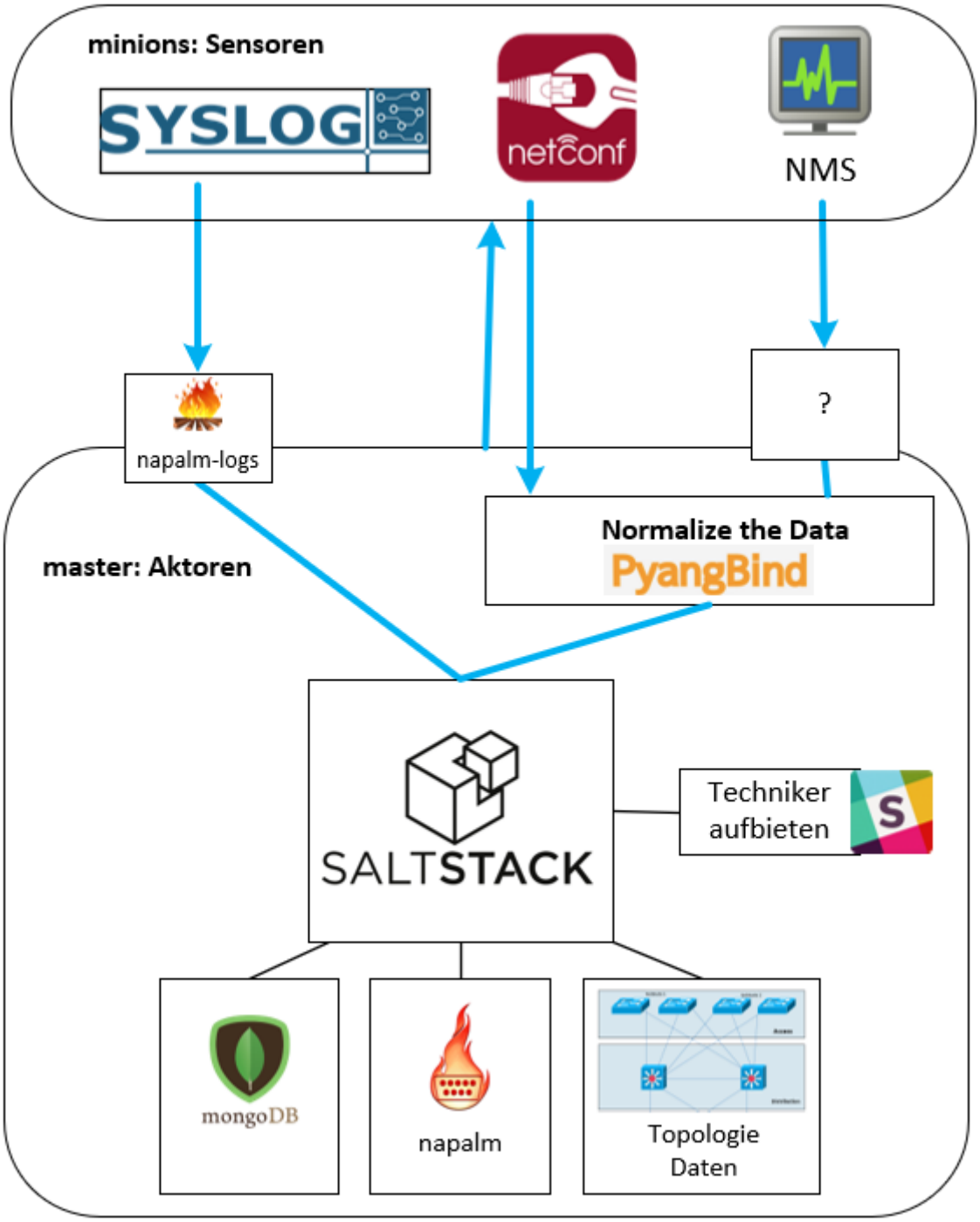


Abbildung 31: Architektur: Konzept

In Kapitel 3.2.1 wurde aufgezeigt, welche Arbeitsschritte für das automatische Troubleshooting nötig sind. Nachfolgend wird aufgezeigt, welche Tools für welche Aufgaben zuständig sind.

3.3.1 Erfassung

Generell sind hierfür die Sensoren zuständig. Die Grundidee ist es, dafür am Anfang nur Syslog zu verwenden. Die Erweiterung auf NETCONF und NMS (Network Monitoring Systems) sind als mögliche

Modifizierungen für das System gedacht. NETCONF arbeitet bereits mit dem YANG-Modell, bei den NMS müsste das System so erweitert werden, dass die Informationen in ein YANG-Modell übersetzt werden. Die von NETCONF und den NMS erhaltenen YANG-Daten können anschliessend mittels Pyangbind in Python-Klassen transformiert werden.

Syslog, welches direkt auf dem Netzwerkgerät läuft, erkennt Events und übermittelt diese per UDP an das in Salt integrierte Tool napalm-logs. Es muss noch ermittelt werden, wo napalm-logs am Ende laufen wird. Es ist möglich napalm-logs entweder auf jedem minion laufen zu lassen, oder dass man es nur einmal auf dem master depylot.
Napalm-logs nimmt diese Syslog Meldungen, transformiert sie in das YANG-Format und schickt sie wieder über den Event-Bus an den Salt-master.

3.3.2 Identifizierung

Die Identifizierung unterscheidet sich von Fall zu Fall und hängt vom in Gang gesetzten Workflow ab. Der Urheber des Fehlers ist dank Salt schnell ermittelt, allerdings könnte die Ursache auf einem anderen Gerät liegen.
Der Salt-master nimmt das Event entgegen. Nun kommt das Salt-Reactor-System zum Einsatz, welches uns erlaubt auf Events mit Actions zu reagieren.

```
reactor:                            # Master config section "reactor"

  - 'salt/minion/*/start':          # Match tag "salt/minion/*/start"
    - /srv/reactor/start.sls        # Things to do when a minion starts
    - /srv/reactor/monitor.sls      # Other things to do

  - 'salt/cloud/*/destroyed':       # Globs can be used to match tags
    - /srv/reactor/destroy/*.sls    # Globs can be used to match file names

  - 'myco/custom/event/tag':        # React to custom event tags
    - salt://reactor/mycustom.sls   # Reactor files can come from the salt fileserver
```

Abbildung 32: Salt-reactor YAML-File

Wie man sieht, ist es möglich auf Custom-Event-Tags mit Custom-Actions zu reagieren. So können die komplexen Workflows zum Identifizieren von Problemen angestossen werden. Diese könnten z.B. extern als Python-Script abgelegt sein und beinhalten unter anderem die in der Domainanalyse beschriebenen Technologien (Ping, Traceroute etc.). Für das Abfragen der Geräte wird das in Salt integrierte napalm benutzt. Dank napalm ist es möglich diese Abfragen herstelleragnostisch zu halten.
Das System muss zur effizienten Problemlösung allzeit über die Netzwerktopologie Bescheid wissen. Zu diesem Zweck ist extern die Topologie auf z.B. einem YAML-File abgelegt.
Erkennt das System, dass es den Problemfall nicht selbst beheben kann, wird eine Slack-Nachricht an den Techniker mit allen nötigen Details ausgelöst.

3.3.3 Klassifizierung

Syslog gibt jedem Event einen Severity-Level. Der Severity-Level wird von uns in einem File angegeben. Der Severity-Level muss gut gewählt sein, da der Techniker, bei einem Problem, welches nicht direkt durch das System behoben werden kann, wissen muss, wie dringend die Behebung des Problems ist (siehe Domainanalyse für mehr Details zu den Severity-Levels).

3.3.4 Behebung

Wurde das Problem durch unseren Workflow identifiziert und als vom System behandelbar klassifiziert, so wird eine Problemlösungsroutine gestartet. Diese wird auch Teil des Salt-Reactors sein (siehe oben) und kann extern mit Python-Scripts erledigt werden.
Die Zugriffe macht der master mithilfe des direkt integrierten Tools napalm. Napalm ist somit für das Abfragen und das Manipulieren der Netzwerkgeräte zuständig.
Kann das System das Problem nicht beheben, wird der Techniker mit einer Slack-Nachricht notifiziert. Die Nachricht beinhaltet Art des Problems, den Severity-Level und womöglich noch andere benötigte Informationen.

3.3.5 Überprüfung

Hier müssen wieder direkte Abfragen auf die Netzwerkgeräte mittels napalm durchgeführt werden. Der Prozess ist ähnlich zu demjenigen in Abschnitt 3.2 und variiert von Fall zu Fall.

3.3.6 Reporting/Logging

Bei erfolgreicher Überprüfung wird ein Eintrag in der MongoDB mit allen nötigen Details erstellt.
Bei nicht erfolgreicher Überprüfung wird ein Techniker über Slack benachrichtigt.

3.4 Architektur: Umsetzung

Wie es in einem Projekt üblich ist, gibt es immer Unterschiede zwischen dem was konzeptioniert wurde und dem was schliesslich umgesetzt wurde. In diesem Kapitel wird auf diese Unterschiede eingegangen. Zu diesem Zweck wurde auch eine neue Version der Systemübersicht erstellt:

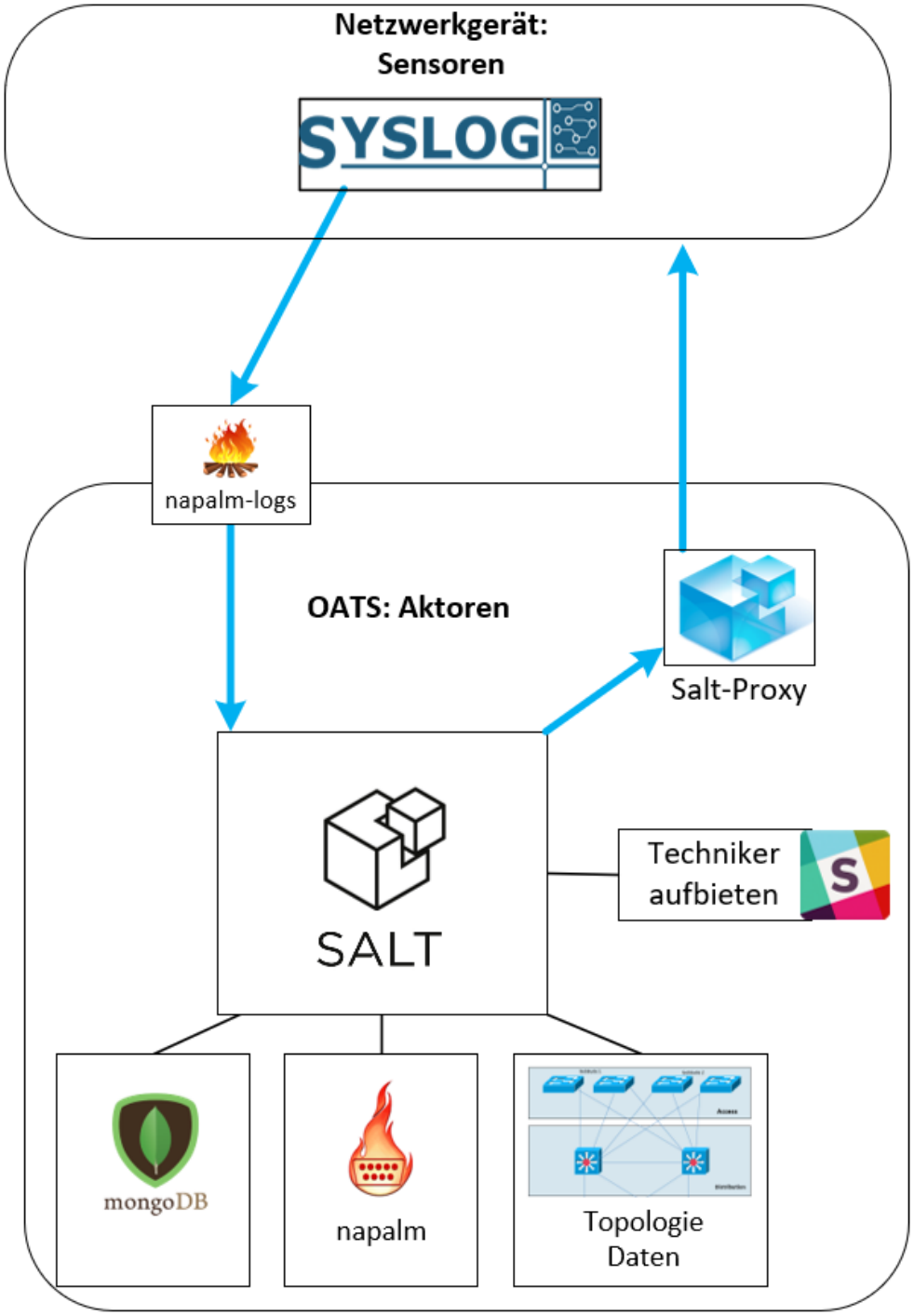


Abbildung 33: Architektur: Umsetzung

3.4.1 Erfassung

Wie bereits erwähnt, wurden NETCONF und potenzielle NMS weggelassen.
Napalm-logs wird nur einmal auf dem master deployt (ist aber nicht in Salt integriert) und leitet die Meldungen nicht direkt an den Salt-master weiter, sondern werden in einem Zwischenschritt vom Oatsclient (siehe 3.6.1) verarbeitet. Der Oatsclient erstellt und versendet die Salt-Events.

3.4.2 Identifizierung

Die Identifizierung von Probleme geschieht an 2 Stellen.
Die erste Stelle ist der Oatsclient, der in der Lage ist Meldungen von napalm-logs zu aggregieren, sodass schon früh erkannt werden kann, wo das Problem liegt.
Die zweite Stelle ist, wie schon beschrieben, innerhalb von Salt. Dort wurde ein Python-file geschrieben in welchem mittels Salt und Napalm Informationen gesammelt werden (z.B. per Pings oder Datenbankabfragen). Zu diesem Zweck ist die Netzwerktopologie in der eigenen MongoDB abgelegt.

3.4.3 Klassifizierung

Bei der Umsetzung wurde bemerkt, dass eine direkte Klassifizierung anhand der Severity-Levels keinen Sinn macht, da z.B. eine Interface-Down Meldung einen Level von 5 hat. Eine Interface-Down Meldung könnte aber bedeuten, dass nur das Interface runtergefahren wurde oder es könnte bedeuten, dass das Gerät nicht mehr online ist.
Hier müsste für die Zukunft ein neues Konzept entwickelt werden.
Die Klassifizierung in „vom System behandelbar“ und „Techniker benötigt“ geschieht fliessend im gleichen Python-file wie die Identifizierung & Behebung. D.h. es wird nur eine Slack Meldung erstellt, wenn das Problem nicht behoben werden konnte.

3.4.4 Behebung

Identifizierung & Behebung wird im gleichen Python-file durchgeführt (falls keine Aggregation stattfindet).
Dabei werden die Befehle nicht direkt vom Salt-master an die Geräte übermittelt, sondern werden vom Salt-master an den jeweiligen Salt-proxy (für jedes Netzwerkgerät existiert ein Salt-proxy) geschickt, welcher den Befehl an das Gerät weiterleitet.
Der Ablauf falls Aggregation durchgeführt wurde ist der gleiche wie wenn keine stattgefunden hat. Nur ist es nicht nötig vor der Behebung des Problems in Python noch Informationen zu sammeln. So muss schliesslich weniger Code geschrieben werden.

3.4.5 Überprüfung

Auch die Überprüfung geschieht direkt im gleichen Python-file. Dafür wird z.B. per Ping kontrolliert, ob ein Gerät welches vorher über ein bestimmtes Interface nicht erreichbar war, nun erreichbar ist.

3.4.6 Reporting/Logging

Reporting/Logging geschieht per MongoDB. Allerdings wird nicht erst am Ende ein Log erstellt. Der Eintrag wird erstellt, sobald das Event im Oatsclient ausgelöst wird und wird bei jedem zusätzlich nötigem Schritt um die neuen Informationen ergänzt. So ist es möglich, falls eine Slack Meldung nötig ist, genaue Informationen zu den bisher getätigten Massnahmen zu liefern.

3.5 Abläufe

Ein beispielhafter Ablauf in OATS könnte folgendermassen aussehen:

1) Ein Interface fährt runter, das betroffene Gerät löst eine Syslog-Meldung an das System aus.
2) Die Syslog-Meldung wird von napalm-logs empfangen. Napalm-logs wandelt die Meldung wie in Kapitel 3.2.2 beschrieben, um.
3) Napalm-logs übermittelt die in YANG modellierte Meldung über ZeroMq an den Oatsclient (siehe Kapitel 3.6.1).
4) Im Oatsclient werden die Meldungen auf 2 Arten verarbeitet:
   a) Der Oatsclient verarbeitet die Daten und sendet ein Salt Event mit den relevanten Daten an den Salt-master weiter (über den Salt-Event-Bus).
   b) Der Oatsclient aggregiert eine Meldung, indem er für eine gewisse Zeit alle Meldungen der gleichen Art zählt. Dadurch kann der Ursprung des Problems eingegrenzt werden. Anschliessend konstruiert er ein Salt-Event und verschickt dieses an den Salt-master.
5) Auf dem Salt-master wird anhand Events entschieden, was für Informationen noch gesammelt werden müssen.
6) Mittels napalm und Salt werden die benötigten Informationen gesammelt.
7) Anhand der gesammelten Informationen wird entschieden, welche Massnahmen ergriffen werden müssen um das Problem zu beheben.
   a) Kann das System das Problem selbst beheben, so werden Die nötigen Schritte dafür eingeleitet.
   b) Kann das System das Problem selbst nicht beheben, so benachrichtigt es einen Techniker
8) Bei selbst eingeleiteten Massnahmen überprüft das System, ob der Fehler behoben wurde. Die Überprüfung geschieht wiederum mittels napalm und Salt. Konnte das Problem nicht behoben werden, so wird ein Techniker per Slack beauftragt.

3.6 Logische Architektur

Die Architektur von OATS besteht aus fünf verschiedenen Komponenten. Diese sind die Implementierung von Napalm Logs, die Implementierung von Salt, die Hilfsfunktionen für den automatischen Datenbankzugriff, die Klassen für den manuellen Datenbankzugriff und die Datenbank selbst.
Die Datenbank und der genaue Aufbau dieser werden unter dem Kapitel Datenspeicherung genauer erklärt weshalb hier nicht weiter darauf eingegangen wird.

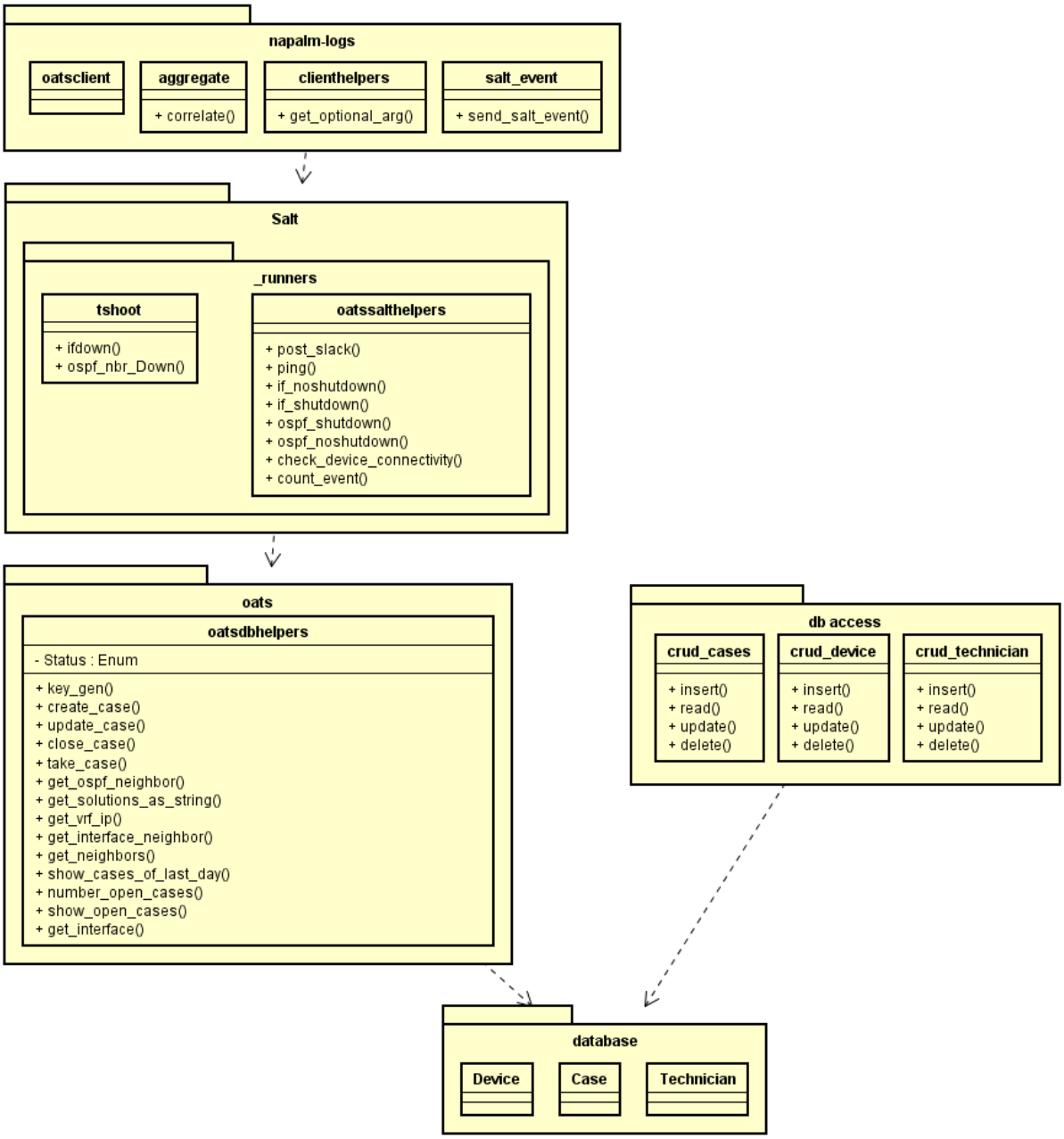


Abbildung 34: Architekturdiagramm

3.6.1 Napalm-logs

Die Implementierung von napalm-logs besteht aus vier Klassen, dem Oatsclient, Aggregate, Clienthelpers und Salt_event.

Anmerkung: Das Tool napalm-logs läuft als eigenständiger Prozess, der die Syslog Meldungen entgegennimmt, die Daten modelliert und sie an den Oatsclient schickt (siehe auch Kapitel 3.7: Deployment). Was hier beschrieben wird ist die Implementierung eines Clients der diese Daten verarbeitet.

Klasse	Beschreibung
oatsclient	Der Oatsclient ist ein Listener der auf einen definierten Port hört und sobald er eine Meldung von napalm-logs erhält, den Troubleshooting Prozess startet. Der Oatsclient verwendet dafür die anderen files innerhalb dieses Ordners.
aggregate	Falls ein Event von mehreren Geräten kommen könnte und je nach dem einen anderen Workflow durchlaufen muss um das Problem zu beheben, wird es aggregiert. Dabei wird für eine gewisse Zeitdauer auf eine Meldung gehorcht und für jedes Vorkommen ein Zähler inkrementiert. Anhand des Zählers wird dann entschieden welches Salt-Event an den Salt-master geschickt werden soll.
clienthelpers	Der oatsclient erhält die Syslog Meldungen im Yang format, in welchem diverse Informationen gespeichert werden, die nicht alle relevant sind zur Lösung des Problems. Die clienthelpers beinhalten die get_optional_arg Funktion welche es uns erlaubt die gewünschten Informationen zu extrahieren.
Salt_event	Das salt_event file beinhaltet die send_salt_event Funktion welche die gegebenen Daten an den Salt-Event-Bus sendet um dort verarbeitet zu werden.

3.6.2 Salt

Bei der Implementation von Salt haben wir die Workflows in den Salt-runners entwickelt, diese werden direkt auf dem Salt-Master ausgeführt. Die files sind tshoot welche den Grundsätzlichen Troubleshooting Workflow beinhaltet sowie die Implementationen der spezifischen Lösungen. Sowie die oatssalthelpers, welche Hilfsfunktionen beinhaltet welche wir benötigen, um Informationen für den Troubleshooting Prozess zu erhalten.

Klasse	Beschreibung
tshoot	Die tshoot Klasse beinhaltet zwei Funktionen die ifdown und die ospf_nbr_down, diese sind die Workflows um jeweils ein Problem zu lösen.
oatssalthelpers	Diese Klasse beinhaltet diverse Funktionen um von Salt und den Workflows auf das Netzwerk zuzugreifen. post_slack ermöglicht es uns eine Nachricht auf Slack zu schreiben um etwa einen Techniker anzufragen. Mit der Ping Funktion kann man von einem Gerät ein anderes anpingen und die Erfolgsrate erfahren. if_noshutdown bzw. if_shutdown sind Funktionen welche ein spezifisches Interface auf einem Netzwerkgerät hoch- und runterfahren. Die OSPF Funktionen erlauben es uns den OSPF Prozess auf einem OSPF verwendenden Netzwerkgerät hoch bzw. runterzufahren. Die Funktion check_device_connectivity überprüft ob es Netzwerkgerät von irgendeinem anderen Netzwerkgerät im Netz noch erreichbar ist. Schliesslich gibt es noch die count_event Funktion, welche asynchron einen Listener aktiviert, welcher zählt wie viele Male in einem definierten Zeitraum ein Event erhalten wird.

3.6.3 Oatsdbhelpers

Die Oatsdbhelpers sind Hilfsfunktionen welche wir in einem Workflow verwenden um auf die Datenbank zuzugreifen. Hier gibt es nur ein file, die oatsdbhelpers. In diesem gibt es einen ENUM in welcher zur Abbildung des Workflowstatus dient.

Funktion	Beschreibung
key_gen	Als Case ID generieren wir mithilfe dieser Funktion eine eigene ID die aus einer Variablen Anzahl Base64 Stellen besteht, wir haben hier mit zwölfstelligen IDs gearbeitet.
Create_Case	Am Anfang des Workflows wird mit dieser Funktion einen Case in der Datenbank angelegt.
Update_case	Falls sich der Status ändert oder weitere Informationen zu einem Case aus einem Workflow in die Datenbank eingetragen werden müssen ist dies mit dieser Funktion möglich.
Close_case	Falls ein Problem mit den definierten Workflows gelöst werden konnte werden diese mit der close_case funktion abgeschlossen.
Take_case	Falls ein Techniker einen Fall manuell lösen muss kann er diesen mit der take_case funktionen als unter Bearbeitung markieren.
Get_ospf_neighbor	Um die OSPF Nachbaren eines Netzwerkgeräts zu erfahren kann man diese Funktion verwenden.
Get_solutions_as_string	Um zu sehen welche Lösungsschritte unternommen wurden um ein Fall zu lösen kann man diese Funktion aufrufen.
Get_vrf_ip	Damit man auf ein Gerät über seine VRF IP zugreifen kann, ermöglicht es diese Funktion die VRF IP zu extrahieren.
Get_interface_neighbor	Jedes verbundene Interface hat einen Nachbar, mit dieser Funktion kann man den Nachbarn eines spezifischen Interface auslesen.
Get_neighbors	Diese Funktion liest die Hostnamen aller Nachbarn eines Netzwerkgerätes aus.
Show_cases_of_last_day	Um zu sehen welche Cases in den letzten 24 Stunden bearbeitet oder erstellt wurden, kann man diese Funktion verwenden.
Number_open_cases	Um die Anzahl der zurzeit offenen Cases nachzusehen, kann man diese Funktion verwenden.
Show_open_cases	Diese Funktion gibt eine Liste von CaseID zurück, von allen Cases die zurzeit noch offen sind.

3.6.4 DB Access

Um die manuellen Zugriffe auf die Datenbank sauber zu schreiben haben wir diese von der automatischen Datenbank Zugriffe abgekoppelt und sie hier abgelegt. Für eine Erweiterung des Systems könnte man basierend auf diesen Funktionen eine Grafische Oberfläche gestalten.
Die Verwendung und die Funktionalität der drei files sind gleich, weshalb diese hier nur einmal für alle drei erklärt wird.
Die files sind Terminalskripts welche einen durch den Create, Read, Update und Delete Prozess für Datenbankeinträge leiten. Beim Ausführen des Skripts kann man wählen welche der vier Funktionalitäten man nutzen möchte und wird dann gebeten Schritt für Schritt die Informationen einzutragen.

Funktion	Beschreibung
Insert	Um einen neuen Eintrag in der Datenbank anzulegen, kann man die Insert Funktion verwenden, diese fragt dann den Nutzer um die einzelnen Informationen welche über die Konsole eingegeben werden können.

Read	Falls man die Einträge manuell durchsuchen möchte kann man sich mit dieser Funktion eine Liste von allen Einträgen ausgeben lassen.
Update	Falls man einen Datenbank Eintrag überarbeiten möchte kann man diese Funktion wählen welche dann nachfragte welche Information des Eintrags man überarbeiten möchte.
Delete	Um einen spezifischen Eintrag in der Datenbank zu löschen kann man diese Funktion wählen und dann die ID angeben.

3.6.5 Schnittstellen

System <> Techniker

Das System übermittelt alle Meldungen an den Techniker via Slack. Der Techniker kann zudem die CRUD Funktionalitäten der Datenbank nutzen, diese sind Python Skripte welche mit Pymongo auf die MongoDB zugreifen.

Innerhalb des Systems

Die Kommunikation zwischen napalm-logs, Oatsclient und Salt geschieht über ZeroMQ.

Netzwerkgerät-OS <> napalm-logs

Die Netzwerkgeräte haben Syslog konfiguriert, diese werden auf einen definierten Port per UDP an den Server auf dem napalm-logs läuft, gesendet.

Salt <> Netzwerkgerät-OS

Um Daten zu sammeln und Befehle direkt auf den Geräten auszuführen, greift Salt mittels napalm auf die Geräte zu. Napalm benutzt für die Zugriffe verschiedene Protokolle/Tools. Für Cisco IOS welches in der Testumgebung der SA eingesetzt wurde ist dies SSH.
Sollten in Zukunft Geräte mit anderen Betriebssystemen als Cisco IOS angesteuert werden, würde sich diese Schnittstelle ändern. Dies ist aber für OATS nicht relevant, da dies von napalm erledigt wird, welches inhärent herstellerunabhängig ist.

3.7 Deployment

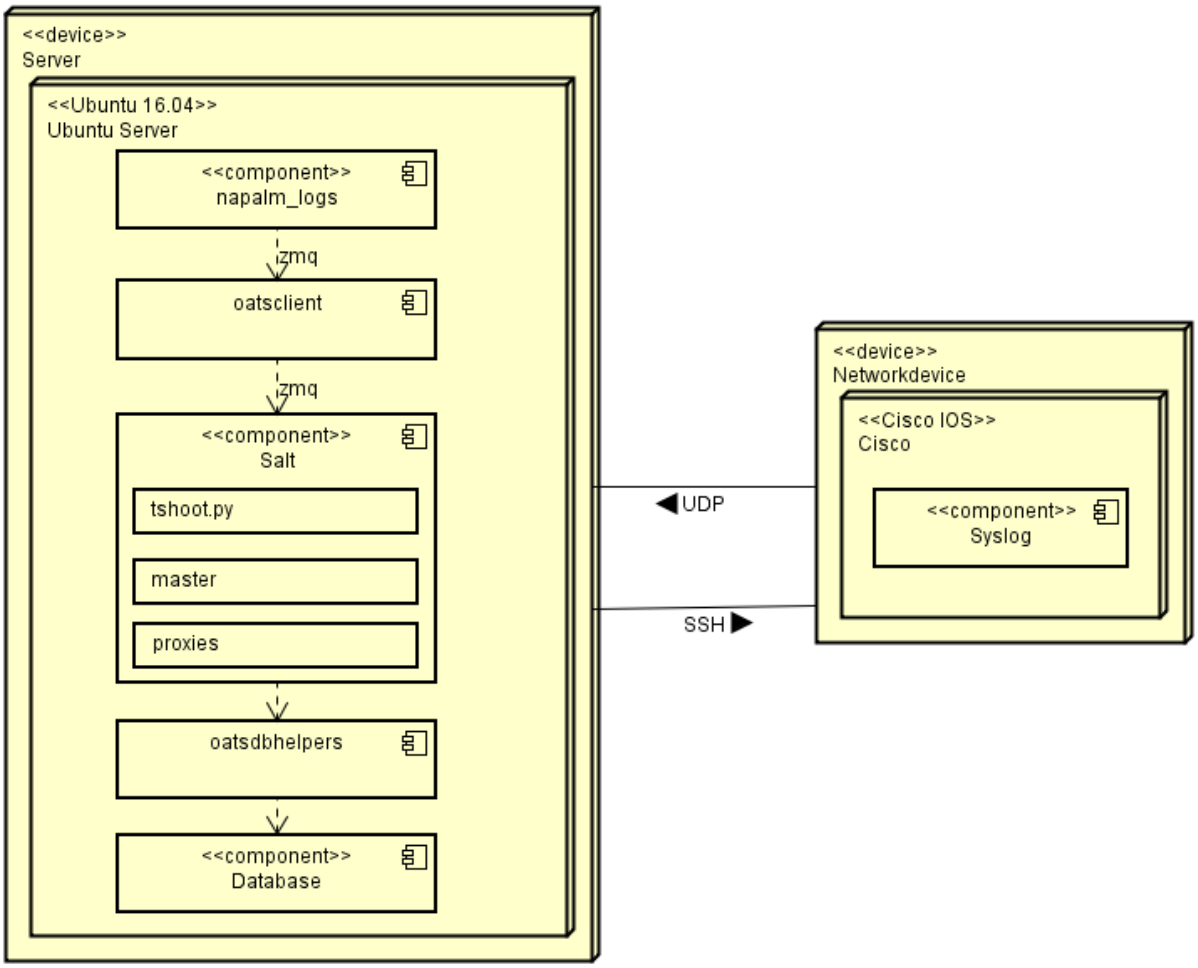


Abbildung 35: Deployment Diagramm

Netzwerkgeräte

Bei den Netzwerkgeräten handelt es sich um Cisco Router und Switches. Auf diesen muss Syslog aktiviert sein um Meldungen übertragen zu können.

Server

Beim Server handelt sich um einen Ubuntu 16.04. Auf diesem sind die verschiedenen Komponenten von OATS aktiviert. Zu Oberst brfindet sich Napalm-logs welches verantwortlich dafür ist dass die Syslog Meldungen empfangen werden. Diese werden von den Netzwerkgeräten über UDP übertragen.
Die erhaltenen Meldungen werden per ZeroMQ zum Oatsclient gesendet um dort verarbeitet zu werden. Wieder per ZeroMQ werden nun die Komponenten von Salt angesteuert. Zum einen das tshoot.py Skript welches den eigentlichen Troubleshooting Prozess durchführt, zusätzlich laufen unter Salt auch der Master und die Proxies für die einzelnen Geräte. Mit der Oatsdbhelpers Komponente kann schliesslich auf die Datenbank zugegriffen werden, welche als Service auf dem gleichen Server läuft.

3.8 Datenspeicherung

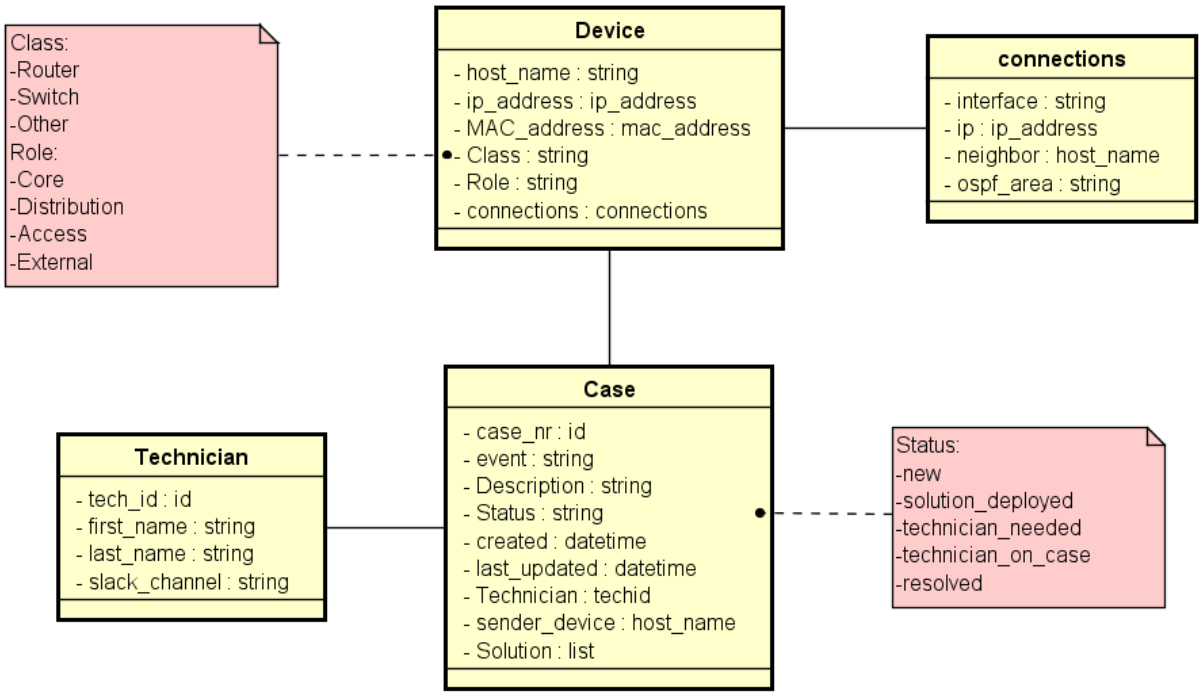


Abbildung 36: Datenmodell

3.8.1 Device

Die Klasse Device beschreibt ein einzelnes Netzwerkgerät. Die Daten die hier gespeichert werden dienen zum einen dazu, dass man vom Salt-Master auf das Gerät zugreifen kann, als zweites um von einem Netzwerkgerät zu einem anderen eine Verbindung aufzubauen.
host_name: Der Hostname eines Netzwerkgeräts ist der Name des Salt-minion oder proxies der für dieses Gerät zuständig ist.
ip_address: Die IP Adresse des Management VRFs auf dem Gerät.
MAC_address: Die MAC Adresse eines Geräts, vor allem wichtig für Layer 2 Geräte.
Class: Beschreibt die Art des Geräts, zurzeit gibt es Router Switches und Other.
Role: Beschreibt die Rolle welches ein Gerät innerhalb des Netzwerks hat. Zurzeit gibt es Core, Distribution, Access und External.
Connections: Die Connections ist eine List welche die einzelnen Ports eines Netzwerkgeräts beschreibt.
Die Device Einträge in der Datenbank enthalten auch Informationen die man eigentlich auch direkt von einem Gerät holen könnte. Hier muss man jedoch beachten, dass diese nicht Verfügbar wären, falls man nicht auf ein Gerät zugreifen kann.

3.8.2 Connections

Connections sind die Verbindungen eines Geräts zu einem anderen. Die Informationen dienen dazu um diese Verbindungen zu überprüfen und zu testen.
Interface: Der Name des Interface wie es auf einem Gerät gespeichert ist, ein Beispiel hierfür wäre: GigabitEthernet1.
IP: Die IP Adresse des Interfaces welche benötigt wird um das verbundene Gerät zu erreichen.
Neighbor: Speichert den Hostnamen des verbundenen Nachbargeräts.
ospf_area: Falls diese Verbindung zu einer bestimmten OSPF Area gehört wird dies hier gespeichert.

3.8.3 Cases

In den Cases werden die Informationen zu den einzelnen Problemfällen gespeichert.
case_nr: Eine zwölfstellige generierte ID die dazu dient, einen spezifischen Fall in der Datenbank wiederzufinden.
Event: Hier wird das Event gespeichert welches den Troubleshooting Fall ausgelöst hat, dieser könnte zum Beispiel ein Interface down Event sein.
Description: Eine kurze Beschreibung für das Problem, dies ist vor allem dann wichtig wenn ein Case manuell erstellt wird.
Status: Status ist ein ENUM der beschreibt wie weit das Problem bereits behoben wurde. Es gibt „new“ für alle neu erstellten Fälle, „solution_deployed“ für Fälle bei denen das System probiert diese zu beheben, „technician_needed“ falls ein Fall nicht automatisch behoben werden konnte und es einen manuellen Eingriff benötigt, „technician_on_case“ falls zurzeit ein Techniker versucht dieses Problem zu lösen und schliesslich „resolved“ wenn ein Problem behoben wurde und der Fall abgeschlossen wurde.
Created: Die Zeit und das Datum wann der Fall erstellt wurde.
last_updated: Der Zeitpunkt an dem der Fall das letzte Mal bearbeitet wurde.
Technician: Falls ein Techniker manuell versucht diesen Fall zu beheben, wird dieser hier eingetragen.
Sender device: Der Hostname des Geräts bei welchem das Problem aufgetreten ist, welches zur Erstellung eines Falls führte, wird hier gespeichert.
Solution: Eine Liste die alle Lösungsschritte beinhaltet welche zur Behebung des Problems vom System unternommen wurde.

3.8.4 Technician

Techniker welche in der Lage sind Probleme zu beheben und Fälle bearbeitet aber auch erstellen können.
Id: Eine selberwählbare ID mit der man ein Techniker eindeutig in der Datenbank finden kann.
First name: Der Vorname eines Technikers.
Last name: Der Nachname eines Technikers.
Slack channel: Der Slack-Channel mit welchem man einen Techniker erreichen kann.

3.9 Prozesse und Threads

Selbst programmierte Threads kommen an 2 unterschiedlichen Stellen des Projekts zum Einsatz. Zusätzlich sind die beiden Tools napalm-logs und Salt z.T. multithreaded. Der Rest verläuft seriell.

3.9.1 oatsclient.py

Für jede von napalm-logs erhaltene Meldung die an den Salt-master weitergeleitet werden soll, wird ein Thread erstellt, damit der Oatsclient beim eventuellen Aggregieren bzw. Erstellen des Salt-Events nicht blockiert wird.

3.9.2 tshoot.py

Die Methode ospf_nbr_down startet einen Thread um die Anzahl von hochgefahrenen OSPF-neighbors als Reaktion auf den Workflow zu zählen. Damit wird entschieden ob der Workflow erfolgreich war oder nicht.

4. Anleitungen

Änderungsgeschichte

Datum	Version	Änderung	Autor
05.12.2017	1.0	Erstellen des Dokuments	rj
11.12.2017	1.1	Ergänzung Installationsanleitung Schreiben Benutzungsanleitung	nv
12.12.2017	1.2	Schreiben Development	nv

4.1 Einführung

4.1.1 Zweck

In diesem Dokument wird beschrieben, wie OATS installiert und benutzt werden kann. Im letzten Kapitel wird zudem beschrieben, wie OATS erweitert werden kann.

4.1.2 Gültigkeitsbereich

Das Dokument ist gültig während der Entwicklung vom automatisierten Troubleshooting und während der nachfolgenden Bachelorarbeit.

4.2 Installationsanleitung

4.2.1 Python

Als Programmiersprache und zur Ausführung verschiedener Module wird Python 2.7.14 benötigt diese ist zum Downloaden auf der offiziellen Python Seite verfügbar.
https://www.python.org/downloads/

4.2.2 Pip

Pip ist ein Python Package Manager welcher benötigt wird um weitere Module zu installieren, in der oben erwähnten Python Version ist pip bereits enthalten, ein separater Download ist verfügbar unter.
https://pip.pypa.io/en/stable/installing/
Sobald pip installiert ist muss man es manuell auf die neuste Version upgraden. Dies ist mit dem folgenden Befehl möglich.

```
pip install -U pip
```

4.2.3 Salt

Es existieren mehrere Arten, mit der Salt installiert werden kann. Um zu wissen welche für den eigenen Anwendungsfall die Beste ist, ist tieferes Verständnis von Salt notwendig. Wir empfehlen deshalb eine offizielle Anleitung von folgender URL:
https://docs.saltstack.com/en/latest/topics/installation/index.html

4.2.4 Napalm

Der normale Installationsvorgang ist unter folgender URL verfügbar:
https://napalm.readthedocs.io/en/latest/installation/index.html

Während des Verlaufs der SA wurde aber napalm 2.0 veröffentlicht. Salt, welches napalm eigentlich unterstützt, hat zu diesem Zeitpunkt allerdings noch nicht nachgezogen, was zur Folge hat, dass man napalm wie folgt installieren muss damit es mit salt zusammen funktioniert:

```
sudo pip install git+https://github.com/napalm-automation/napalm-base.git
sudo pip install –install-option=’ios’ –force-reinstall -U napalm
```

4.2.5 Napalm-logs

Siehe: http://napalm-logs.readthedocs.io/en/latest/installation/index.html
Wir empfehlen die direkte Installation mittels:

```
sudo pip install napalm-logs
```

4.2.6 OATS

Es muss lediglich das OATS Github Repository geklont (Ort des Ordners nicht relevant) und ein Skript zum Kopieren der Dateien ausgeführt werden:

```
git clone https://github.com/nvinzens/SA_AT
sudo ./copy_repo_files.sh
```

Das Skript verschiebt alle nötigen Dateien an den richtigen Ort und erstellt die dazugehörigen Directories.

4.2.7 MongoDB

Installation

Als Datenbank wird eine MongoDB benutzt dies muss auf einem Server installiert und ausgeführt werden. Unter: https://docs.mongodb.com/manual/installation/ sind Tutorials verfügbar welche einen durch die Installation auf einem spezifischen Gerät führen.
Falls die MongoDB nicht auf derselben Umgebung laufen wird wie der Rest des OATS müssen die MongoClient Verbindungen jeweils mit der Verbindung zum Server angepasst werden.

Oatsdbhelpers

Damit OATS automatisch auf die Datenbank zugreifen kann, haben wir die dazu verwendeten Funktionen in einem Package abgelegt. Dieses Package, welches wir oatsdbhelpers genannt haben, ist unserem Github Projekt unter OATS/saltstack/oatsdbhelpers zu finden.
Installiert werden kann dieses Package mit pip dem Package Manager von Python.

```
pip install /path/to/oatsdbhelpers
```

4.3 Konfigurationsanleitung

Gewisse Teile von OATS müssen noch mittels Konfigurationsdateien an die Umgebung angepasst werden.

4.3.1 Napalm-Logs

Die napalm-logs Konfigurationsdatei befindet sich unter (logs ist die Konfigurationsdatei):

```
/etc/napalm/logs
```

Darin werden werden die Listener- sowie die Publisher-Addresse spezifiziert.

```
address: 10.20.1.10
port: 33333
publish_address: 10.20.1.10
publish_port: 49017
transport: zmq
disable_security: true
log_level: info
publisher:
  zmq:
    send_unknown: true
log_file: /etc/napalm/log
```

`address` und `publish_address` geben an unter welcher IP-Adresse auf Syslog Meldungengehorcht wird und unter welcher IP die resultiernden napalm-logs Meldungen veröffentlicht werden. Diese müssen mit der IP-Adresse des Servers auf dem OATS läuft übereinstimmen (in der SA war dies 10.20.1.10).
`port` gibt an, auf welchem Port auf Syslog Meldungen gehorcht wird. Er kann frei gewählt werden, es muss aber sichergestellt sein, dass er nicht von einem anderen Prozess belegt ist. In der SA wurde Port 33333 gewählt.
`publish_port` gibt an, auf welchem Port napalm-logs Meldungen veröffentlich werden. Dieser muss auf 49017 gesetzt sein, da der dazugehörige Client (oatsclient.py) standartmässig die Meldungen von diesem Port erwartet.
Der Rest der Datei sollte so belassen werden.

4.3.2 Salt-Proxies

Salt-proxies dienen der Kommunikation mit den Netzwerkgeräten. Dabei wird für jedes Gerät, welches per OATS angesteuert werden können soll, ein Proxy erstellt.
Die Proxy-Konfigurationsdateien befinden sich unter:

```
/srv/saltstack/pillar
```

In top.sls müssen alle Proxies angegeben werden.
Die Proxy-Konfigurationsdateien haben folgende Form (Beispiel: R11 aus der SA):

```
proxy:
  proxytype: napalm
  driver: ios
  host: 10.20.1.11
  username: ***
  passwd: ***
  provider: napalm
  optional_args:
    secret: ***
```

Angepasst werden müssen folgende Felder:

`host`: Die IP-Adresse des Netzwerkgeräts
`username`: Der Username mit dem man sich auf dem Gerät einloggen kann
`passwd`: Das Passwort für den User
`secret`: Das Enable-Passwort des Routers (optional)
Die restlichen Felder sollten belassen werden.
Es ist zu empfehlen das hosts-file des Servers auf dem OATS läuft mit den Netzwerkgerät-Hostnamen zu ergänzen, z.B.:

```
10.20.1.11 R11
10.20.1.12 R12
10.20.1.13 R13
```

4.3.3 Netzwerkgeräte

Alle Geräte die Syslog Meldungen an OATS schicken sollen müssen wie folgt konfiguriert werden:

Privilege:

```
router(config)#username <user> privilege 15
```

user: Der User der in den Proxies angegeben wurde. So wird sichergestellt, dass der User über alle Rechte verfügt, die er benötigt um via napalm Befehle auszuführen.

Syslogging

```
router(config)#logging host <OATS-IP> transport udp port <napalm-logs-port>
```

OATS-IP: Die IP welche in der napalm-logs Konfigurationsdatei als Listener-IP angegeben wurde (10.20.1.10 in der SA).
napalm-logs-port: Der Port der in der napalm-logs Konfigurationsdatei als Listener-Port angegeben wurde (33333 in der SA).

```
router(config)#logging trap <level>
```

level: Der Syslog-Logging-Level (empfohlen: 5)

Hostnames:
Damit die bestehenden Workflows funktionieren, muss auf jedem Gerät konfiguriert werden, welche Hostnamen die umliegenden Geräte verfügen und über welche IP diese erreichbar sind:

```
router(config)#ip host <hostname> <IP-address>
```

hostname: Hostname des Routers, z.B. R11
IP-address: IP-Adresse des Routers, z.B. 172.16.12.1
Dies muss für jeden umliegenden Router auf jedem Router wiederholt werden.

4.4 Benutzungsanleitung

Folgende Anleitung setzt voraus, dass alle Komponenten korrekt installiert wurden und sich alle relevanten Dateien am richtigen Ort befinden.
Da Salt voraussetzt, dass der Salt-prozess über Root Rechte verfügt, müssen auch nachfolgenden Schritte mit Root-Rechten ausgeführt werden. Dazu dient folgender Befehl:

```
~$sudo su
```

4.4.1 Salt Environment

Folgende Abschnitte dienen nur zum Aufzeigen, wie alle Salt spezifischen Prozesse gestartet werden können. Für ein tieferes Verständnis zu Salt sollte https://docs.saltstack.com konsultiert werden. Die Urheber dieses Dokuments empfehlen zusätzlich folgendes Tutorial durchzuführen:
https://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html

Salt Master

Der Salt-master dient als zentrale Sammelstelle für Events und führt als Reaktion darauf Workflows aus. Er übernimmt auch die Kommunikation mit den Salt-proxies und minions. Zum Starten des Salt-masters als Daemon muss folgender Befehl ausgeführt werden (optional: Debug-Modus):

```
# salt-master -d
```

bzw.

```
# salt-master -l debug
```

Salt Minion

Für OATS wird nur ein minion benötigt, nämlich derjenige der auf dem master selbst (dient zur Konfiguration des masters). Folgender Befehl startet den minion als Daemon:

```
# salt-minion -d
```

An diesem Punkt lohnt es sich zu überprüfen, ob alles korrekt funktioniert. Folgender Befehl testet, ob master und master-minion miteinander kommunizieren können:

```
# salt master test.ping
```

Dies sollte folgenden output generieren:

```
master:
    True
```

Salt Napalm-Proxies

Nun müssen noch die Salt napalm-proxies gestartet werden. Die proxies dienen zur Kommunikation mit den Netzwerkgeräten und benutzen die in Salt ingtegrierte napalm-API. Die proxies können mit folgendem Befehl als Daemon gestartet werden (wobei <ID> mit dem jeweiligen Namen des proxys ersetzt werden muss). Die Namen der proxies müssen mit denen unter /srv/saltstack/pillar/übereinstimmen:

```
# salt-proxy --proxyid <ID> -d
```

Die in der SA benutzten proxies waren R11-R16 und SW01/SW02. Mittels test.ping kann nun wieder getestet werden, ob die proxies hochgefahren wurden:

```
# salt '*' test.ping
```

Dies sollte folgenden output generieren (Reihenfolge kann variieren):

```
R16:
    True
SW02:
```

```
    True
master:
    True
R14:
    True
R12:
    True
SW01:
    True
R15:
    True
R11:
    True
R13:
    True
master:
    True
```

4.4.2 Napalm-Logs

Napalm-logs dient als Sammelstelle für alle im Netz generierten Syslog-Meldungen und bringt diese in eine Form aus der Daten extrahiert werden können. Ein während der SA geschriebener client (oatsclient.py) verarbeitet diese Daten und schickt sie an den Salt-master weiter.

Listener

Der napalm-logs Listener wird wie folgt gestartet:

```
# napalm-logs
```

Dies blockiert das Terminal, d.h. alle nachfolgenden Prozesse müssen in neuen Terminals gestartet werden.

Client

Damit der nachfolgende Befehl zum Starten des Clients funktioniert, muss man sich im richtigen Ordner befinden. Dies ist in unserem Fall:

```
# cd /etc/napalm/scripts/
```

Starten des Clients:

```
/etc/napalm/scripts# python oatsclient.py
```

Auch dies blockiert das aktuelle Terminal. Dieses generiert aber output, kann also im Vordergrund gelassen werden, um zu beobachten was vor sich geht.

4.4.3 MongoDB

Die MongoDB wir je nach OS unterschiedlich gestartet, basierend auf der Installation. Für eine Linux Ubuntu Version kann sie mit dem Befehl:

```
sudo service mongod start
```

gestartet werden. Um zu überprüfen ob sie korrekt gestartet wurde kann man im Logfile unter /var/log/mongodb/mongod.log nach der folgenden Linie suchen:

```
[initandlisten] waiting for connections on port <port>
```

Wobei es sich bei <port> um den unter /etc/mongod.conf konfigurierten Port handelt, Standard ist hier 27017.
Nun müssen die Anfangsdokumente in die Datenbank eingefügt werden, in unserem Projekt arbeiten wir mit der Datenbank `oatsdb` und den Collections `cases`, `network` und `technician`.
Der Befehl um ein Dokument zu importieren sieht wie folgt aus:

```
mongoimport --db oatsdb --collection <collection> --drop --file
/path/to/file/file.json
```

Die Datenbank ist hier bereits `oatsdb` die Collection kann bei <collection> eingefügt werden. Das "`–drop`" Keyword löscht eine bereits vorhandene Collection mit dem gleichen Namen, falls weitere Dokumente in die gleiche Collection importiert werden, muss dieses weggelassen werden.
Vorlagen zur Dokumentstruktur sind in unserem Github Projekt unter OATS/saltstack/database/data/templates zu finden.
Hat man die MongoDB installiert und die Files importiert, ist die Datenbank operationsfähig. Falls man einfach auf die Datenbank zugreifen möchte, haben wir unter OATS/saltstack/database/python_classes/ drei Python Skripts erstellt welche man nun über die Konsole mit Python öffnen kann.

```
python /OATS/saltstack/database/python_classes/db_crud_cases.py
python /OATS/saltstack/database/python_classes/db_crud_device.py
python /OATS/saltstack/database/python_classes/db_crud_technician.py
```

Die Skripts greifen jeweils auf die Collection zu nach der sie benannt wurden.

4.4.4 Testing der Workflows

Nun ist OATS funktionsfähig und wird bei allfälligen Fehlern die richtigen Workflows ausführen (Zum Zeitpunkt des Endes der SA sind dies INTERFACE_CHANGED und OSPF_NEIGHBOR_DOWN).
Ein INTERFACE_CHANGED Event kann in der SA Testumgebung generiert werden, indem ein Subinterface welches mit einem anderen Router im gleichen Netz verbunden ist, heruntergefahren wird:

```
R11#conf t
R11(config)#interface GigabitEthernet2.12
R11(config-subif)#shutdown
```

Dies sendet eine INTERFACE_CHANGED Syslog Meldung zu napalm-logs und löst so den Troubleshooting-Prozess aus. Am Ende des Workflows wird das Interface automatisch wieder hochgefahren.
Analog kann der OSPF_NEIGHBOR_DOWN Workflow ausgelöst werden, indem der OSPF Prozess eines Routers runtergefahren wird:

```
R14#conf t
R14(config)#router ospf 1
R14(config-router)#shutdown
```

Sobald die OSPF dead timer der umliegenden Router ausgelaufen sind, lösen die dazugehörigen Syslog Meldung den OSPF-NEIGHBOR_DOWN Troubleshooting-Prozess aus. Am Ende des Workflows sollte der OSPF-Prozess wieder hochgefahren werden.

4.5 Development

Um einen neuen Troubleshooting-Workflow zu OATS hinzuzufügen müssen folgende Schritte getätigt werden:

4.5.1 Syslog

Als erstes muss eine Syslog Meldung identifiziert werden, mit der ein Troubleshooting-Prozess ausgelöst werden kann. Existiert eine solche Meldung muss napalm-logs erweitert werden, damit die übermittelten Daten aus der Meldung extrahiert werden können.

4.5.2 Napalm-Logs

Wie napalm-logs um neue Syslog Meldung erweitert werden kann ist hier beschrieben:
http://napalm-logs.readthedocs.io/en/latest/developers/device_profiles.html

Im Rahmen der SA wurden bereits mehrere solche Profile geschrieben. Die Profile, sowie auch der Prefix-Parser für Cisco IOS befinden sich im OATS Github-Repository unter `napalm/logs/ios/`. Wir empfehlen, dass man sich beim Development von neuen Profilen an den existierenden orientiert. Auch bereits in napalm-logs existierende Profile von anderen Betriebssystemen können sich als hilfreich erweisen. Diese findet man unter:
https://github.com/napalm-automation/napalm-logs/tree/master/napalm_logs/config

4.5.3 Oatsclient

Der Oatsclient (oatsclient.py) erfüllt folgende Funktionen:

- Empfangen der Daten von napalm-logs
- Erstellen und Senden der Events an den Salt-event-bus (mittels salt_event.py)
- Aggregation von Events (mittels aggregate.py)

Alle Salt-events welche an den Salt-event-bus übermittelt werden haben folgende Form:
`napalm/syslog/*/OSPF_NEIGHBOR_DOWN/dead_timer_expired`

Konstruktion der Salt-Events

Um zu verstehen woher OSPF_NEIGHBOR_DOWN und dead_timer_expired extrahiert werden, muss man verstehen, wie die Daten von napalm-logs modelliert werden. Als Beispiel dient hier eine OSPF-NEIGHBOR_DOWN message:


```
{
  "error": "OSPF_NEIGHBOR_DOWN",
  "facility": 3,
  "host": "vmx01",
  "ip": "127.0.0.1",
  "message_details": {
      "date": "Jun 21",
      "facility": 3,
      "host": "vmx01",
      "hostPrefix": null,
      "message": "OSPF neighbor 1.1.1.1 (realm ospf-v2 ge-0/0/0.0 area 0.0.0.0)
state changed from Full to Down due to InActiveTimer (event reason: BFD session
timed out and neighbor was declared dead)",
      "pri": "29",
      "processId": "2902",
      "processName": "rpd",
      "severity": 5,
      "tag": "RPD_OSPF_NBRDOWN",
      "time": "14:03:12"
  },
  "os": "junos",
  "severity": 5,
  "timestamp": 1498053792,
  "yang_message": {
      "network-instances": {
          "network-instance": {
              "global": {
                  "protocols": {
                      "protocol": {
                          "ospf": {
                              "ospfv2": {
                                  "areas": {
                                      "area": {
                                          "area": {
                                              "interfaces": {
```

```
                                                "interface": {
                                                    "GigabitEthernet2.12": {
                                                        "neighbors": {
                                                            "neighbor": {
                                                                "172.16.12.1": {
                                                                    "state": {

"adjacency-state": "DOWN",

"adjacency-state-change-reason": "Dead timer expired",

"adjacency-state-change-reason-message": "DEAD"
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    },
    "yang_model": "openconfig-ospf"
}
```


Die message ist als Yang-Modell modelliert und wird im Oatsclient serialisiert (bzw. in ein python-dict transformiert).
OSPF_NEIGHBOR_DOWN entspricht dem Feld `error` und dead_timer_expired dem Feld `adjacency-state-change-reason`.

Direktes Weiterleiten der Events

Der einfachste Anwendungsfall von OATS ist das direkte Weiterleiten der wichtigsten Daten aus der syslog-message in den Salt-Event-Bus. Um dies zu erreichen muss zuerst ein `opt_arg` für das neue Event definiert werden. Dies geschieht in clienthelpers.py in der Methode `get_optional_arg(msg, error)`. Nur Events denen ein `opt_arg` zugeteilt werden kann, werden weitergeleitet. Da jedes von napalm-logs übermittelte Event ein unterschiedliches Modell hat, muss für jedes Event eine neue Methode geschrieben werden, mit der der `opt_arg` extrahiert werden kann. Der `opt_arg` kann auch frei gewählt werden. Er dient nur dem klaren Unterscheiden von Events im Salt-Event-Bus, da zwei unterschiedliche Events das gleiche `error` Feld haben können (z.B. INTERFACE_CHANGED).
Es bietet sich auch an, für das neue Event direkt eine konstante in aggregate.py zu erstellen, um sicherzustellen, dass immer der gleiche String benutzt wird.

Aggregieren von Events

Auch um Events zu aggregieren muss wie oben zuerst ein `opt_arg` definiert werden. In aggregate.py muss `AGGREGATE_EVENTS` um das Event erweitert werden und `EVENT_OPTIONAL_ARGS` muss um den dazugehörigen `opt_arg` erweitert werden.

Die Methode `__get_n_of_required_events(error, host, yang_message, case)` muss so erweitert werden, dass für das neue Event bekannt ist auf wie viele weitere Events der gleichen Art gewartet werden soll.
Die Methode `__get_optional_arg(error)` dient um einen sekundären opt_arg zu definieren, der dem Event angehängt werden soll, wenn die Anzahl der Events nicht mit derjenigen von `__get_n_of_required_events` übereinstimmt. Auch diese Methode muss um einen frei wählbaren `opt_arg` erweitert werden.

Überlegungen für die Zukunft

Dem Projektteam ist bewusst, dass es nicht ideal ist, dass der Code direkt editiert werden muss, um ein neues Event an den event-bus zu senden.
Idealerweise wäre es möglich, die Events als YAML-Files abzulegen, in denen alle wichtigen Parameter wie `opt_arg` und `n_of_required_events` direkt mitgegeben werden können.
Leider hat die Zeit dafür im Rahmen der SA dafür nicht gereicht.

4.5.4 Salt

Salt erfüllt innerhalb von OATS folgende Funktionen:

- Entgegennehmen von Events
- Ausführen von Workflows

Hier wird nur erklärt, welche Dateien man erweitern/ändern/neu erstellen muss um OATS zu erweitern. Für Informationen wie Salt funktioniert verweisen wir auf:
https://docs.saltstack.com/en/latest/
Um Salt für OATS zu erweitern sind folgende Schritte nötig:

Salt Master File

Die Salt-master Datei muss um das neue Event erweitert werden, auf welches gehorcht werden soll. Es ist wichtig, dass das im Oatsclient erstellte und das in der master Datei angegebene Event übereinstimmen.
Die master Datei befindet sich standartmässig unter `/etc/salt.`
Die Events werden direct im Salt-reactor innerhalb der Salt-master Datei wie folgt angegeben:

```
reactor:
  - 'napalm/syslog/*/INTERFACE_CHANGED/down':
    - /srv/saltstack/reactor/ifdown_workflow.sls
```

Das Event auf das gehorcht werden soll:

```
napalm/syslog/*/INTERFACE_CHANGED/down
```

Der Salt-state der als Reaktion auf das Event ausgeführt werden soll:

```
/srv/saltstack/reactor/ifdown_workflow.sls
```

Salt Reactor

Im Salt-reactor-directory werden state files abgelegt, welche Salt Funktionen (bzw. die in OATS geschriebenen Workflows) ausführen. Sie befinden sich wie man oben bereits sieht unter `/srv/saltstack/reactor`.
Ein state file welches einen INTERFACE_DOWN-Workflow ausführt sieht folgendermassen aus:

```
{% set event_data = data['data'] %}

ifdown_workflow:
  runner.tshoot.ifdown:
    - host: {{ event_data['minion'] }}
```

```
    - origin_ip: {{ event_data['origin_ip'] }}
    - yang_message: {{ event_data['yang_message'] }}
    - error: {{ event_data['error'] }}
    - tag: {{ event_data['tag'] }}
    - current_case: {{ event_data['case'] }}
```

Die in geschweiften Klammern vorkommenden Ausdrücke sind Jinja-statements. Mit Jinja werden dem jeweiligen Event die Daten entnommen und dem Workflow übergeben. Mehr Informationen zu Jinja findet man unter:
http://jinja.pocoo.org/docs/2.10/

Der Name (z.B. ifdown_workflow) ist frei wählbar.

`runner.tshoot.ifdown` gibt an welche Funktion in welchem file in welchem directory ausgeführt werden soll:
`runner`: gibt an, dass es sich beim auszuführenden file um einen Salt-runner handelt (https://docs.saltstack.com/en/latest/ref/runners/) und sich im directory `/srv/saltstack/salt/_runners` befindet.
`tshoot`: gibt an wie das python-file heisst (tshoot.py).
`ifdown`: gibt an welche Funktion innerhalb des files ausgeführt werden soll.

Zuunterst folgt eine Liste von Parametern, die von der ifdown-Funktion übernommen werden, d.h. Die Funktion ifdown im file tshoot.py hat folgende Form:

```
ifdown(host, origin_ip, yang_message, error, tag, current_case=None)
```

Salt-Runners / Workflows

Die Workflows werden in OATS durch Salt-runners abgebildet, d.h. die gesamte Logik passiert innerhalb von Python-files im `_runners` directory.
Um einen neuen Workflow zu schreiben muss lediglich eine neue Funktion innerhalb von tshoot.py erstellt werden, welche von einem reactor state file ausgeführt wird.
Es wird empfohlen die bereits bestehenden workflows ifdown und ospf_neighbor_down als Template zu nehmen. Auch existieren bereits eine Reihe von Helfer-Methoden innerhalb von oatssalthelpers.py. Dort werden mittels salt.cmd und salt.execute Salt-runner und execution modules ausgeführt, mit welchen z.B: mittels napalm Informationen von Geräten gesammelt werden können.
Siehe auch:
https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.napalm_network.html

Überlegungen für die Zukunft

Das Ausführen von bereits existierenden Workflows ist durch Salt sehr einfach gestaltet. Das Schreiben von neuen Workflows kann allerdings beliebig komplex werden, weswegen Programmierwissen vorausgesetzt wird.
Der Aufwand innerhalb eines Workflows kann massiv verringert werden, wenn ein Teil der Datensammlung bereits im Oatsclient (durch Aggregation oder Correlation) erledigt wird. So wird in `ospf_nbr_down` lediglich der OSPF-Prozess auf dem betroffenen Router neu gestartet und überprüft ob dies das Problem behoben hat.
Wenn ein Grossteil der Daten bereits ausserhalb von Salt gesammelt wird, wäre es vorstellbar, dass man auch hier die Workflows direkt als YAML-files ablegen könnte. Dafür müsste man z.B. schon wissen auf welchem Gerät welches Problem besteht, bevor ein Event an Salt übermittelt wird.
Auch erwähnt werden muss, dass im jetzigen Stand von OATS eine MongoDB, welche an OATS angebunden ist, existieren muss. Gewisse Informationen sind nur dort verfügbar (z.B. welche Geräte per OSPF routen). Idealerweise wäre diese austauschbar, sodass diese Informationen auch durch andere Quellen geliefert werden können. Leider was auch dafür nicht mehr genug Zeit verfügbar im Rahmen der SA.

5. Systemtests

Änderungsgeschichte

Datum	Version	Änderung	Autor
24.10.2017	1.0	Erstellen des Dokuments	rj
30.10.2017	1.1	Überarbeitung der Testszenarios	rj
15.12.2017	2.0	Überarbeitung vor der Abgabe	nv

5.1 Voraussetzungen

OATS sowie die Testumgebung sind installiert und werden ausgeführt.

5.2 Test Spezifikation

UC1: Problemfall erfassen

Was	Beschreibung
1.1	Events können an das System gesendet werden
1.1 Test	1. Debug auf dem Master aktivieren 2. Absetzen eines Events auf dem minion
1.2	Die erhaltenen Events werden registriert und weiterverarbeitet
1.2 Test	1. Ausgabe für erhaltenes Event aus 1.1 generieren

UC2: Normalisieren von Daten

ID	Beschreibung
2.1	NETCONF Daten können erhalten werden
2.1 Test	1. Debug auf dem Master aktivieren 2. Absetzen eines NETCONF Events auf einem minion
2.2	Die erhaltenen NETCONF Daten werden über Pyangbind umgewandelt
2.2 Test	1. Pyangbind Debug aktivieren 2. Die umgewandelten Daten aus dem Event überprüfen

UC3: Informationen von Geräten sammeln

ID	Beschreibung
3.1	Das System fragt bei Geräten Informationen ab
3.1 Test	1. Debug auf einem angeschlossenen Netzwerkgerät aktivieren 2. Absetzen einer Informationsanfrage auf dem Master
3.2	Das System erhält die Daten welche es angefragt hat
3.2 Test	1. Debug auf dem Master aktivieren

UC4: Lösung auswählen

ID	Beschreibung
4.1	Basierend auf von den Events erhalten Informationen über das Problem, kann das System entscheiden welcher Workflow angestossen wird
4.1 Test	2. Ein Test Workflow erstellen welcher an jeder Entscheidung eine Ausgabe erstellt

UC5: Befehl an Geräte senden

ID	Beschreibung
5.1	Das System sendet Konfigurationen oder Skripte an ein Gerät
5.1 Test	1. Debug auf einem angeschlossenen Netzwerkgerät aktivieren 2. Senden einer Konfigurationsänderung vom Master
5.2	Das Gerät führt erhaltene Skripte aus oder ändert Konfigurationen
5.2 Test	1. Zustand der Netzwerkgeräte Konfiguration überprüfen

UC6: Techniker aufbieten

ID	Beschreibung
6.1	Ein Techniker kann über das System aufgeboten werden.
6.1 Test	1. Nachricht aus dem System absetzen 2. Auf Slack überprüfen ob die Nachricht gesendet wurde
6.2	Das System sendet Aufgebote nur an Techniker die als verfügbar eingetragen sind
6.2 Test	1. Zwei verschieden Technikerdaten einrichten 2. Einer als Verfügbar einstellen 3. Nachricht absetzen 4. Im Slack überprüfen ob die Nachricht korrekt gesendet wurde

UC7: Problemfall Status bearbeiten

ID	Beschreibung
7.1	Der Status eines Problemfalls kann in der Datenbank eingesehen werden
7.1 Test	1. Überprüfen des Datenbankeintrags
7.2	Der Problemstatus kann geändert werden
7.2 Test	1. Status eines noch nicht erledigten Eintrags ändern

UC8: Lösung verifizieren

ID	Beschreibung
8.1	Das System kann ausgeführte Lösungen verifizieren
8.1 Test	1. Erstellen eines Verifikation Workflows 2. Ausgabe falls dieser erfolgreich war

UC9: Melden von häufigen Fehlern

ID	Beschreibung
9.1	Das System detektiert falls ein Fehler häufig auftritt
9.1 Test	1. Zähler für ein Testevent erstellen 2. Mehrmaliges Absetzen des Testevents
9.2	Das System meldet das häufige Auftreten von Fehlern
9.2 Test	1. Threshold für den Zähler aus 9.1 erstellen 2. Überprüfen ob die Nachricht an das Slack gesendet wurde

UC10: Reports und Statistiken abrufen

ID	Beschreibung
10.1	Vorgefertigte Reports und Statistiken über das System können abgerufen werden
10.1 Test	1. Anfragen eines Reports
10.2	Daten zum Erstellen von eigenen Reports und Statistiken können abgerufen werden
10.2 Test	1. Daten aus der Datenbank exportieren

UC11: Netzwerk Health anzeigen

ID	Beschreibung
11.1	Informationen zur Netzwerk Health können abgerufen werden
11.1 Test	1. Counter welcher die Anzahl an nicht einwandfreien Geräten zeigt anzeigen 2. Ein Gerät als Fehlerhaft setzen, 3. Überprüfen ob der Counter dies übernimmt

UC12: Problemlog erstellen

ID	Beschreibung
12.1	Erhaltene Events führen das System dazu einen Problemlog in der Datenbank zu erstellen
12.1 Test	1. Ein Event absetzen 2. Überprüfen ob es in die Datenbank eingetragen wurde

UC13: Techniker CRUD

ID	Beschreibung
13.1	Die Dokumentation der Netzwerkdaten erklärt wo diese sich befinden und wie sie eingebunden werden können.
13.1 Test	1. Eine aussenstehende Person ist in der Lage die Daten basierend auf der Dokumentation zu finden.
13.2	Die CRUD Funktionen für die Daten der Techniker sind verfügbar.

UC14: Problemfall CRUD

ID	Beschreibung
14.1	Die Problemfall Information können in der Datenbank eingesehen werden.
14.1 Test	1. Öffnen der Datenbank 2. Suchen nach den Problemfällen
14.2	Die CRUD Funktionen für Problemfall Logs sind verfügbar

UC15: Netzwerkdaten CRUD

ID	Beschreibung
15.1	Die Dokumentation der Netzwerkdaten erklärt wo diese sich befinden und wie sie eingebunden werden können.
15.1 Test	1. Eine aussenstehende Person ist in der Lage die Daten basierend auf der Dokumentation zu finden
15.2	Die CRUD Funktionen für Netzwerkdaten sind verfügbar.

UC16: Workflow verwalten CRUD

ID	Beschreibung
16.1	Die Dokumentation der Workflows, welche vom System verwendet werden, erklärt wo sich diese befinden.
16.1 Test	1. Eine aussenstehende Person ist in der Lage die Daten basierend auf der Dokumentation zu finden
16.2	Die CRUD Funktionen für die Workflows sind verfügbar.

5.3 Testprotokoll

Legende:

erfüllt bzw. umgesetzt	teilweise erfüllt bzw. teilweise umgesetzt	nicht erfüllt bzw. nicht umgesetzt

Anmerkung: Zum Zeitpunkt des Tests ist für alle workflow-relevanten Tests nur der INTERFACE_DOWN Workflow umgesetzt				
1.1 UC1: Problemfall erfassen				
Was	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
1.1	Events können an das System gesendet werden (--> Erweitern von napalm-logs)			Zum Zeitpunkt des Tests umgesetzte Events: INTERFACE_DOWN, INTERFACE_CHANGED, OSPF_NEIGHBOR_DOWN
1.2	Die erhaltenen Events werden registriert und weiterverarbeitet			
1.2 UC2: Normalisieren von Daten				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
2.1	NETCONF Daten können erhalten werden			Netconf Anbindung wurde nicht umgesetzt.
2.2	Die erhaltenen NETCONF Daten werden über Pyangbind umgewandelt			Netconf Anbindung wurde nicht umgesetzt.
1.3 UC3: Informationen von Geräten sammeln				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
3.1	Das System fragt bei Geräten Informationen ab			tshoot.ifdown sammelt Informationen per Pings, Salt unterstützt über napalm viele andere Methoden, mit denen Informationen gesammelt werden können.
3.2	Das System erhält die Daten welche es angefragt hat			Die Daten werden von napalm in JSON-Format zurückgeliefert.

1.4 UC4: Lösung auswählen				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
4.1	Basierend auf von den Events erhalten Informationen über das Problem, kann das System entscheiden welcher Workflow angestossen wird			Im Salt-reactor wird angegeben, auf welches Event gehorcht wird und welcher Workflow ausgeführt werden muss um das Event zu behandeln. Zum Zeitpunkt des Tests wird nur auf INTERFACE_CHANGED mit dem optional_arg "down" gehorcht --> napalm/syslog/*/INTERFACE_CHANGED/down/*, worauf ifdown.tshoot ausgeführt wird.
1.5 UC5: Befehl an Geräte senden				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
5.1	Das System sendet Konfigurationen oder Skripte an ein Gerät			tshoot.ifdown lädt shutdown / noshutdown configs auf das betroffene Gerät
5.2	Das Gerät führt erhaltene Skripte aus oder ändert Konfigurationen			lässt sich z.B. über das Log auf den jeweiligen Geräten überprüfen (oder direkt in napalm-logs/salt, da wiederum ein Event ausgelöst wird wenn das Interface hochfährt)
1.6 UC6: Techniker aufbieten				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
6.1	Ein Techniker kann über das System aufgeboten werden.			Es kann kein bestimmter Techniker aufgeboten werden, da diese Daten noch fehlen. Stattdessen wird eine Nachricht auf Slack gepostet, mit der sich ein Team selbst organisieren könnte
6.2	Das System sendet Aufgebote nur an Techniker die als verfügbar eingetragen sind			Siehe oben
1.7 UC7: Problemfall Status bearbeiten				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
7.1	Der Status eines Problemfalls kann in der Datenbank eingesehen werden			Per Python-Script & CLI. Wird zurzeit als JSON-Format dargestellt, könnte evtl. noch menschenfreundlicher dargestellt werden.

7.2	Der Problemstatus kann geändert werden			Der Status kann geändert werden, indem ein ganzer DB-Eintrag editiert wird. Evtl. ermöglichen, dass nur einzelne Felder editiert werden können. Evtl. anzeigen, dass beim updaten ein leerer Eintrag zu keiner Änderung in der DB führt. Darstellung beim Ändern vom Status ist nicht ideal (z.B. status can only be one the following options)
1.8 UC8: Lösung verifizieren				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
8.1	Das System kann ausgeführte Lösungen verifizieren			Verifizierung geschieht direkt im Workflow
1.9 UC9: Melden von häufigen Fehlern				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
9.1	Das System detektiert falls ein Fehler häufig auftritt			Das System "schluckt" Events, wenn ein Event in kurzer Zeit mehrmals vorkommt (R11 fährt runter --> löst Events von allen Nachbargeräten aus, die aber alle das gleiche Event beschreiben), um unnötiges mehrfaches Ausführen des gleichen Workflows zu verhindern. Gefahr von nicht erkannten relevanten Events besteht.
9.2	Das System meldet das häufige Auftreten von Fehlern			Nicht umgesetzt
1.10 UC10: Reports und Statistiken abrufen				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
10.1	Vorgefertigte Reports und Statistiken über das System können abgerufen werden			Ausgaben über die zurzeit noch offenen Fälle sowie die Fälle welche in einer letzten Zeit geändert wurden, können erstellt werden.
10.2	Daten zum Erstellen von eigenen Reports und Statistiken können abgerufen werden			Daten können aus der Datenbank exportiert werden.
1.11 UC11: Netzwerk Health anzeigen				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**

11.1	Informationen zur Netzwerk Health können abgerufen werden			Per Python-Script kann angezeigt werden, wie viele offene DB-cases vorhanden sind.
1.12 UC12: Problemlog erstellen				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
12.1	Erhaltene Events führen das System dazu einen Problemlog in der Datenbank zu erstellen			
1.13 UC13: Techniker CRUD				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
13.1	Die Dokumentation der Technikerdaten erklärt wo diese sich befinden und wie sie eingebunden werden können.			Beschreibung ist vorhanden, muss noch für Techniker vefügbar gemacht werden --> Benutzeranleitung schreiben
13.2	Die CRUD Funktionen für die Daten der Techniker sind verfügbar.			siehe Anmerkungen UC7
1.14 UC14: Problemfall CRUD				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
14.1	Die Problemfall Information können in der Datenbank eingesehen werden.			siehe Anmerkungen UC7
14.2	Die CRUD Funktionen für Problemfall Logs sind verfügbar			siehe Anmerkungen UC7
1.15 UC15: Netzwerkdaten CRUD				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**

15.1	Die Dokumentation der Netzwerkdaten erklärt wo diese sich befinden und wie sie eingebunden werden können.			siehe Anmerkungen UC7
15.2	Die CRUD Funktionen für Netzwerkdaten sind verfügbar.			siehe Anmerkungen UC7
1.16 UC16: Workflow verwalten CRUD				
ID	**Beschreibung**	**Erledigt**	**Test**	**Anmerkungen**
16.1	Die Dokumentation der Workflows, welche vom System verwendet werden, erklärt wo sich diese befinden.			Ist nicht umsetzbar wie anfangs gedacht, da Workflows zu komplex sind um sie nicht-programmatisch verwalten zu können. In der Benutzeranleitung wird erklärt, wie ein Workflow entwickelt werden kann.
16.2	Die CRUD Funktionen für die Workflows sind verfügbar.			Siehe oben

III. Glossar

1. Begriffe

Aktor	Ein Element, das eine Eingangsgröße in eine andersartige Ausgangsgröße umwandelt, um einen gewünschten Effekt hervorzurufen (Wikipedia, 2017).
Construction	Dritte Phase von RUP. Dient der Entwicklung und dem Testing des Systems.
Daemon	Programm, welches im Hintergrund abläuft und gewisse Dienste zur Verfügung stellt.
Elaboration	Zweite Phase von RUP. Dient zum Erstellen eines Architekturprototyps, einer Domainanalyse und dem Aufsetzen der Anforderungen an das System. Zusätzlich wird in dieser Zeit die nachfolgende Construction Phase geplant (Anm.: Die Studienarbeit wurde nicht vollständig mit RUP durchgeführt, sondern enthält nur einige Elemente davon)
Event	Dt. Ereignis. Ein spezielles Ereignis auf Netzwerkgeräten (z.B. unbeabsichtigtes Runterfahren des Geräts), welches einen Troubleshooting-Prozess auslöst.
Event Driven Automation	Beschreibt das automatische Ausführen von Tasks als Reaktion auf gewisse Events in einem System. So können z.B. manuelle Änderungen im System erkannt werden und rückgängig gemacht werden.
Inception	Erste Phase von RUP. Dient zur Findung eines Konzepts (Beschreibung der Funktionalität des zu entwickelnden Systems) und der Zielsetzung.
Interface (Netzwerk)	Dt. Netzwerkkarte. Dient der Verbindung zwischen 2 Geräten. Verbindet im Netzwerkumfeld vor allem Switches und Router.
Interface (Software)	Dt. Schnittstelle. Bezeichnet Schnittstellen zwischen zwei oder mehreren Systemen.
Network-Health	Aktueller Zustand des Netzwerks in dem OATS eingesetzt wird. Der Zustand leitet sich aus der offenen Anzahl Problemfälle im Netz ab.
Open Source	Bezeichnet Software deren Quellcode öffentlich verfügbar ist und gratis verwendet werden kann.
Problemfall	Auftreten eines Problems im Netzwerk in dem OATS eingesetzt wird.
Salt-engine	Extern laufende Salt-Prozesse, z.B. der Salt-reactor oder der Salt-Event-Bus
Salt-Event	Die Repräsentation eines Events welches von Syslog ausgelöst und von napalm-logs verarbeitet wurde, innerhalb von Salt
Salt-Event-Bus	Die ZeroMQ Message-Queue für Salt-Events.
Salt-master	Läuft in OATS 1x auf dem zentralen Server. Zuständig für die Orchestrierung und Kommunikation mit den Salt-minions und proxies.
Salt-minion	Objekte die eine verkleinerte Version von Salt ausführen um die lokale Ausführung von Befehlen und Kommunikation mit dem master zu ermöglichen.
Salt-proxy	Sind dem Salt-master untergeordnet und dienen als Proxy (dt. Vermittler) zu je einem Netzwerkgerät. Sie verfügen dabei über die nötigen Informationen über das Netzwerkgerät, um mit diesem zu kommunizieren.
Salt-reactor	Listener für Salt-Events auf dem Salt-Event-Bus. Führt als Reaktion auf solche Events einen Workflow aus

Sensor	Detektor von Events im Netzwerk.
Skript	Beschreibt während der Arbeit entwickelte Python-Programme.
Streaming-Telemetry	Ein moderner Ansatz um SNMP-basiertes Monitoring zu ersetzen. Dabei liefert ein Netzwerkgerät kontinuierlich Daten, welche an einer zentralen Stelle gesammelt und analysiert werden können.
Techniker	Mitarbeiter in einer Firma in dem OATS eingesetzt wird. Verfügt über vertieftes Wissen über Netzwerke, aber nicht über Programmierkenntnisse.
Tool	Dt. Dienstprogramm. Ein Programm, welches systemnahe Aufgaben ausführt. Bietet eine API mit der verschiedene Funktionen ausgeführt werden können. So werden in OATS z.B. Zugriffe auf die Netzwerkgeräte durch napalm ausgeführt, wobei der Anwender sich nicht darum kümmern muss, um welches Gerät es sich handelt.
Traffic	Netzwerkverkehr
Transition	Vierte und Letzte Phase von RUP. Dient als Übergabephase in welcher das Projekt abgeschlossen wird. Dabei liegt das Hauptaugenmerk auf der Dokumentation und dem Deployment der entwickelten Software.
Troubleshooting	Problemlösungsstrategie. Umfasst das logische und systematische Suchen nach der Quelle eines Problems. Wird im Netzwerkumfeld vor allem im Zusammenhang mit Netzwerkgeräten (Router Switches) benutzt.
Workflow	Dt. Arbeitsablauf. Beschreibt eine Reihenfolge von Arbeitsvorgängen.

2. Abkürzungen

API	Application Programming Interface
ARP	Address Resolution Protocol
BDPU	Bridge Protocol Data Units
BGP	Border Gateway Protocol
CDP	Cisco Discovery Protocol
CRC	Cyclic Redundancy Check
DHCP	Dynamic Host Configuration Protocol
EIGRP	Enhanced Interior Gateway Protocol
HSRP	Hot Standby Router Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task force
IF	Interface
IP SLA	Internet Protocol Service Level Agreement
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
JSON	Javascript Object Notation
LACP	Link Aggregation Protocol
MAC	Media Access Controll (Address)
MIB	Management Information Base
MTU	Maximum Transmission Unit
NAPALM	Network Automation and Programmability Abstraction Layer with Multivendor support
NAT	Network Address Translation

NETCONF	Network Configuration Protocol
NOSQL	Not Only SQL
nv	Nico Vinzens
OATS	Open Source Automated Troubleshooting System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PAgP	Port Aggregation Protocol
PAT	Port Address Translation
PDU	Packet Data Unit
pip	pip installs packages
QoS	Quality of Service
rj	Raphael Jöhl
RPC	Remote Procedure Call
RUP	Rational Unified Process
SA	Studienarbeit
SLAAC	Stateless Address Autoconfiguration
SLACK	Searchable Log of All Conversation and Knowledge
SNMP	Simple Network Managing Protocol
SQL	Structured Query Language
SSH	Secure Shell
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
UC	Use Case
UDP	User Datagram Protocol
UID	Unique Identifier
VLAN	Virtual Local Area Network
VRF	Virtual Routing and Forwarding
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language (ursprünglich: Yet Another Markup Language)
YAQL	Yet Another Query Language

IV. Bibliographie

1. Quellenverzeichnis

Barroso, D. (02. 10 2017). *Welcome to NAPALM's documentation!* Von NAPALM: https://napalm.readthedocs.io/en/latest/index.html abgerufen

Barroso, D. (15. 10 2017). *Welcome to the NAPALM Documentation*. Von NAPALM: http://napalm.readthedocs.io/en/latest/ abgerufen

Barroso, D. (04. 10 2017). *YANG for Dummies*. Von napalm-automation: https://napalm-automation.net/yang-for-dummies/ abgerufen

Cisco. (2015). *CCNP Routing and Switching THOOT 300-135.* Indianapolis: Cisco Press.

Cisco. (03. 10 2017). *Building Scalable syslog Management Solutions*. Von Cisco: https://www.cisco.com/c/en/us/products/collateral/services/high-availability/white_paper_c11-557812.html#wp9000392 abgerufen

Cisco. (09. 10 2017). *Cisco IOS IP SLAs Configuration Guide, Release 12.4*. Von Cisco: https://www.cisco.com/c/en/us/td/docs/ios/12_4/ip_sla/configuration/guide/hsla_c/hsoverv.html abgerufen

Cisco. (09. 10 2017). *Cisco Troubleshooting EIGRP*. Von Cisco: https://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/21324-trouble-eigrp.html abgerufen

Cisco. (09. 10 2017). *IP SLA Fundamentals*. Von The Cisco Learning Network: https://learningnetwork.cisco.com/blogs/vip-perspectives/2017/06/13/ip-sla-fundamentals abgerufen

Cisco. (09. 10 2017). *Troubleshooting OSPF*. Von Cisco: https://www.cisco.com/c/dam/en/us/support/docs/ip/open-shortest-path-first-ospf/12151-trouble-main-00.jpeg abgerufen

Cisco Learning Network. (30. 09 2017). *The 20 most common issues in Routing*. Von The Cisco Learning Network: https://learningnetwork.cisco.com/thread/63944 abgerufen

Cisco Tail-f. (03. 10 2017). *Tail-f-Systems*. Von What is NETCONF?: http://www.tail-f.com/wordpress/wp-content/uploads/2013/03/Tail-f-NETCONF-YANG-Service-Automation-LISA-Usenix-2011.pdf abgerufen

ExamCollection. (30. 09 2017). *ExamCollection*. Von How to TroubleShoot common Routers and Switches issues: https://www.examcollection.com/certification-training/network-plus-how-to-troubleshoot-common-routers-and-switches-issues.html abgerufen

IETF. (03. 10 2017). *RFC 3411*. Von IETF: https://tools.ietf.org/html/rfc3411 abgerufen

IETF. (07. 10 2017). *RFC 3954 - Cisco Systems NetFlow Services Export Version 9*. Von IETF: https://tools.ietf.org/html/rfc3954 abgerufen

IETF. (04. 10 2017). *RFC 5424 - The Syslog Protocol*. Von IETF: https://tools.ietf.org/html/rfc5424 abgerufen

IETF. (01. 10 2017). *RFC 5424 - The Syslog Protocol*. Von IETF: https://tools.ietf.org/html/rfc5424 abgerufen

IETF. (03. 10 2017). *RFC 6241 - The NETCONF Protocol*. Von IETF: https://tools.ietf.org/html/rfc6241 abgerufen

IETF. (03. 10 2017). *RFC 792 - Internet Control Message Procotol*. Von IETF: https://tools.ietf.org/html/rfc792 abgerufen

InterProjektWiki. (07. 10 2017). *InterprojektWiki*. Von InterprojektWiki: NetFlow: https://pliki.ip-sa.pl/wiki/Wiki.jsp?page=NetFlow abgerufen

ISO. (30. 09 2017). *ISO/IEC 9126*. Von International Organization for Standardization: https://www.iso.org/standard/22749.html abgerufen

Linux/Unix. (06. 10 2017). *Linux man-pages - traceroute*. Von Linux man-pages: http://man7.org/linux/man-pages/man8/traceroute.8.html abgerufen

MariaDB. (14. 10 2017). *mariaDB*. Von mariaDB: https://mariadb.com/ abgerufen
Microsoft. (09. 10 2017). *Microsoft Support*. Von The OSI Model's Seven Layers Defined and Functions Explained: https://support.microsoft.com/en-us/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained abgerufen
Microsoft. (06. 10 2017). *Tracert*. Von Technet Microsoft: https://technet.microsoft.com/en-us/library/cc940128.aspx abgerufen
Mitchell, B. (10. 05 2017). *What Is a Network Sniffer?* Von lifewire: https://www.lifewire.com/definition-of-sniffer-817996 abgerufen
MongoDB. (14. 10 2017). *MongoDB*. Von MongoDB: https://www.mongodb.com abgerufen
Morell, G. (08. 10 2017). *MAC Flaps - why are they bad*. Von Network Development Team: https://blogs.it.ox.ac.uk/networks/2011/02/04/mac-flaps-why-are-they-bad/ abgerufen
OpenConfig. (15. 10 2017). *Vendor-neutral, model-driven network management designed by users*. Von OpenConfig: http://openconfig.net/ abgerufen
PostgreSQL. (14. 10 2017). *PostgreSQL*. Von PostgreSQL About: https://www.postgresql.org/about/ abgerufen
Red Hat Ansible. (14. 10 2017). *ansible*. Von ansible: https://www.ansible.com/ abgerufen
Rivard, E. (09. 10 2017). *Pearson IT Certification*. Von Troubleshooting Along the OSI Model: http://www.pearsonitcertification.com/articles/article.aspx?p=1730891 abgerufen
Saltstack. (15. 10 2017). *Salt Open Source*. Von Saltstack: https://saltstack.com/salt-open-source/ abgerufen
Saltstack. (16. 10 2017). *Saltstack Doc*. Von SALT.MODULES.SLACK_NOTIFY: https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.slack_notify.html abgerufen
Shakir, R. (10. 15 2017). *robshakir/pyangbind*. Von GitHub: https://github.com/robshakir/pyangbind abgerufen
Shaw, A. (14. 10 2017). *Medium*. Von Ansible v.s. Salt (SaltStack) v.s. StackStorm: https://medium.com/@anthonypjshaw/ansible-v-s-salt-saltstack-v-s-stackstorm-3d8f57149368 abgerufen
Slack. (16. 10 2017). *Slack*. Von Slack: https://slack.com/ abgerufen
Stackstorm. (14. 10 2017). *Stackstorm*. Von Stackstorm: https://stackstorm.com/ abgerufen
Ulinic, M. (15. 10 2017). *Using napalm-logs for event-driven network automation and orchestration*. Von napalm-automation: https://napalm-automation.net/napalm-logs-released/ abgerufen
VMS. (01. 10 2017). *VMS - find a new path*. Von 60 Minute Labour Charge: https://www.vms4x4.com/products/60-minute-labour-charge abgerufen
wikibooks. (13. 12 2017). *Linux-Praxisbuch: Syslog*. Von wikibooks: https://de.wikibooks.org/wiki/Linux-Praxisbuch:_Syslog abgerufen
wikibooks. (04. 10 2017). *Linux-Praxisbuch: Syslog*. Von wikibooks: https://de.wikibooks.org/wiki/Linux-Praxisbuch:_Syslog abgerufen
Wikipedia. (15. 10 2017). *ZeroMQ*. Von Wikipedia: https://en.wikipedia.org/wiki/ZeroMQ abgerufen

2. Abbildungsverzeichnis

Abbildung 1: Troubleshooting Workflow 10
Abbildung 2: Architektur (Barroso, Welcome to the NAPALM Documentation, 2017) (Ulinic, 2017) (MongoDB, 2017) (Slack, 2017) (Saltstack, 2017) (wikibooks, 2017) 12
Abbildung 3: 3: ping -n5 auf hsr.ch 18
Abbildung 4: tracert auf google.com 21
Abbildung 5: SNMP-Paket Aufbau (Wikipedia, 2017) 22
Abbildung 6: Trap-PDU-Header 23
Abbildung 7: YANG Modelldefinition 26
Abbildung 8: Beispiel eines YANG Datensatzes 27
Abbildung 9: Importieren und Erweitern eines bestehenden Modells 27
Abbildung 10: NetFlow Funktionsweise (Wikipedia, 2017) 30
Abbildung 11: OSPF Troubleshooting-Prozess 38
Abbildung 12: EIGRP Troubleshooting Prozess 39
Abbildung 13: BGP Troubleshooting-Prozess 41
Abbildung 14: Campus-Netz 45
Abbildung 15: Problemdomänen als Flowchart 47
Abbildung 16: Problemlösungs-Workflow 56
Abbildung 17: Auszug der NAPALM Supportmatrix 60
Abbildung 18: get_interfaces call 61
Abbildung 19: Syslog-Meldung eines Cisco-Devices 62
Abbildung 20: Resultierende Meldung 62
Abbildung 21: Syslog Meldung eines Juniper-Devices 62
Abbildung 22: Ansible 63
Abbildung 23: Ansible Tower 64
Abbildung 24: Ansible Support 64
Abbildung 25: Stackstorm 66
Abbildung 26: Stackstorm Infrakstruktur 67
Abbildung 27: Saltstack 69
Abbildung 28: Salt Funktionsweise 69
Abbildung 29: Salt Architektur 70
Abbildung 30: ZeroMQ Code Beispiel 74
Abbildung 31: Architektur: Konzept 75
Abbildung 32: Salt-reactor YAML-File 76
Abbildung 33: Architektur: Umsetzung 78
Abbildung 34: Architekturdiagramm 81
Abbildung 35: Deployment Diagramm 85
Abbildung 36: Datenmodell 86

V. Anhänge

Projekt: Automatisiertes Troubleshooting

Raphael Jöhl
Nico Vinzens

Anhang A: Aufgabenstellung

Projekt: Automatisiertes Troubleshooting

Aufgabenstellung

Viele Arbeitsschritte bei Fehlerbehebungen könnten eigentlich automatisiert werden. So Arbeitet eine nette Stimme bei der Support Hotline aller grossen Telekom Konzerne zuerst die Checkliste ab. Das gleiche hat auch in den anderen IT Bereichen seine Gültigkeit. So sammelt ein Netzwerk Engineer bei einem Support Ticket zuerst meistens nur Informationen. Wo befindet sich das Gerät? In welchem Subnetz ist es? Kann ich es via ICMP erreichen? Solche Arbeiten eignen sich sehr gut zum Automatisieren. So können bei einem Fehlerfall die üblichen Tasks bereits abgearbeitet werden. Sind genügend Informationen vorhanden können automatisch Lösungsversuche gestartet werden oder das Problem temporär umgangen werden, so dass der Techniker am nächsten Tag ausgeschlafen die Bereinigung abschliessen kann.

Aufgabe: Entwickeln eines Systems, welches dank dem Konzept von «Event Driven Automation» das Troubleshooting automatisieren kann. Über Sensoren sollen Workflows angestossen werden, welche wiederum sogenannte Aktoren ansteuern. Es muss nicht eine komplette Eigenentwicklung geben, da es viele Komponenten bereits als OpenSource Lösungen gibt. Im gesunden Rahmen sollen diese Projekte durch Integrationen und Anpassungen zu einer Gesamtlösung verschmelzen.

Rapperswil, 19.12.2017

Name(n) STETTLER REMO

Unterschrift(en)

Anhang B: Projektplan

B.1 Projektbeschreibung

B.1.1 Zweck und Ziel

Viele Arbeitsschritte bei Fehlerbehebungen könnten eigentlich automatisiert werden. So Arbeitet eine nette Stimme bei der Support Hotline aller grossen Telekom Konzerne zuerst die Checkliste ab. Das gleiche hat auch in den anderen IT Bereichen seine Gültigkeit. So sammelt ein Netzwerk Engineer bei einem Support Ticket zuerst meistens nur Informationen. Wo befindet sich das Gerät? In welchem Subnetz ist es? Kann ich es via ICMP erreichen? Solche Arbeiten eignen sich sehr gut zum Automatisieren. So können bei einem Fehlerfall die üblichen Tasks bereits abgearbeitet werden. Sind genügend Informationen vorhanden können automatisch Lösungsversuche gestartet werden oder das Problem temporär umgangen werden, so dass der Techniker am nächsten Tag ausgeschlafen die Bereinigung abschliessen kann.

Aufgabe: Entwickeln eines Systems, welches dank dem Konzept von «Event Driven Automation» das Troubleshooting automatisieren kann. Über Sensoren sollen Workflows angestossen werden, welche wiederum sogenannte Aktoren ansteuern. Es muss nicht eine komplette Eigenentwicklung geben, da es viele Komponenten bereits als Open Source Lösungen gibt. Im gesunden Rahmen sollen diese Projekte durch Integrationen und Anpassungen zu einer Gesamtlösung verschmelzen.

Persönliche Ziele: Beide Teammitglieder sind daran interessiert, im Leben nach dem Studium im Netzwerkbereich zu arbeiten. Deswegen ist das persönliche Ziel des Teams, sich über einen längeren Zeitraum mit Technologien im Netzwerkbereich zu befassen, um für den späteren Berufsalltag vertieftes Wissen zu sammeln. Dabei sollen gelernte Software Engineering Practices angewandt werden.

B.1.2 Lieferumfang

Ein System, welches die verschiedenen Open Source Lösungen sinnvoll verbindet, damit ein automatisches Troubleshooting möglich ist.

B.1.3 Annahmen und Einschränkungen

Die Projektdauer ist auf Ende des Herbstsemesters 2017 terminiert. Das Projekt sollte im Rahmen des geforderten Zeitaufwands von 480 Arbeitsstunden, bei 2 Projektmitgliedern realisierbar sein. Das Hauptziel der Arbeit liegt in der Erarbeitung eines automatischen Troubleshooting Systems für Netzwerke.

B.2 Projektorganisation

Studiengang:	Informatik
Semester:	HS 2017/2018
Fachrichtung:	Application Design Cloud Solutions
Studenten:	Raphael Jöhl, Nico Vinzens
Verantwortlicher:	Beat Stettler
Betreuer:	Urs Baumann

B.3 Management Abläufe

B.3.1 Kostenvoranschlag

Das Projekt läuft während 14 Wochen. Insgesamt stehen für die Studienarbeit, mit zwei Studenten, 480 Stunden zur Verfügung. Für die Projektlaufzeit ergibt dies für jeden Student ca. 17 Stunden Aufwand pro Woche.

Projektdauer	14 Wochen
Projektmitglieder	2 Personen
Arbeitsstunden pro Woche und Person	17.14
Arbeitsstunden gesamt	480
Projektstart	18. September 2017
Projektende	22. Dezember 2017

B.3.2 Arbeitsaufteilung

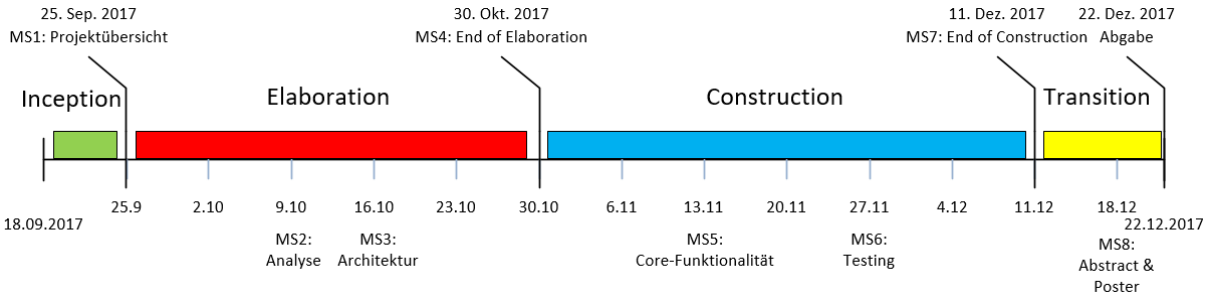
Raphael Jöhl:

- DB aufsetzen / verwalten
- Testing
- Dokumentation

Nico Vinzens:

- Programmieren der Workflows
- Infrastruktur
- Dokumentation

B.3.3 Zeitliche Planung



Jeweils am Montag setzt sich das Team zusammen, um den Fortschritt der letzten Woche zu besprechen und um sicherzustellen ob alles was am nächsten Tag gezeigt werden soll, bereit ist.

B.3.4 Inception

Die Inception Phase dauert eine Woche und dient dem Erstellen einer Projektübersicht, eines ersten Entwurfs der zeitlichen Planung, sowie der allgemeinen Recherche des Themas der Studienarbeit.

B.3.5 Elaboration

Die Elaboration Phase dauert 5 Wochen. Am Ende der Elaboration sollen alle offenen Fragen geklärt sein. D.h. die Anforderungen sind bekannt, alle Tools sind bekannt und es existiert ein funktionierender Prototyp. Die Elaboration Phase ist etwas länger angesetzt, als bei IT-Projekten typisch, da es sich um eine komplett neue Applikation handelt, welche erarbeitet werden soll.

B.3.6 Construction

Die Construction Phase dauert 6 Wochen. Während dieser Zeit wird der Prototyp weiterentwickelt. Am Ende der Construction soll das System lauffähig sein und den Anforderungen entsprechen.

B.3.7 Transition

Die Transition Phase dauert 2 Wochen. Diese Zeit wird verwendet um alle Dokumente fertigzustellen und die Präsentation vorzubereiten und durchzuführen.

B.4 Milestones

B.4.1 MS1: Projektübersicht erstellt

Datum	25. September 2017
Ziele	Erste Version des Projektplans Allgemeine Übersicht über das Projekt
Dokumente	Projektübersicht inkl. Projektplan
Verfügbare Zeit	1 Woche: 2 * 17h = 34h

B.4.2 MS2: Analyse durchgeführt

Datum	10. Oktober 2017
Ziele	Domainanalyse durchgeführt, die Domainanalyse beinhaltet: • Eine Analyse der in unserer Domain eingesetzten Technologien • Eine Analyse der auftretenden Probleme in unserer Domain • Eine Analyse der in unserer Domain eingesetzten Geräte Anforderungen festgelegt (Dokumentiert das WAS): • Use-Case-Diagramm erstellt • Anforderungen an das System aufgelistet Nichtfunktionale Anforderungen definiert (Dokumentiert das WIE)
Dokumente	Domainanalyse Anforderungsspezifikation
Verfügbare Zeit	2 Wochen: 2 * 2 * 17h = 68h

B.4.3 MS3: Architektur bekannt

Datum	16. Oktober 2017
Ziele	Dokument erstellt in welchem die Architektur des Systems beschrieben wird, mit folgenden Inhalten: • Systemübersicht: beschreibt die Komponenten des Systems • Tools: stellt die im System eingesetzten Tools vor • Schnittstellen: beschreiben der Schnittstellen inner- und ausserhalb des Systems. Beschreibt auch die wichtigsten Abläufe im System. • Datenmodellierung: zeigt auf, wie die Daten im System dargestellt bzw. abgelegt werden
Dokumente	Architekturdokument
Verfügbare Zeit	1 Woche: 2 * 17h = 34h

B.4.4 MS4: End of Elaboration

Datum	30. Oktober 2017
Ziele	Prototyp erstellt, dies beinhaltet: • Testumgebung aufgestellt • Aufsetzen eines Prototyps der in dieser Umgebung läuft Testplan aufgestellt, der in MS5 durchgeführt wird
Dokumente	Präsentation Prototyp (nicht als Dokument, live mit den Betreuern) Testplan
Verfügbare Zeit	2 Wochen: 2 * 2 * 17h = 68h

B.4.5 Core-Funktionalität implementiert

Datum	27. November 2017
Ziele	Gerüst des Systems ist aufgebaut, Arbeitspakete 1 bis 11, und 1 Workflow, Arbeitspaket 12 implementiert.
Dokumente	Zwischenresultate
Verfügbare Zeit	2 Wochen: 2 * 2 * 17h = 68h

B.4.6 MS5: Testing durchgeführt

Datum	27. November 2017
Ziele	Tests durchgeführt (wie in MS4 beschrieben): • Testprotokolle erstellt • Massnahmen aus Testresultaten abgeleitet • Ab diesem Zeitpunkt werden keine neuen Funktionalitäten hinzugefügt, nur noch bestehende verbessert/gefixt
Dokumente	Testprotokolle Testresultate inkl. Massnahmen
Verfügbare Zeit	2 Wochen: 2 * 2 * 17h = 68h

B.4.7 MS6: End of Construction

Datum	11. Dezember 2017
Ziele	Funktionalität des Systems stimmt mit Anforderungen überein, die in der Elaboration definiert wurden
Dokumente	Umgesetzte Anforderungen werden in die Doku übertragen
Verfügbare Zeit	2 Wochen: 2 * 2 * 17h = 68h

B.4.8 MS7: Abstract & Poster erstellt, Dokumente aktualisiert

Datum	18. Dezember 2017
Ziele	Erstellung Abstract & Poster Alle Dokumente auf den neuesten Stand aktualisiert
Dokumente	Abstract & Poster
Verfügbare Zeit	1 Woche: 2 * 17h = 34h

B.4.9 Abgabe

Datum	22. Dezember 2017
Beschreibung	Alle Dokumente abgegeben Schlusspräsentation durchgeführt
Dokumente	Alle für die Bewertung relevanten Dokumente Schlusspräsentation
Verfügbare Zeit	1 Woche: 2 * 17h = 34h

B.5 Besprechungen

- Die offiziellen Besprechungen des Teams mit den Betreuern finden jeweils dienstags um 08:30 im Gebäude 8 der HSR statt. Dabei werden die Resultate der vorigen Woche und offene Fragen des Projektteams diskutiert.
- Das Team sendet den Betreuern am Montag jeweils die relevanten Dokumente für die Besprechungen am Dienstag.

B.6 Arbeitsabläufe

Das Team trifft sich offiziell an 3 Tagen pro Woche:

- Montag: Überprüfen und Senden der Dokumente für die Dienstagsbesprechungen, sowie Formulierung der offenen Fragestellungen
- Dienstag:
  - 08:30 Besprechung mit den Betreuern
  - 08:00 – 15:00: Gemeinsames Arbeiten am Projekt
- Freitag: 10:00-15:00: Gemeinsames Arbeiten am Projekt

Dies ergibt ca. eine gemeinsame Arbeitszeit von 9-12 Stunden. Die restliche Arbeitszeit wird von den Teammitgliedern frei auf die Woche verteilt, sodass die 17 Stunden pro Woche im Schnitt ungefähr erreicht werde

B.7 Arbeitspakete

In diesem Abschnitt werden die Arbeitspakete für die Construction-Phase beschrieben. Es werden dabei 2 grundsätzliche Kategorien unterschieden:

- **Arbeitspakete die die Funktionalität des Systems betreffen**
- **Arbeitspakete die das Projektmanagement betreffen**

B.7.1 Funktionalität

Pro Arbeitspaket werden der dazugehörige Use Case (siehe Anforderungsspezifikation), der Name und der Inhalt des Pakets beschrieben. Anfang der Construction wurde der Arbeitsumfang aller Pakete geschätzt. Bei den implementierten Paketen wurde die effektiv benötigte Zeit nachgetragen. Bei der Schätzung wurde keine Rücksicht darauf genommen, wieviel Zeit innerhalb der SA noch zur Verfügung steht, da es von Anfang klar war, dass nicht alle Anforderungen umgesetzt werden können. Die zu diesem Zeitpunkt noch verfügbare Zeit beträgt: $7\ Wochen \cdot\ 17h\ \cdot 2\ Personen =\ ca. 240h$
Anmerkung: Zum Zeitpunkt des Erstellens der Arbeitspakete war ein funktionsfähiger Prototyp vorhanden, d.h. gewisse Grundfunktionalitäten waren bereits vorhanden.

Nr.	Name	Use Case	Inhalt	Soll [h]	Ist [h]
1	Anbinden von MongoDB an das System	UC12	Komplettes Anbinden von MongoDB an das System, damit das Erstellen/Bearbeiten von Problemfällen möglich wird.	20	30.5
2	Manuelles Erstellen von Problemfällen	UC1, UC14	Erstellen einer Schnittstelle, über die ein Techniker manuell einen Troubleshooting-Prozess auslösen kann.	20	9
3	Normalisieren von Daten	UC2	Einbinden von napalm-logs/Pyangbind/ähnlichen Tools in das System, damit es möglich wird Troubleshooting-Prozesse auszulösen.	8	10.5
4	Manuelles Bearbeiten von Problemfällen	UC7, UC14	Erstellen einer Schnittstelle, über die ein Techniker manuell Problemfälle innerhalb des Systems bearbeiten kann (z.B. Abschliessen eines Falls).	6	0
5	Melden von häufigen Fehlern	UC9	Erstellen einer Logik, um zu erkennen, wenn ein Event sehr oft vorkommt, was auf ein grundsätzliches Problem im Netzwerk hinweisen könnte.	8	6
6	Netzwerk-Health anzeigen	UC11	Erstellen einer Schnittstelle und der dahinterliegenden Logik, um das Abfragen des aktuellen Netzwerkzustands zu ermöglichen	8	9
7	Techniker CRUD	UC13	Erstellen einer Schnittstelle um das Erstellen/Bearbeiten von Technikern die im Problemfall aufgeboten werden können zu ermöglichen.	10	9
8	Netzwerkdaten CRUD	UC 15	Erstellen einer Schnittstelle um das Bearbeiten der Netzwerkdaten zu ermöglichen.	20	10

9	Anbinden von Streaming-Telemetry	UC2	Anbinden eines Streaming-Telemetry-Tools an das System, damit Problemfälle auch durch Streaming-Telemetry-Daten ausgelöst werden können.	50	0
10	Event-Korrelation	UC 12	Erstellen einer Logik, um zu erkennen, dass zwei oder mehrere unterschiedliche Events bzw. Events von unterschiedlichen Geräten, den gleichen Problemfall beschreiben.	30	10.5
11	Workflow verwalten	UC 16	Erstellen einer Schnittstelle, um das Erstellen/Verwalten von Workflows zu ermöglichen, bzw. zu vereinfachen.	6	1
12 Prio1	Workflow: Gerät/Link fährt unterwartet runter	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um das unerwartete Runterfahren eines Links/Netzwerkgeräts zu behandeln.	18	50
13	Workflow: STP-Probleme	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um STP-Probleme zu behandeln.	17	0
14 Prio1	Workflow: Routing-Probleme	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um Routing-Probleme zu behandeln.	17	0
15 Prio2	Workflow: OSPF-Probleme	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um OSPF-Probleme zu behandeln.	12	22
16 Prio2	Workflow: BGP-Probleme	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um BGP-Probleme zu behandeln.	12	0
17	Workflow: EIGRP-Probleme	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um EIGRP-Probleme zu behandeln.	14	0
18 Prio1	Workflow: CDP-Probleme LLDP	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um CDP-Probleme zu behandeln.	12	0
19	Workflow: LACP-Probleme	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um LACP-Probleme zu behandeln.	12	0
20	Workflow: Flapping MAC-Adressen	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um Flapping MAC-Adressen zu behandeln.	10	0
21 Prio3	Workflow: langsame Links	UC3, UC4, UC6, UC8, UC12	Erstellen eines Workflows um langsame Links zu erkennen (Streaming-Telemetry) und zu behandeln.	12	0
			Total:	**322**	**167.5**

B.7.2 Projektmanagement

Nr.	Name	Inhalt	Soll [h]	Ist [h]
22	Projektplan überarbeiten	Aktualisieren der Dokumentation inkl. Erstellen und Schätzen der Arbeitspakete.	6	18.5
23	Anforderungsspezifikation überarbeiten	Aktualisieren der Dokumentation.	3	6
24	Domainanalyse überarbeiten	Aktualisieren der Dokumentation.	3	4.25
25	Architekturdokument überarbeiten	Aktualisieren der Dokumentation.	3	11.5
26	Technische Risiken überarbeiten	Aktualisieren der Dokumentation.	1	0.75
27	Systemtests durchführen	Durchführung der in der Elaboration definierten Tests.	3	2.5
29	Unit Tests schreiben		10	2
28	Testdokumentation erstellen/überarbeiten	Dokumentation der durchgeführten Tests erstellen, inkl. der daraus abgeleiteten Massnahmen	1	3.75
29	Glossar erstellen	Erstellen eines Glossars	4	1.5
30	Poster erstellen	Erstellen eines Posters	2	2
31	Abstract schreiben	Schreiben des Abstracts	1	1.75
32	Technischer Bericht schreiben	Schreiben des Technischen Berichts	2	3.5
33	Installationsanleitung schreiben	Schreiben der Installationsanleitung (wie setzt man das System auf)	6	6.75
34	Benutzerdokumentation schreiben	Schreiben der Benutzerdokumentation (wie bedient man das System, wie erweitert man das System	6	0.75
35	Persönlicher Bericht schreiben	Schreiben des persönlichen Berichts	2	11.05
36	Wöchentliche Besprechungen	Durchführen der Wöchentlichen Besprechungen die je zwischen 30-60 Minuten dauern	10	13
38	Allgemeines Projektmanagement	Arbeitspaket um Zeit zu erfassen die nicht direkt einem anderen Arbeitspaket zugeordnet werden kann	10	31.25
		Total:	**73**	**120.8**

B.8 Risikomanagement

B.8.1 Risiken

Die Risikoplanung ist im Anhang zu finden.
Auf der ersten Seite sind folgende Informationen zu finden:

- Risikobeschreibung
- Max. Schaden: Geschätzter Zeitverlust, sollte das Risiko eintreten
- Eintrittswahrscheinlichkeit: Geschätzte Wahrscheinlichkeit, dass das Risiko eintritt
- Gewichteter Schaden: Schaden der aufgrund von eingetretenen Risiken zusätzlich aufgewendet werden muss
- Verhalten beim Eintreten: Massnahmen die ergriffen werden, sollte das Risiko eintreten

Je nach dem zu welchem Zeitpunkt ein Risiko eintritt, wird der Schaden grösser oder kleiner. Der gewichtete Schaden berücksichtigt dies.
Auf der zweiten Seite werden die eingetretenen Risiken erfasst.

B.8.2 Umgang mit Risiken

Sollten Probleme auftreten, wird die dafür aufgewendet Zeit im Risikodokument niedergeschrieben. Die Zeit wird entweder als Verlust (gegen Ende des Projekts nicht anders möglich) akzeptiert, oder wird durch zusätzlichen Mehraufwand des Projektteams kompensiert.

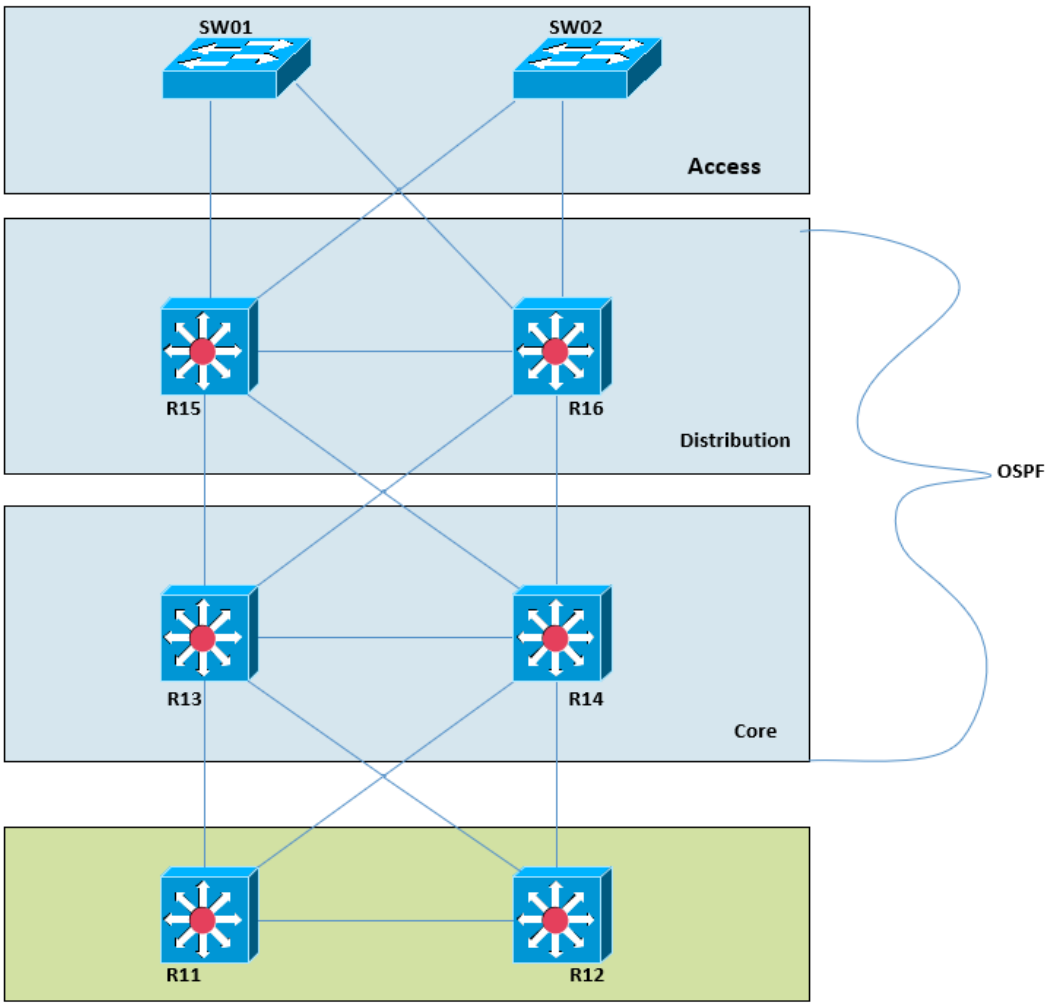
B.9 Infrastruktur

B.9.1 Entwicklungswerkzeuge

- PyCharm[1]: für Python-Code
- Vim[2]: zum Editieren von Text direkt auf dem OATS Server
- Vagrant[3]: zum Testen einer Lightweight-Version von OATS
- Microsft Office[4]: Excel & Word, zum Erstellen der Dokumentation & Zeiterfassung
- Git/Github[5]: Versionskontrolle
- Astah[6]: zum Erstellen von UMLs

B.9.2 Testumgebung

Folgendes Netz wurde durch die Betreuer aufgesetzt und durch das Projektteam für die Entwicklung von OATS benutzt:



Die Router und Switches befinden sich alle im gleichen Subnetz. Zudem sind alle über ein Management-Netz erreichbar, in welchem sich auch der Server auf dem OATS läuft befindet.

[1] https://www.jetbrains.com/pycharm/
[2] http://www.vim.org/
[3] https://www.vagrantup.com/
[4] https://products.office.com/de-ch/home
[5] https://github.com/
[6] http://astah.net/de

B.10 Testing

Gegen Ende der Construction Phase wurde OATS einem Systemtest unterzogen. Die Resultate sind befinden sich im Anhang abgelegt.

B.11 Qualitätsmassnahmen

B.11.1 Dokumentation

Sämtliche Dokumente werden in ein eigenes Git-Repository (separat vom Code) eingecheckt und deren Qualität wird durch Peer-Reviews sichergestellt.

B.11.2 Zeiterfassung / Arbeitspakete

Es wurde bewusst kein umfassendes Tool wie Redmine oder Jira benutzt um die Zeiterfassung bzw. die Arbeitspakete zu verwalten. Dies wurde direkt in Excel erledigt.
Der Grund dafür liegt im grossen Zeitaufwand der benötigt wird um ein solches Tool aufzusetzen, der sich unserer Meinung für 2 Teammitglieder nicht lohnt. Vor allem weil sich das Team täglich trifft und dadurch immer über das direkte Gespräch über den aktuellen Stand des Projekts informiert wird.

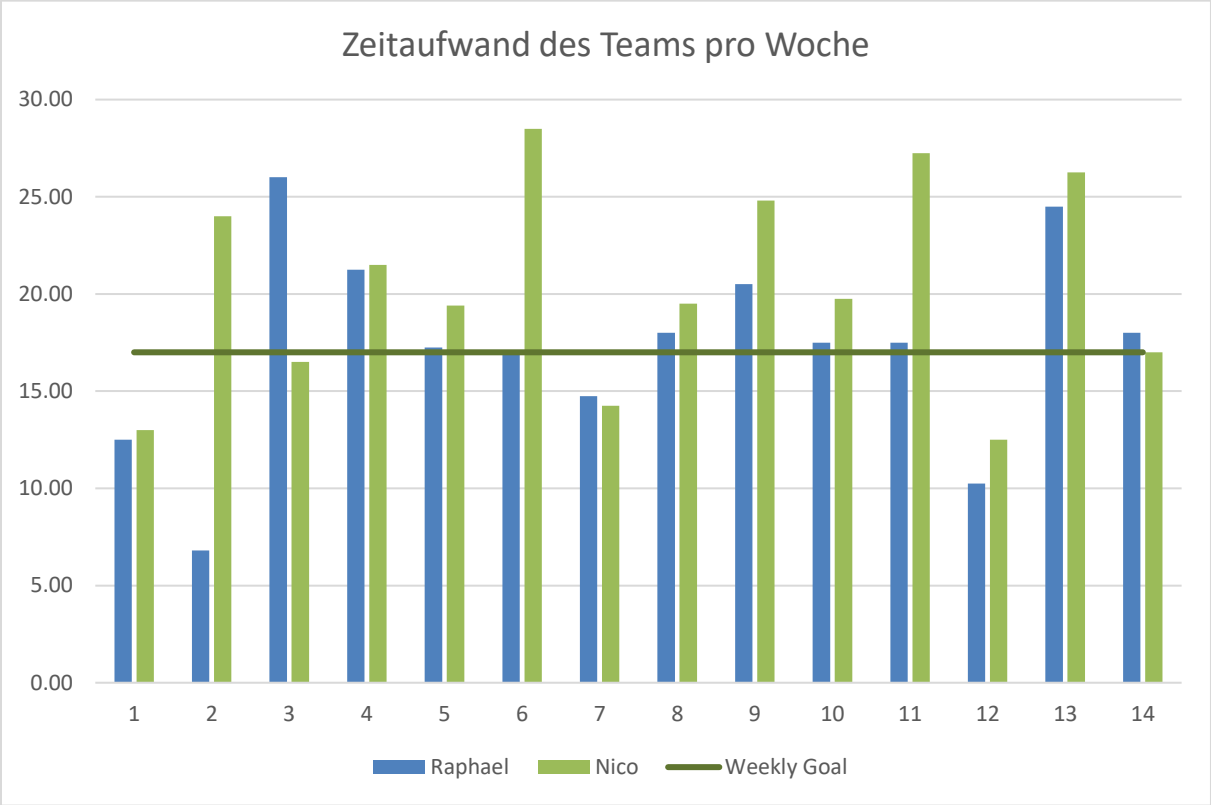
B.11.3 Coding Guidelines / Entwicklung

Als Coding Guideline während der Entwicklung wurde der in PyCharm integrierte Coding Guide PEP8[7] verwendet.

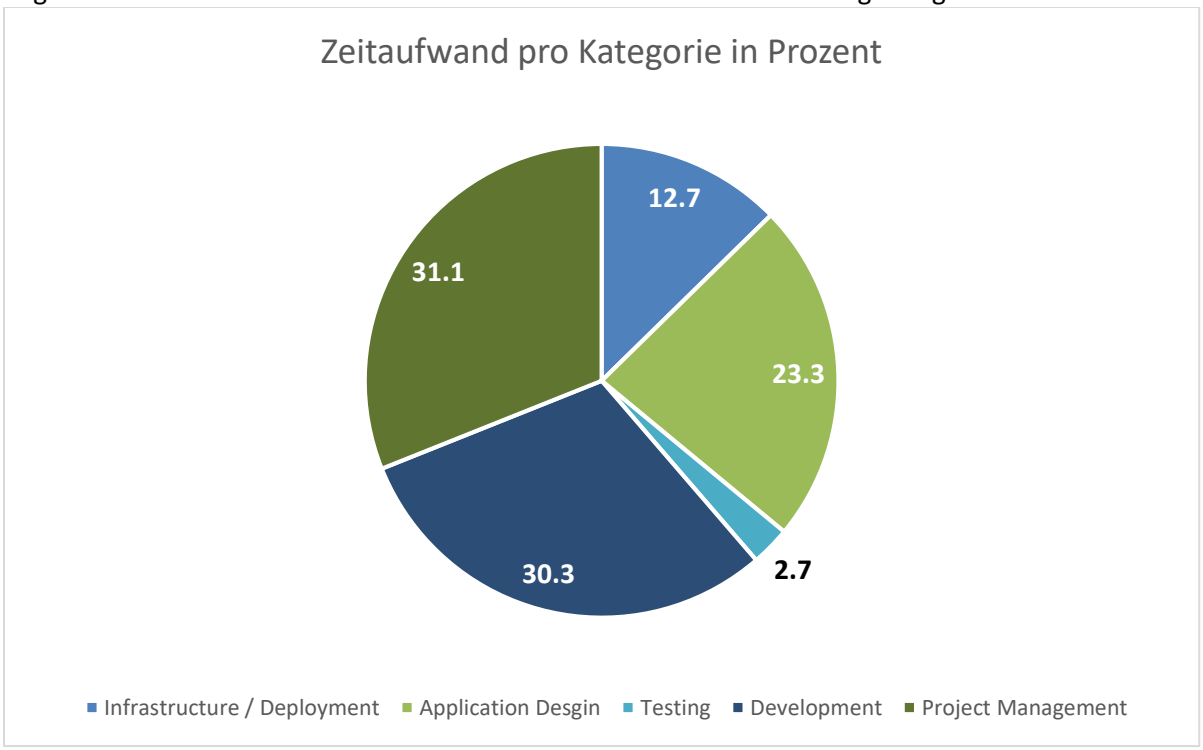
[7] https://www.python.org/dev/peps/pep-0008/

Anhang C: Zeitabrechnung

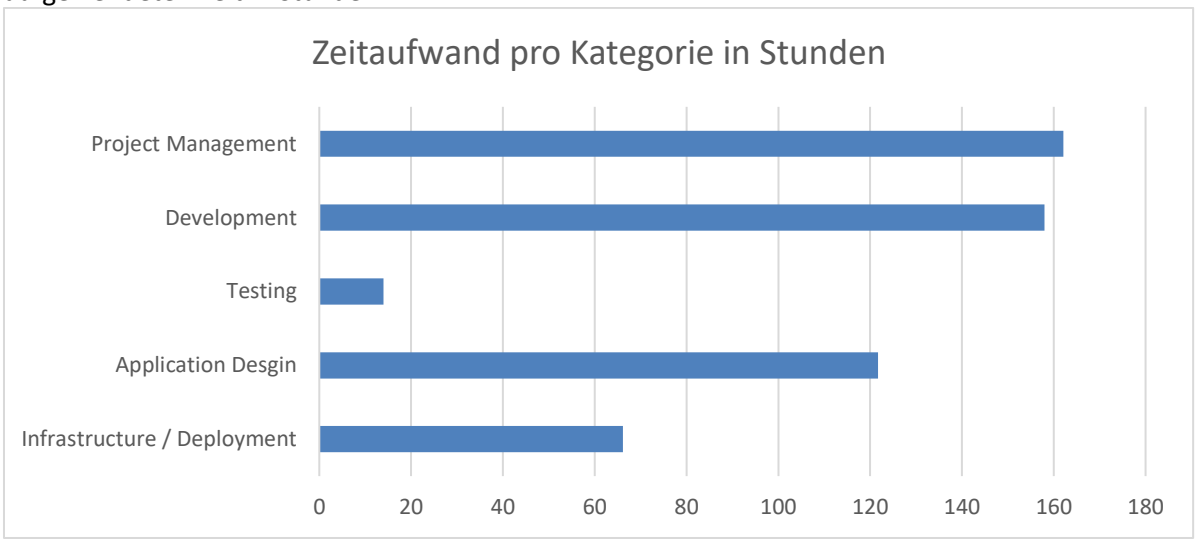
Die erste Grafik der Zeitabrechnung zeigt den Zeitaufwand der beiden Projektmitglieder pro Woche. Zusätzlich ist ein Balken mit dem Weekly Goal eingetragen. Insgesamt wurden 526 Stunden für die komplette Studienarbeit aufgewendete was leicht mehr ist als die vorgegebenen 480 Stunden. Auffällige Ausreisser in den einzelnen Wochen ist hier als erstes die Woche zwei, in welcher Raphael krankheitsbedingt fast eine komplette Woche ausfiel. Des Weiteren wurde in der Woche zwölf etwas weniger Zeit aufgewendet, da hier das Abschliessende Programmieren, sowie der Code Cleanup weniger Zeit als erwartet benötigte.



Die zweite Grafik zeigt den Prozentualen Zeitaufwand des Projekts pro Kategorie. Mit etwas mehr als 31 Prozent benötigte das Projektmanagement am meisten Zeit, hierzu gehören die wöchentlichen Sitzungen sowie das Schreiben der Dokumente. Mit 30 Prozent ist das eigentliche Development die zweitgrösste Kategorie, hierzu gehörte das Programmieren der Arbeit. Für das Applikation Design wurde 23 Prozent der Zeit aufgewendet, hierzu gehörten die Analyse der möglichen Tools welche man verwenden könnte, sowie die theoretische Ausarbeitung der Architektur. Das Aufsetzen der Infrastruktur benötigte etwa zwölf Prozent der aufgewendeten Zeit, hierzu gehörten Arbeiten wie das Installieren der verschiedenen Tools, das Aufsetzen des Prototyps, sowie das Einrichten der Testumgebung. Die letzte Kategorie war das Testing, hier wurden nur etwa drei Prozent der Zeit aufgewendet, Hierzu gehören das Schreiben der Unit Tests, sowie die Durchführung der Systemtests. Die Zeit hier ist bewusst kürzer da für die eingesetzten Tools bereits viele Unit Tests existieren (von der Community geschrieben), sowie die Tatsache dass gewisse Funktionen nicht mit Unit Tests abgedeckt werden konnten da diese eine Funktionierende Netzwerkumgebung voraussetzen.



Die letzte Grafik zeigt erneut den Zeitaufwand für die einzelnen Kategorien jedoch mit der effektiv aufgewendeten Zeit in Stunden.



Anhang D: Technische Risiken

D.1 Risikomanagement

Projekt:		Automatisiertes Troubleshooting	Dieses Dokument beschreibt die spezifischen Projektrisiken, welche bei der Durchführung auftreten könnten. Allgemeine Risiken (wie Krankheit, Unfall etc.) werden hier nicht abgebildet. Der gewichtete Schaden, ist die aufgrund von eingetretenen Risiken zusätzliche erwartete Zeit die aufgewendet werden muss. Er wird am Projektende mit dem eingetretenen Schaden verglichen.				
Erstellt am:		27.09.2017					
Autor:		Nico Vinzens					
Gewichteter Schaden:		25.8					
Nr	**Titel**	**Beschreibung**	**max. Schaden [h]**	**Eintrittswahrscheinlichkeit**	**Gewichteter Schaden**	**Vorbeugung**	**Verhalten beim Eintreten**
R1	Inkompatible Tools	Verschiedene Tools sind nicht miteinander kompatibel	25	20%	5	Gründliche Recherche der Tools inklusive Details wie z.B. verwendete Datenformate.	Inkompatible Komponenten mit existieren Tools oder einer Eigenlösung ersetzen
R2	Fehlerhaftes/nicht verstandenes Tool	Ein Tool welches ausgewählt wurde, funktioniert nicht wie erwartet, wurde falsch verstanden oder ist fehlerhaft	24	20%	4.8	Tools bevorzugen, die schon längere Zeit im Einsatz sind und über eine grosse Community verfügen. Stable-Releases verwenden, keine Beta-Versionen.	Überprüfen ob das Problem in akzeptabler Zeit von den Entwicklern behoben werden kann. Sonst wechseln von Tools bzw. Entwicklung einer Eigenlösung.

R3	Fehlerhafte Infrastruktur	Beim Aufsetzen der Infrastruktur wird ein Fehler begangen, der erst spät bemerkt wird (z.B. durch fehlendes Know-How). Dadurch passieren möglicherweise Folgefehler und die Infrastruktur muss nochmals neu aufgesetzt werden.	15	10%	1.5	Aufsetzen der Infrastruktur klar dokumentieren und überprüfen.	Kleinere Fehler können direkt behoben werden, bei grösseren Fehlern muss die gesamte Infrastruktur neu aufgesetzt werden.
R4	Schlechte Kommunikation	Die Kommunikation zwischen den Projektmigliedern oder mit den Betreuern ist schlecht, was zu Fehlern bei der Arbeit führt.	8	20%	1.6	Besprechungen protokollieren, regelmässige Besprechungen zwischen den Teammitgliedern	Fehler beheben und Kommunikation verbessern.
R5	Dokumente- / Datenverlust	Daten werden unbewusst gelöscht, überschrieben	4	10%	0.4	Einsatz von Versionierungstools	Wiederherstellen des Backups, Commit
R6	Wechselnde Anforderungen	Neue oder ständig wechselnde Anforderungen	30	5%	1.5	Klare Abgrenzung des Scopes und gute Planung	Anforderungen analysieren und allenfalls einplanen
R7	Fehlerhafte dependencies	Dependencies, auf die das System aufbaut sind fehlerhaft.	30	20%	6	Recherche der dependencies, stable Versionen benutzen	Rücksprache mit Entwicklern der dependency um Fehler zu beheben.

R8	Anforderungen sind komplexer als angenommen	Komplexität wurde unterschätzt. Der angenommene Zeitaufwand übersteigt die Planung um ein vielfaches.	50	10%	5	Gute Recherche	Rücksprache mit Betreuer und Team über weiteres Vorgehen, evtl. verringern des Funktionsumfangs
Summe			**186**		**25.8**		

D.2 Eingetretene Risiken

Risiko No.	Beschreibung	Grund	Zusätzlich aufgewendete Zeit [h]	Weiteres Vorgehen
R4	Die Kommunikation zwischen den Projektmigliedern oder mit den Betreuern ist schlecht, was zu Fehlern bei der Arbeit führt.	Projektteam und Betreuer hatten unterschiedliche Vorstellungen bei der Projektorganisation. Das Projektteam hätte Nachfragen müssen, was die Erwartungen der Betreuer sind.	6.00	MS2 wurde nicht erfüllt, der Milestone wird um eine Woche verschoben und es wird eine Analyse durchgeführt.
R7	Dependencies, auf die das System aufbaut sind fehlerhaft.	Für Napalm wurde eine neue Version released (2.0), Salt hatte dies nicht direkt miteinbezogen, es musste zusätzlich Zeit aufgewendet werden, damit der Prototyp wieder funktionierte.	4.00	Fehler wurde behoben, keine speziellen Massnahmen

Der eingetretene Schaden umfasst 10h, ist also tiefer als die gewichteten 25.8 Stunden.