Studienarbeit HS21

NUTS

Studiengang Informatik
OST - Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Autoren:	Severin Grimm Marco Martinez
Version:	23. Dezember 2021
Betreuer:	Prof. Beat Stettler Méline Sieber Lukas Murer

Abstract

Einleitung

Netzwerke sind elementare Bestandteile jedes Unternehmens, die immer wieder von Änderungen betroffen sind. Um grössere Probleme vorzubeugen, wird nach Änderungen manuell von Netzwerkingenieuren getestet. Das Problem dabei ist, dass das Know-How oftmals bei den Netzwerkingenieuren liegt und das menschliche Fehler beim Testen vorkommen. Die Automatisierung von Netzwerktests versucht dieses Problem zu eliminieren.

Ein Produkt für die Automatisierung von Netzwerktests ist NUTS (NetTowel Network Unit Testing), dass vom INS (Institue of Networked Solutions) entwickelt wird. NUTS ist ein pytest Plugin, dass mithilfe der Open-Source Frameworks Nornir und Napalm es ermöglicht, Unit-Tests in einem Netzwerk auszuführen. Diese Tests werden in einer Testdefinitionssprache in YAML definiert und von NUTS herstellerunabhängig ausgeführt.

Ziel

Die vorliegende Arbeit befasst sich mit dem Testing von Netzwerken durch NUTS und wie dieses Produkt in der Industrie verwendet werden kann. Es sollen mit Industriepartnern Kundenanforderungen evaluiert werden, ob Interesse an automatisiertem Testing vorhanden ist. Auch wie aktuelles, meist manuell ausgeführtes Testing automatisiert werden kann. Die gefundenen Tests sollen als sogenannte "Test-Bundles" in NUTS umgesetzt werden.

Zusätzlich sollen weitere Open-Source Netzwerkautomatisierungsframeworks analysiert werden, die von NUTS verwendet werden können.

Ergebnis

Die Interviews mit den Industriepartnern erlaubten es ein umfangreiches Testportfolio zu kreieren, das Test-Bundles enthält, die für die Industrie interessant sind. Es sind diverse Test-Bundles implementiert. Dazu gehört unter anderem ein Test, ob die richtigen VLANs einem Interface zugewiesen sind oder wie viele CDP Nachbaren ein Netzwerkgerät hat. Die meisten Tests im Testportfolio können nicht umgesetzt werden, da das eingesetzte Framework Napalm nicht die benötigten Daten von den Netzwerkgeräten liefern kann. Von den weiteren analysierten Netzwerkautomatisierungsframeworks lässt sich keines als umfassende Gesamtlösung einsetzen. Daher wurde für die Umsetzung weiterer Test-Bundles evaluiert, wie eine mögliche Architektur von NUTS aussehen sollte.

Künftig ist es sinnvoll NUTS so umzubauen, damit es nicht als pytest Plugin verwendet wird und nicht so stark vom Framework Napalm abhängig ist. Die Daten herstellerunabhängig von Netzwerkgeräten abholen sollte nicht von Napalm gelöst werden, sondern von NUTS selbst. Dies ist mit der aktuellen Struktur von NUTS nicht möglich und erfordert einen grösseren Umbau der Architektur.

Inhaltsverzeichnis

Aufgabenstellung

Ausgangslage

In der Software-Entwicklung ist es üblich, Änderungen oder Erweiterungen am Code automatisiert zu testen, bevor der die neue Version in die Produktion übernommen wird. Dabei testet man nicht nur die geänderten Code Bestandteile, sondern möglichst sämtlicher Code. Man spricht dabei von einer Testabdeckung, welche möglichst gross sein sollte (z.B. 90%).

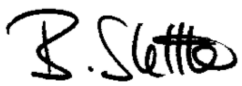
Im Netzwerkumfeld werden nach Änderungen dagegen meist nur minimale Tests durchgeführt. 2-3 show Befehle auf dem veränderten Gerät oder ein ping reichen ja meist auch aus, um die unmittelbar vorgenommene Änderung zu überprüfen. Indirekten Fehler wie z.B. an höheren Protokollen oder Services können dadurch aber unentdeckt bleiben.

In einer früheren Arbeit wurde eine Software genannt «Network Unit Tests NUTS» entwickelt, welche es erlaubt, automatisierte Tests an einem Netzwerk durchführen zu können. Dieses umfasst bisher Device Tests (z.B. show interface) sowie ARP und Routing Tests (z.B. show ospf nei). Dafür wird das Python Testing Framework "Pytest" verwendet sowie Nornir zur Speicherung der Toplogie und zur Ausführung der Tests auf der Infrastruktur. Die Auswertung der Tests wird bei der Ausführung auf der Konsole angezeigt und zusätzlich in einem Testreport für die spätere Ansicht gespeichert. Die Programmausführung kann manuell oder automatisch über einen Deployment-Prozess gestartet werden.

Aufgabe

In dieser Arbeit soll die Problematik des automatisierten Testing anhand von 2-3 echten Firmenumgebungen analysiert und vertieft werden. Dabei sollen die in Firmen bisher benutzten Test-Cases bei Inbetriebnahme neuer Komponenten, bei Changes und beim Life-Cycle von Geräten erfasst und kritisch hinterfragt resp. mit zusätzlichen Testvorschlägen erweitert werden. Viele Firmen sind zudem daran, vermehrt neue SDN basierte Netzwerk-Technologien wie (z.B. SD-Access, SD-WAN, ACI) einzusetzen. Dabei wird der Verkehr der Kunden in Overlays (z.B. VPN, VXLAN) transportiert, welche durch die Infrastruktur Tests nicht direkt erfasst werden. Hierfür braucht es neue Testideen. Als erstes Resultat der Arbeit soll eine umfassende Liste von sinnvollen und praxisnahen Test-Cases resultieren. Ebenfalls zu analysieren ist, wie ein Testtool wie NUTS in die Systemumgebung und Arbeitsprozesse der Firmen eingebunden werden kann – also wie wird die Topologie übernommen (damit diese nicht erneut von Hand in NUTS erstellt und gepflegt werden muss), wie möchte der Kunde die Testausführung starten und in welcher Form möchte er die Resultate geliefert erhalten.

Im Entwicklungsteil soll die bisherige Software um möglichst viele weitere Tests erweitert werden. Beispielsweise Topologie Tests (z.B. Layer 2: CDP, LLDP, Layer 3: BGP), Traffic Tests (RTT, Jitter Messungen) und Service Tests (DHCP, DNS) sowie um neuen Testideen zur Überprüfung von SDN basierten Infrastrukturen und Services. Idealerweise wird die Funktionalität der so entstandenen Software in einem zeitgemässen Testlabor (z.B. bei einem Kunden) getestet.



Prof. B. Stettler
Rapperswil, den 1.10.2021

Management Summary

Ausgangslage

Das Netzwerk ist ein elementarer Bestandteil jedes Unternehmens. Dieses ist meist statisch und bei Änderungen können grössere Probleme auftreten. Die Automatisierung im Netzwerkbereich versucht dieses Problem zu eliminieren. Die grössere Komplexität eines automatisierten Netzwerkes soll durch das Netzwerktesting entschärft werden.
Das Institute for Networked Solution der Ostschweizer Fachhochschule hat ein Produkt NUTS (NetTowel Network Unit Testing System) für das automatisierte Testing von Netzwerken entwickelt. NUTS ist ein pytest Plugin welches Unit-Tests, definiert im YAML Format, in einem Netzwerk auszuführen kann. Die Unit-Test werden mit pytest als Python Code bereitgestellt und "Test-Bundles" genannt. Die zu testenden Netzwerkgeräte werden von NUTS mit dem Open-Source Framework Nornir verwaltet. Nornir besitzt ein Inventar im YAML Format, wo die nötigen Daten der Netzwerkgeräte, wie Hostname, SSH Zugriffsport und Adresse eingetragen werden. Durch diese Korrelation der Daten mit der Testsprache und den Daten im Inventar kann eine SSH Verbindung zu den Netzwerkgeräten aufgebaut und die benötigten Daten abgeholt werden. Zuständig für das herstellerunabhängige Abholen der Daten ist das Open-Source Framework Napalm. Dieses kann auch als Plugin in Nornir verwendet werden, wie es in NUTS geschieht. Damit profitiert Napalm vom Inventar von Nornir.

Die aktuelle Anzahl an Test-Bundles ist klein. Ziel dieser Studienarbeit ist es, neue mögliche Test-Bundles mit Industriepartnern zu evaluieren. Vom entstanden Testportfolio sollen möglichst viele dieser Test-Bundles implementiert werden. Zusätzlich sollen weitere Frameworks analysiert und deren Verwendungszweck in NUTS evaluiert werden.

Vorgehen

Da Testing im Netzwerkbereich weniger bekannt ist werden im ersten Schritt die Test-Stufen der Software Entwicklung für Netzwerktests definiert.

Für die Erstellung des Testportfolios werden Interviews mit den Informatikabteilungen dreier Industriepartnern, der Migros, der MAN Energy Solutions und der Stadt Zürich, durchgeführt. Dieses Testportfolio wird gleichzeitig mithilfe der Industriepartnern priorisiert. Für die einzelnen Test-Bundles wird die mögliche Umsetztung mit dem aktuell benutzten Framework Napalm evaluiert. Bei technischen Limitation von Napalm werden zusätzliche Frameworks und Methoden analysiert, welche die Umsetztung ermöglichen. Es wird ausserdem evaluiert, wie diese Industirepartner ihre Netzwerkgeräte inventarisieren und wie diese in das Inventar von NUTS übernommen werden können.
Bei vorhandener Zeit werden möglichst viele dieser Test-Bundles in NUTS implementiert. Die Tests werden als pytest Unit-Test in Pyton Code geschrieben.

Ergebnisse

In dieser Studienarbeit konnten die Netzwerktesting Stufen definiert und desweiteren ein umfangreiches Testportfolio für weitere Test-Bundles erstellt werden. Diverse mögliche Test-Bundles wurden mit Python in NUTS implementiert.

Zusätzlich wurden weitere Netzwerkautomatisierungsframeworks analysiert und deren Verwendungszweck für NUTS definiert. Dabei wurde festgestellt, dass aktuelle Open-Source Frameworks nicht die benötigten Funktionen für NUTS liefern. Das Problem liegt bei den Daten, die herstellerunabhängig von den Netzwerkgeräten abgefragt werden können. Als Alternative zu bestehenden Frameworks konnte durch einen "Proof-of-Concept" eine neue Architektur evaluiert werden, um dieses Problem zu lösen.

In der weiteren Entwicklung von NUTS sollte die Architektur überarbeitet werden. Dabei soll darauf geachtet werden, dass NUTS nicht mehr so eng mit pytest verknüpft ist. Dadurch hat der Software-Code weniger interne Kopplung und wird Transparenter. Ein weiterer bedeutsamer Punkt ist die Verbesserung des herstellerunabhängigen Abholen der Daten von den Netzwerkgeräten. Dieses sollte durch NUTS abgehandelt werden. Dabei kann man zum Beispiel mit dem Nornir-Plugin Netmiko die Daten von Netzwerkgeräten abfragen und mittels eines Preproccessing den herstellerunabhängigen Teil für die Testsprache in YAML bereitstellen. So kann sichergestellt werden, dass neue Test-Bundles problemlos implementiert werden können und nicht abhängig von Open-Source Frameworks wie Napalm sind.

In einem letzten Schritt wurde evaluiert, ob NUTS weiterhin eine Client-Applikation bleiben soll. Die Analyse hat ergeben, dass NUTS als Server Applikation sinnvoller ist.

I. Technischer Bericht

1 Einleitung

1.1 Problemstellung

Computernetzwerke bestehen aus zahlreichen Komponenten, wie Router, Switches, Access Points und Endgeräte. Dadurch können Netzwerke, insbesondere Enterprise Netzwerke, komplex werden. Jede Netzwerkkomponente hat eine Konfiguration und Aufgabe und bilden damit ein funktionierendes Netzwerk. Durch neue Frameworks und die Bereitstellung von APIs durch die Hersteller, ist es in den letzten Jahren einfacher geworden diese Netzwerkgeräte mit Software zu konfigurieren. Jedoch wird das Testen dieser Geräte oftmals immer noch durch Netzwerkingenieure manuell erledigt. Durch die komplexen Netzwerke, kann diese Überprüfung fehleranfällig sein. Schleicht sich so ein Fehler ein, kann das komplette Netzwerk die Verfügbarkeit nicht garantieren. Netzwerke sind heutzutage nicht mehr nur statisch. Routing Protokolle suchen dynamisch einen Weg durch das Netzwerk und durch die kommenden SDN (Software-Defined-Networks) Netzwerke wird das noch verstärkt.

Bei der Entwicklung von Software wird schon seit längerem zur Entwicklungszeit getestet. Dies wird mit sogenannten Unit-Tests durchgeführt. Das sind Tests, die eine spezifische Komponente ("eine Unit") in der Applikation automatisiert testen. Die einzelnen Komponenten werden ausgeführt und es wird überprüft, ob die Ausgabe auch einem erwarteten Ergebnis entspricht. Dies ist besonders hilfreich, wenn neue Funktionen entwickelt werden oder bestehende Funktionen verbessert/geändert werden. Bevor eine Änderung veröffentlicht wird, können die einzelnen Komponenten automatisiert nochmals getestet werden. Dadurch kann bestätigt werden, dass die Applikation noch nach Anforderungen funktioniert. Bei fehlschlagenden Tests, können Verbesserungen durchgeführt werden.

Für Netzwerktests ist dies nicht ganz so problemlos. Protokolle wie OSPF, BGP, Spanning Tree, etc. sind standardisiert und können dadurch auch zwischen Geräten von verschiedenen Herstellern benutzt werden. Die Konfiguration, Implementation und die Überprüfung ist jedoch herstellerspezifisch. Die zur Verfügung gestellten APIs der Hersteller und die Kommandos auf den CLIs sind unterschiedlich. Durch die Absenz einer einheitlichen Konfigurationssprache wird es erschwert eine Applikation zu entwickeln, die für alle Hersteller funktioniert.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist es, neue Test-Bundles aus realen Use-Cases für NUTS zu entwickeln, und das automatisierte Testen von Netzwerken mit NUTS in realen Firmenumgebungen zu evaluieren. Die Migros, die Stadt Zürich und die MAN Energy Solutions haben Ihr Interesse an NUTS bekundet und würden sich für Interviews zur Verfügung stellen.

Analyse

Es sollen Interviews mit den Netzwerkabteilungen dieser drei Firmen geführt werden. Durch diese Interviews sollen aktuelle Test-Cases kritisch analysiert und mögliche NUTS Test-Bundles definiert werden. Das Ziel ist eine umfassende Liste mit praxisnahen Netzwerktests, die etablierte und neue Technologien enthält. Aus dieser Liste mit Netzwerktest soll ein Testportfolio zusammengestellt werden, das in NUTS umgesetzt wird. Des Weiteren sollen gängige Netzwerkautomatisierungsframeworks analysiert werden und deren Verwendungszweck in NUTS definiert werden.

Zusätzlich soll evaluiert werden, wie ein automatisiertes Testing Tool wie NUTS in die Arbeitsprozesse von Firmen integriert werden kann. Dabei soll Fokus darauf gelegt werden, wie bestehende Topologien in NUTS aufgenommen werden können.

Entwicklung

In der Entwicklungsphase sollen die Netzwerktests aus dem Testportfolio zu NUTS hinzugefügt und getestet werden. Wenn möglich, sollen die neuen Test-Bundles in realen Szenarios mit NUTS bei Kunden getestet werden.

1.3 Herausforderungen

Die grösste Herausforderung ist es, aussagekräftige Tests für ein komplexes Netzwerk zu schreiben. Da nun nicht mehr nur Netzwerkgeräte getestet werden müssen, sondern auch Endgeräte, werden die Tests auch komplexer. Dabei scheint die Limitation auf den entsprechenden Geräten zu sein, da viele dieser Geräte IoT Geräte sind.

1.4 Vorarbeit

In einer Studienarbeit im Frühlingssemester 2020 wurde die Applikation "NUTS" von Mike Schmid und Janik Schlatter entwickelt. Mit NUTS ist es möglich, Netzwerktests anhand einer Testdefinition in YAML durchzuführen. Die komplette Applikation ist in Python geschrieben und lässt sich auf allen Geräten, die Python3 unterstützen, ausführen. Im Hintergrund werden die Frameworks Nornir und Napalm verwendet. Mit Nornir ist es möglich, Befehle an verschiedene Netzwerkgeräte zu senden, und das Resultat als Rückgabewert zu verwalten.

Die Vorarbeit wurde nachträglich vom INS weiterentwickelt und kann auf Github[1] gefunden werden. Das weiterentwickelte NUTS nimmt die Rückgabewerte der Geräte und vergleicht diese als pytest mit den zu erwartenden Werten. Die zu testenden Geräte werden in einem Nornir Inventory gehalten. Es gibt bereits einige Test-Bundles, die produktiv in einem Netzwerk eingesetzt werden können.

[1] Link zum Repository: https://github.com/INSRapperswil/nuts

II. Analyse

1 Einführung

In dieser Analyse werden aktuelle und neue Netzwerktests für NUTS angeschaut.

2 Definitionen

2.1 Wörter

Dadurch, dass NUTS eine Testing Applikation für Netzwerk Unit-Tests ist, kann es schnell zu Verwirrungen kommen beim Wort "Test". Mit folgenden Definitionen[2] soll Klarheit mit diesen Begrifflichkeiten geschaffen werden.

Wort	Definition
Test	Eine Pytest Test Funktion
Test-Bundle	Eine Reihe von Tests, die logisch zusammengehören. Werden von NUTS auf Geräten ausgeführt.
Test Instanz	Die Ausführung eines Test-Bundles auf einem spezifischen Gerät.
Test Klasse	Eine spezifische Pytest Test Klasse, die gebraucht wird, um Test-Bundles auszuführen.
Selbsttest	Ein Unit Test, um die Test Klasse zu testen.
Testportfolio	Sammlung von Test-Bundles, kreiert aus Interviews mit Industriepartner

Tabelle 1: Definitionen Test

Des Weiteren gibt es noch Bibliothek und andere projektspezifische Wörter, die unten erläutert werden.

Wort	Definition
Inventar	Ein Nornir Inventar, das alle Informationen über die Netzwerkgeräte enthält.

Tabelle 2: Weitere Definitionen

2.2 Testing-Techniken

2.2.1 Testing in der Softwareentwicklung

In der Softwareentwicklung ist das Testing des Codes bereits weitgehend etabliert und definiert. Es gibt Abgrenzungen und Beschreibungen, wie die Teile einer Software getestet werden sollen und in welchem Ausmass. Um eine Analogie zwischen dem Testing in der Softwareentwicklung und dem Testing von Netzwerken herzustellen, muss zuerst angeschaut werden, wie das Testing in der Software Entwicklung funktioniert.

Test-Pyramide

In der Softwareentwicklung gibt es vier verschiedene Test-Stufen, auf die hier eingegangen wird. Zu notieren ist, dass es keine "eindeutige Wahrheit" gibt. Diese vier Test-Stufen sind eine gängige, verbreitete Variante für Testing im Software-Bereich. Es gibt jedoch noch weitere Stufen, auf die hier nicht eingegangen wird.

[2] Definitionen ZVG von Méline Sieber, Zugriff: 12.10.2021, [Sie]

Visuell kann man diese vier Stufen als Pyramide darstellen:

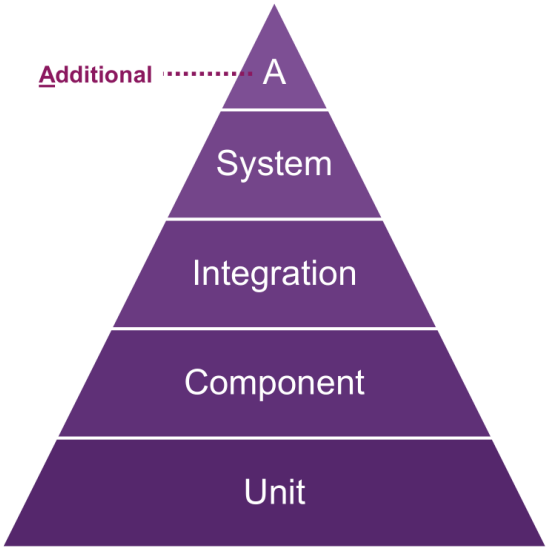


Abbildung 1: Test-Pyramide [3]

Tests weiter unten in der Pyramide, sollen mehr verwendet werden. Wandert man in den Test-Stufen nach oben, werden es immer weniger Tests. Grundsätzlich gilt:

> *Testen Sie alle relevanten Verhaltensweisen auf der niedrigstmöglichen Ebene, und nur die Verhaltensweisen welche verbleiben, auf der nächsthöheren Ebene.* [4]

Unit-Test

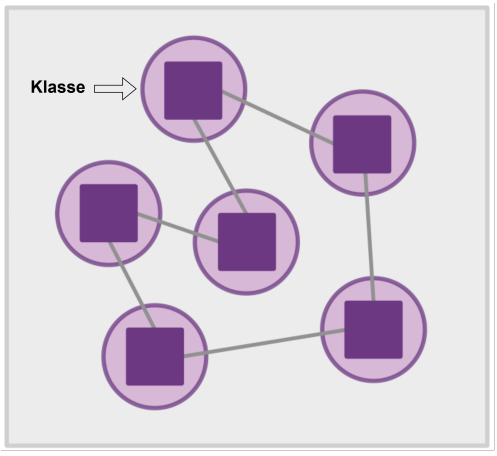


Abbildung 2: Unit-Tests[5]

Die Unit-Tests sind die Grundlage. Es sind viele, kleine Tests, die einzelne Einheiten testen. Einheiten sind in der Softwareentwicklung zum Beispiel Methoden einer Klasse. Es wird Überprüft, ob eine Methode bei einem Input

[3] Definition von Folie 14, Zugriff: 28.10.2021, [Käl]
[4] Prof. Dr. Farhad Mehta, SE1 HS20, Ostschweizer Fachhochschule
[5] Definition von Folie 12, Zugriff: 28.10.2021, [Käl]

auch den erwarteten Output zurückgibt, um zu verifizieren, dass die Methode funktioniert. Dadurch können bei jeglichen Änderungen am System die Unit-Tests ausgeführt werden und die Integrität der Komponente bestätigt werden. Weil ein Unit-Test nur einen kleinen Teil testet, ist dieser auch "günstig". Es können viele Unit-Tests in kurzer Zeit ausgeführt werden und einen grossen Teil des Systems testen. Daher sind sie auch die Grundlage und sollten am meisten eingesetzt werden.

Komponenten-Test

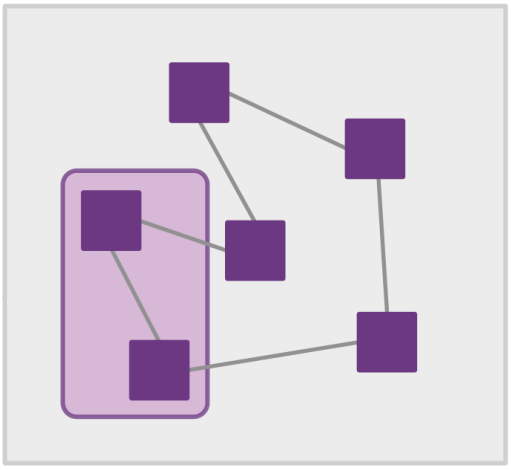


Abbildung 3: Komponenten-Tests[6]

Die nächst höhere Stufe sind die Komponenten-Tests. Diese testen Klassen, die Funktional zusammengehören. Das sind zum Beispiel Klassen, die sich um die Benutzerverwaltung kümmern. Komponenten-Tests sind gut um die Integrität von zusammengehörenden Teilen einer Software zu bestätigen. Dadurch können diese Komponenten auch einfacher ausgetauscht werden.

Integrations-Test

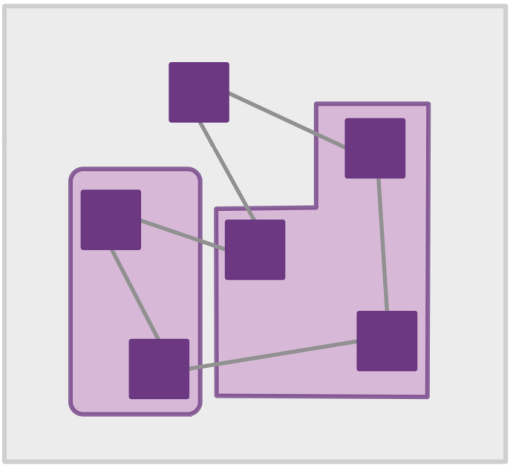


Abbildung 4: Integrations-Tests[7]

[6]Definition von Folie 12, Zugriff: 28.10.2021, [Käl]

Mit Integrations-Tests werden die Schnittstellen zwischen Komponenten getestet. Dadurch kann sichergestellt werden, dass auch die einzelnen logischen Komponenten miteinander funktionieren. Das kann zum Beispiel die Komponente für die Benutzerverwaltung sein, die mit der Komponente für den Datenbankzugriff interagiert.

System-Test

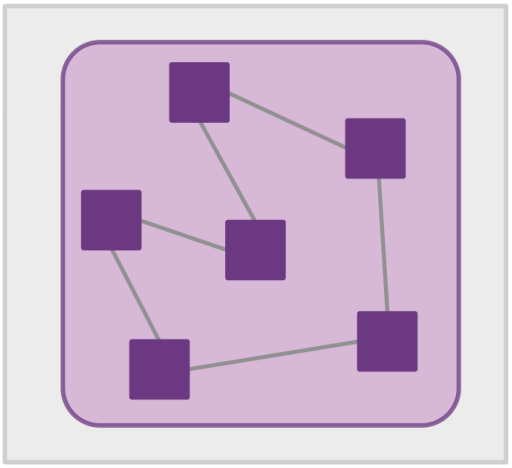


Abbildung 5: System-Tests[8]

Auf der obersten definierten Stufe der Pyramide sind die System-Tests. Es wird das komplette System getestet, wie es auch später im Einsatz sein wird. Diese werden oft manuell ausgeführt, können aber auch automatisiert werden.

2.2.2 Testing im Netzwerk

Hier werden die Stufen des Testing in der Softwareentwicklung auf Netzwerktests angewendet. Dabei wird versucht, die grössen der Test-Stufen ähnlich darzustellen. Das heisst, Unit-Tests als kleinste Tests bis zu System-Tests als grösste Tests.

Da keine perfekte Analogie von Software zu Netzwerken hergestellt werden kann, wurden in diesem Fall die Komponenten- und Integrations-Test als eine Stufe gewählt.

Unit-Tests

Unit-Tests im Netzwerk Bereich beziehen sich, wie in der Softwareentwicklung, auf die kleinst-mögliche zu testende Stufe. In diesem Fall sind dies die einzelnen Geräte wie Router, Switches, Access Points, Firewalls, und weitere. Die Test-Bundles beziehen sich auf Konfigurationen der Geräte und deren gesammelten Informationen im Netzwerk. Bei der Konfiguration kann man Testen, ob gewisse Funktionen aktiv sind, wie zum Beispiel Spanning-Tree bei einem Switch, die existierenden User oder ob die Interfaces die richtigen Attribute haben. Bei den gesammelten Informationen sind es Daten wie Routing-Einträge, Nachbar-Informationen über CDP und LLDP oder auch Umgebungsdaten wie die Temperatur des CPU.

Integration-Tests

Integrations-Tests im Netzwerk testen die Funktionalität zwischen den Geräten. Hier wird die Konnektivität der Geräte im einen Netzwerk getestet. Möglich tests sind zum Beispiel ein Traceroute im Netzwerk, ob die First Hop Redundanz von Router funktioniert oder ob sich ein Telefon auch wirklich aufs Voice-VLAN

[7] Definition von Folie 13, Zugriff: 28.10.2021, [Käl]
[8] Definition von Folie 13, Zugriff: 28.10.2021, [Käl]

verbindet. Es können auch interaktive Tests durchgeführt werden, ob sich zum Beispiel Routen anpassen, wenn gewisse Uplinks deaktiviert werden.

System-Tests

Einzelne Netzwerke, zum Beispiel an einem Firmenstandort oder ein Datacenter, werden als System angesehen. System-Tests testen die Funktionalität zwischen diesen Netzwerken. Dies können zum Beispiel MPLS Tests sein, die netzwerkübergreifende Verbindungen prüfen. Auch kann die Verfügbarkeit von Diensten wie DHCP und DNS, die von anderen Netzwerken zur Verfügung gestellt werden, getestet werden.

Einteilung von Test-Bundles

Dadurch, dass die Aufteilung in Netzwerken auf die logische Struktur eines Netzwerkes definiert ist, können gewisse Test-Bundles auch auf verschiedenen Test-Stufen benutzt werden. Ein Ping-Test ist ein gutes Beispiel. Es kann ein Ping von einem Gerät auf ein eigenes Interface gesendet werden, um zu überprüfen, ob die OSI Layer bis Layer 3 funktionieren. Das ist ein Unit-Test. Wird ein Ping auf ein anderes Gerät im gleichen Netzwerk ausgeführt, kann die Konnektivität bis Layer 3 zwischen zwei Geräten bestätigt werden. Das ist ein Integrations-Test. Wird ein Ping auf ein Gerät in einem anderen Netzwerk oder im Internet ausgeführt, kann die Konnektivität zwischen Netzwerken bestätigt werden. Das ist ein System-Test.

2.3 Definition Name "NUTS"

NUTS wurde seit der ersten Version als "Network Unit-Test" Software definiert. Da Unit-Tests jedoch klein sind und in der Definition nur einzelne Geräte testen können, wird die Funktionalität eingeschränkt. Besser ist es, wenn auch Integrations- und System-Tests durchgeführt werden können. Falls dies ermöglicht wird, was den Einsatzbereich beträchtlich vergrössert, wäre der Name von NUTS jedoch irreführen. Daher könnte noch überlegt werden, den Namen auf einen allgemeineren Begriff anzupassen.

3 Frameworks

NUTS ist ein pytest-Plugin das Nornir und die Nornir-Plugins Napalm und Netmiko verwendet. Um NUTS möglichst generisch und modular zu halten, ermöglicht NUTS verschiedene Frameworks einzubinden. Der Vorteil besteht bei Frameworks darin, dass die Arbeit nicht bei einem einzelnen anfällt sondern bei der Community, somit können Kosten zur Entwicklung einer Applikation drastisch gesenkt werden. Ebenfalls sind Frameworks in der Regel etwas reifer, optimierter und performanter, da meist viele Experten involviert sind.

3.1 Nornir

> *Nornir ist ein reines Python-Automatisierungs-Framework, das direkt von Python aus verwendet werden kann. Während die meisten Automatisierungs-Frameworks ihre eigene domänenspezifische Sprache (DSL) verwenden, mit der Sie beschreiben können, was Sie tun möchten, können Sie mit Nornir alles von Python aus steuern.* [9]

3.1.1 Verwendung von Nornir in NUTS

Nornir ist bereits ein bedeutsamer Bestandteil von NUTS. Nornir verwaltet das Inventar und bieten einen flexiblen Plugin Mechanismus. Nach etwas genauerer Analyse wurde der Entschluss gefasst, dass die Funktionalität von Nornir gut auf den Scope von NUTS passt und deshalb weiterhin ein Bestandteil davon bleiben soll.

3.1.2 Vor- und Nachteile von Nornir in NUTS

Vorteile

- Scopes der beiden Tools decken sich grossflächig
- einfache Inventarfunktion für NUTS
- kann parallel genutzt werden
- muss nicht exklusiv genutzt werden
- Open-Source

Nachteile

- relativ kleine Community im Vergleich zu Ansible / Saltstack
- hohe Komplexität

[9] Übersetzt mit deepl.com, Zugriff: 11.11.2021, [Tec]

3.2 Napalm

NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support) ist eine Python-Bibliothek, die eine Reihe von Funktionen zur Interaktion mit verschiedenen Netzwerkgeräte-Betriebssystemen über eine einheitliche API implementiert. NAPALM unterstützt mehrere Methoden, um sich mit den Geräten zu verbinden, Konfigurationen zu manipulieren oder Daten abzurufen.[10]

3.2.1 Verwendung von Napalm in NUTS

Napalm ist eine gute Grundlage für NUTS um nicht herstellerspezifisch agieren zu müssen. Eine Analyse ergibt, dass Napalm ebenso wie Nornir, ein essentieller Bestandteil von NUTS ist und darum auch beibehalten werden sollen. Was die Analyse ebenfalls gezeigt hat, ist das Napalm leider nur beschränkt agieren kann. Viele Funktionen sind nicht für alle Plattformen vorhanden. Wobei bei weitem nicht alle Plattformen unterstützt werden. Die Matrix ist ein guter Anfang, leider ist schnell klar geworden, dass NUTS noch weitere Lösungen finden muss, die sich nicht auf Napalm stützen.

	EOS	IOS	IOSXR	IOSXR_NETCONF	JUNOS	NXOS	NXOS_SSH
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_ipv6_neighbors_table	❌	✅	❌	❌	✅	❌	❌
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	✅	✅	✅	✅	✅	✅	✅
get_vlans	✅	✅	❌	❌	✅	✅	✅
is_alive	✅	✅	✅	✅	✅	✅	✅
ping	✅	✅	❌	❌	✅	✅	✅

Abbildung 6: Napalm Kompabilitätsmatrix [11]

[10] Übersetzt mit deepl.com, Zugriff: 11.11.2021, [Byea]

3.2.2 Vor- und Nachteile von Napalm in NUTS

Vorteile

- Open-Source
- einfaches Konzept
- gute Dokumentation

Nachteile

- es wird mit Kompabilitätsmatrizen gearbeitet
- limitierte Funktionalität
- Support für 4 OS

3.3 Netmiko

> *Bei der Netzwerkautomatisierung von Screen-Scraping-Geräten geht es in erster Linie darum, die Ausgaben von Show-Befehlen zu sammeln und Konfigurationsänderungen vorzunehmen.*
>
> *Netmiko zielt darauf ab, diese beiden Operationen durchzuführen, und zwar über eine sehr breite Palette von Plattformen hinweg. Es versucht, dies zu erreichen, während es von der Statuskontrolle auf niedriger Ebene abstrahiert (d.h. den Regex-Musterabgleich auf niedriger Ebene so weit wie möglich eliminiert).* [12]

3.3.1 Verwendung von Netmiko in NUTS

Netmiko bietet einen einheitlichen Weg wie sich NUTS auf alle Netzwerkgeräte verbinden kann. Deshalb ist Netmiko interessant für NUTS. Netmiko ist leichtgewichtig und bietet eine grundlegende SSH Funktion. Dabei zu beachten ist, dass Netmiko nur die Verbindung aufbaut. Die Abfrage muss vom Entwickler selbst definiert werden. Ebenso muss die Antwort auf die Abfrage geparsed werden und in ein sinnvolles Format überführt werden. Die Analyse hat ergeben dass Netmiko in NUTS eingesetzt werden soll, wenn kein anderes Framework die Aufgabe übernehmen kann.

3.3.2 Vor- und Nachteile von Netmiko in NUTS

Vorteile

- Open-Source
- Funktioniert auf den meisten Geräten
- wenig Komplexität

Nachteile

- Herstellerspezifisch
- Antwort muss geparsed werden

[11] Napalm Matrix, Zugriff: 11.11.2021, [Byea]
[12] Übersetzt mit deepl.com, Zugriff: 11.11.2021, [Byeb]

3.4 Ansible

> *Ansible ist eine radikal einfache IT-Automatisierungs-Engine, die die Cloud-Bereitstellung, das Konfigurationsmanagement, die Anwendungsbereitstellung, die Intra-Service-Orchestrierung und viele andere IT-Anforderungen automatisiert.*
>
> *Ansible wurde vom ersten Tag an für mehrstufige Bereitstellungen entwickelt und modelliert Ihre IT-Infrastruktur, indem es beschreibt, wie alle Ihre Systeme miteinander in Beziehung stehen, anstatt nur ein System nach dem anderen zu verwalten.*
>
> *Es verwendet keine Agenten und keine zusätzliche benutzerdefinierte Sicherheitsinfrastruktur, so dass es einfach zu implementieren ist - und vor allem verwendet es eine sehr einfache Sprache (YAML, in Form von Ansible Playbooks), die es Ihnen ermöglicht, Ihre Automatisierungsjobs auf eine Weise zu beschreiben, die einfachem Englisch nahe kommt.* [13]

3.4.1 Verwendung von Ansible in NUTS

Ansible ist eine praktikable Lösung zur Konfiguration eines Netzwerkes. Dazu hat Ansible ein breites Netzwerkportfolio.

Es wurde festgestellt, dass dieses Portfolio sich stark auf die Konfiguration von Netzwerkgeräten ausrichtet. Ansible scheint kompetent in diesem Bereich zu sein, dieser deckt sich jedoch nicht mit dem Scope von NUTS. Es ist möglich Ansible in einem "Check-Mode" laufen zu lassen. Trotz dieser Funktion scheint Ansible nicht das richtige Tool für NUTS zu sein.

Die Funktionen die Ansible bringt, sind limitiert brauchbar für den Use-Case von NUTS. Die Implementation würde sich nicht lohnen für den Nutzen der Ansible bringt.

3.4.2 Vor- und Nachteile von Ansible in NUTS

Vorteile

- Open-Source
- grosse Community
- basiert auf SSH und ist Agentless

Nachteile

- Scope nicht auf Testing sondern auf Konfiguration
- grosses Framework
- skaliert schlecht, da alles sequentiell ausgeführt wird

[13] Übersetzt mit deepl.com, Zugriff: 11.11.2021, [Hat]

3.5 Saltstack

> *SaltStack ist ein revolutionärer Ansatz für das Infrastrukturmanagement, der Komplexität durch Geschwindigkeit ersetzt. SaltStack ist so einfach, dass es in wenigen Minuten einsatzbereit ist, so skalierbar, dass Zehntausende von Servern verwaltet werden können, und so schnell, dass die Kommunikation mit jedem System in Sekundenschnelle erfolgt.* [14]

3.5.1 Verwendung von Saltstack in NUTS

Saltstack ist ein grosses Framework und funktioniert mit einem SaltAgent, der auf allen Maschinen installiert werden muss.
Saltstack verwendet für die Netzwerkchecks Napalm. NUTS verwendet bereits Napalm und somit ist Saltstack nicht interessant für den Use-Case von NUTS. Saltstack hat ausserdem noch einzelne Checks, die direkt implementiert wurden.
Die Analyse hat ergeben, dass es sich nicht lohnt ein so gigantisches Tool wie Saltstack in NUTS einzubauen, wenn es nur wenige Checks unterstützt und die Komplexität drastisch steigert.

3.5.2 Vor- und Nachteile von Saltstack in NUTS

Vorteile

- Open-Source
- grosse Community
- performant

Nachteile

- verwendet SaltAgent
- grosses Framework
- Scope deckt sich nicht mit NUTS

[14] Übersetzt mit deepl.com, Zugriff: 11.11.2021, [Inc]

3.6 Pytest

> *Das pytest-Framework macht es einfach, kleine Tests zu schreiben, und ist dennoch skalierbar, um komplexe funktionale Tests für Anwendungen und Bibliotheken zu unterstützen.* [15]

3.6.1 Verwendung von pytest in NUTS

Pytest ist ein Python Testing Framework. das von grundlegenden Tests wie Daten vergleichen bis zu komplexen Tests mit Vorbereitung und Mocks alles abdeckt. Es ist defacto Standard im Testing mit Python. Pytest bietet die Funktionalität Plugins für das Framework zu schreiben, um den vollen Umfang von pytest zu nutzen. Generell ist dieser Plugin-Mechanismus komplex und schwer zu debuggen.

3.6.2 Vor- und Nachteile von pytest in NUTS

Vorteile

- Open-Source
- grosse Community
- Plugins können geschrieben werden

Nachteile

- hohe Kopplung im Code zu NUTS
- grosses Framework
- wenig Transparenz im Code

[15] Übersetzt mit deepl.com, Zugriff: 15.12.2021, [Kre]

4 Netzwerktests

Im aktuellen Stand von NUTS existieren bereits Netzwerktests, die ausgeführt werden können. Im Repository befinden sich zusätzlich noch Issues für neue Tests die umgesetzt werden könnten.

4.1 Vorhandene Test-Bundles

Die aktuelle Version unterstützt folgende Netzwerktests. Eine genauere Beschreibung kann in der Dokumentation[16] von NUTS gefunden werden.

Technologie	Nr	Testbeschreibung
BGP	VT01 VT02	Existieren vordefinierte Nachbaren auf einem Router. Stimmt die Anzahl Nachbaren auf einem Router.
OSPF	VT03 VT04	Existieren vordefinierte Nachbaren auf einem Router. Stimmt die Anzahl Nachbaren auf einem Router.
CDP	VT05	Existieren vordefinierte CDP Nachbaren auf einem Router.
LLDP	VT06	Existieren vordefinierte LLDP Nachbaren auf einem Router.
iperf	VT07	Hat die Verbindung zwischen zwei Hosts eine definierte minimale Bandbreite.
Ping	VT08	Können sich zwei Hosts pingen.
Interfaces	VT09	Existiert ein Interface auf einem Host und hat die richtigen Attribute.
MPLS	VT10	Ist ein VRF auf den richtigen Interfaces konfiguriert und hat die korrekten Route Distinguisher.
OS	VT11 VT12	Existiert ein Benutzer auf einem Host. Existieren nur vordefinierte Benutzer auf einem Host.

Tabelle 3: Vorhandene Test-Bundles

4.2 Geplante Test-Bundles

Folgende Netzwerktests sind schon vor unserer Analyse für NUTS geplant. Die Liste kann auch auf dem Repository eingesehen werden.

Technologie	Nr	Testbeschreibung
Traceroute	GT01	Wird die korrekte Route zwischen zwei Hosts gewählt.
VLAN	GT02	Stimmen die VLANs und Tags auf einem Switch.
BGP	GT03	Werden die korrekte Anzahl BGP prefixes angekündigt.
DNS	GT04	Sind Domänen auflösbar.
DHCP	GT05	Ist der DHCP Server erreichbar.
Ports	GT06	Sind die richtigen Ports eines Hosts offen.
Web	GT07	Erhält der Host einen richtigen HTTPS Status Code von einem Webserver.
Interfaces	GT08	Sind die IP Adressen eines Interfaces auf einem Host korrekt.

Tabelle 4: Geplante Test-Bundles

[16] Link zur Dokumentation von NUTS: https://nuts.readthedocs.io/en/latest/testbundles/alltestbundles.html

4.3 Testportfolio

Technologie	Nr	Testbeschreibung	Stufe	Framework	Priorität
Routing	T01	Sind nur gewollte Routing Protokolle auf einem Router aktiv.	U	Napalm	2
	T02	Ist eine Default Route definiert.	U	Netmiko	1
BGP	T03	Wird spezifische Route angekündigt.	U	Netmiko	2
VRF	T04	Sind die richtigen VRF auf einem Router.	U	Napalm	2
	T05	Stimmt die Anzahl Routen eines VRFs auf einem Router.	U	Netmiko	3
	T06	Ist ein Host per Ping in einem definierten VRF erreichbar.	I	Napalm	2
OSPF	T07	Wird die richtige Anzahl Routen angekündigt.	U	Netmiko	2
	T08	Sind die richtigen Area Typen definiert.	U	Netmiko	2
IS-IS	T09	Hat der Router den korrekten Level.	U	Netmiko	2
	T10	Existieren vordefinierte Nachbaren auf einem Router.	I	Netmiko	2
	T11	Stimmt die Anzahl Nachbaren auf einem Router.	I	Netmiko	2
	T12	Wird die richtige Anzahl Routen angekündigt.	U	Netmiko	3
Spanning-Tree	T13	Ist Spanning-Tree aktiv auf einem Switch.	U	Netmiko	2
	T14	Wird der richtige Spanning-Tree Modus verwendet.	U	Netmiko	3
	T15	Ist Spanning-Tree auf einem Interface im richtigen State.	U	Netmiko	1
	T16	Hat ein Interface die richtige Rolle. (Desg, Root, Altn)	U	Netmiko	2
	T17	Ist der richtige Switch Root Bridge.	I	Netmiko	2
First Hop Redundanz	T18	Ist der Router im richtigen State.	U	Netmiko	2
	T19	Stimmt die Gateway IP Adresse.	U	Netmiko	2
	T20	Wird das richtige First-Hop-Redundancy Protokoll verwendet.	U	Netmiko	2
VSS	T21	Ist der Switch in der richtigen VSS Domäne.	U	Netmiko	2
	T22	Ist die richtige Bandbreite konfiguriert.	U	Netmiko	3
	T23	Sind die Nachbaren auf den richtigen Interfaces.	I	Netmiko	2
EtherChannel	T24	Ist die EtherChannel Konfiguration auf einem Interface korrekt.	U	Netmiko	2
	T25	Sind die richtigen Interfaces im Port-Channel.	U	Netmiko	1
	T26	Ist der Port-Channel im richtigen Layer (2 oder 3).	U	Netmiko	1
ARP	T27	Sind ARP Einträge vorhanden.	I	Napalm	1
	T28	Sind die Anzahl ARP Einträge im richtigen Bereich.	I	Napalm	2
SNMP	T29	Überprüfen ob Traps generiert wurden.	I	Netmiko	2
	T30	Wird die richtige EngineID verwendet.	U	Netmiko	3
	T31	Haben die SNMP Community String die richtige Einstellung (Lesen/Schreiben).	U	Netmiko	1
CDP	T32	Stimmt die Anzahl CDP Nachbaren	I	Netmiko	2

LLDP	T33	Stimmt die Anzahl LLDP Nachbaren	I	Napalm	2
IP Telefonie	T34	Verbindet sich ein Telefon über das Voice VLAN.	I	Netmiko	3
VLAN	T35	Hat ein VLAN die richtigen Interfaces zugewiesen.	U	Napalm	3
	T36	Stimmt die Anzahl VLANs.	U	Napalm	2
	T37	Sind nur definierte VLANs auf einem Switch.	U	Napalm	1
	T38	Ist ein Port im richtigen Switchport-Modus.	U	Netmiko	3
	T39	Sind die richtigen VLAN einem Trunk zugewiesen.	U	Netmiko	1
	T40	Ist das richtige VLAN als Native definiert.	U	Netmiko	2
	T41	Hat ein Router Subinterface das korrekte VLAN zugewiesen.	U	Netmiko	3
Umgebung	T42	Ist die Temperatur im richtigen Bereich.	U	Napalm	3
	T43	Sind die Komponenten Healthy.	U	Napalm	3
Port-Security	T44	Ist die statisch gesetzte MAC-Adresse korrekt.	U	Netmiko	2
	T45	Ist die gesetzte Sticky-MAC-Adresse korrekt.	U	Netmiko	2
	T46	Stimmt die Anzahl dynamisch-lernbarer MAC-Adressen.	U	Netmiko	2
MAB	T47	Ist MAB konfiguriert auf einem Switch.	U	Netmiko	2
	T48	Stimmt die MAB Konfiguration auf dem Interface.	U	Netmiko	2
	T49	Stimmt die Anzahl MAB authentisierter Geräte.	I	Netmiko	2
dot1x	T50	Ist dot1x konfiguriert auf einem Switch.	U	Netmiko	2
	T51	Stimmt die dot1x Konfiguration auf dem Interface.	U	Netmiko	2
	T52	Stimmt die Anzahl dot1x authentisierter Geräte.	I	Netmiko	2
Redundanz	T53	Besteht eine definierte Verbindung noch, wenn ein Uplink heruntergefahren wird.	I, S	Netmiko	2
Konfiguration	T54	Stimmt die Konfiguration mit der standard Konfiguration überein.	U	Napalm	1
	T55	Stimmt die Running Konfiguration mit der Startup Konfiguration überein.	U	Napalm	2
VXLAN	T56	Existieren definierte VN-Segmente auf einem Host.	U	Netmiko	2
	T57	Stimmt das Mapping von VLANs zu VN-Segmenten.	U	Netmiko	2
	T58	Stimmen die gefundenen NVE Peers.	I	Netmiko	2
	T59	Stimmt die Multicast Gruppe auf dem VTEP.	U	Netmiko	3
	T60	Wurde das VNI über die richtige Plane gelernt.	I	Netmiko	3
VPN	T61	Lässt sich eine TLS Verbindung aufbauen.	S	Netmiko	2
	T62	Lässt sich eine IPSec Verbindung aufbauen.	S	Netmiko	2
WLAN	T63	Sind definierte SSIDs verfügbar.	I	Netmiko	3
Multicast	T64	Stimmen die PIM Nachbaren.	I	Netmiko	3
	T65	Stimmt die Multicast Route.	I	Netmiko	3

Tabelle 5: Testportfolio

4.3.1 Legende

Stufe:
U: Unit-Test
I: Integrations-Test
S: System-Test

Priorität:
1: Hohe Priorität
2: Mittlere Priorität
3: Tiefe Priorität

4.3.2 Priorisierung

Die Priorität der Test-Bundles wurde mithilfe der Interviews festgelegt.

Bei den Interviews wurde für jedes Test-Bundle eine Priorität von 1 (Hoch) bis 3 (Tief) ermittelt. Diese Prioritäten wurden über alle Interviews korreliert und wieder in ein Priorität 1 (Hoch) bis 3 (Tief) eingeteilt.

4.3.3 Ausgewählte Test-Bundles

Es wurden die folgenden Test-Bundles ausgewählt:

- GT02: Stimmen die VLANs und Tags auf einem Switch
- T04: Virtual Routing & Forwarding
- T27: ARP Einträge
- T28: ARP Einträge im vorgegebenen Bereich
- T32: CDP Nachbar Anzahl
- T33: LLDP Nachbar Anzahl
- T35: Hat ein VLAN die richtigen Interfaces zugewiesen.
- T37: Sind nur definierte VLANs auf einem Switch.
- T55: Stimmt die Running Konfiguration mit der Startup Konfiguration überein.

Diese Test-Bundles wurden ausgewählt weil sie mit Napalm und ohne Änderungen an der Architektur von NUTS umgesetzt werden können.

5 Architektur

Die durchgeführte Analyse der Frameworks und die Interviews mit Industriepartnern haben verschiedene Problemstellungen aufgezeigt, die aktuell in der Architektur von NUTS existieren. Dafür wurden mögliche Lösungsansätze evaluiert, die zukünftige Architekturen im Bereich des Software-Codes und des generellen Modells für NUTS beinhalten.

5.1 Architektur-Modell

Zurzeit ist NUTS ein "Command-Line Tool" das lokal auf einem Computer ausgeführt wird. Die Voraussetzung ist, dass auf dem Computer Python3 installiert ist. Je nach dem wo sich dieser Computer im Netzwerk befindet, können auch verschiedene Tests ausgeführt werden. Ein Computer der sich nicht im zu testenden Netzwerk befindet, kann nur Netzwerkgeräte testen, die er per SSH erreicht werden. Dabei können auch nur Tests ausgeführt werden, die vom Betriebssystem des Netzwerkgerätes unterstützt werden. Falls sich der Computer im zu testenden Netzwerk befindet, können auch Tests ausgeführt werden, die die Sicht eines "Clients" darstellen. Dies beinhaltet zum Beispiel Tests, ob Webseiten im lokalen Netzwerk erreichbar sind, ob definierte WLAN-SSIDs verfügbar sind oder ob spezielle Peripheriegeräte funktionieren.

Ausserdem ist NUTS wie erwähnt eine Client-Applikation, die zuerst installiert werden muss. Eine Serverversion und ein GUI existieren nicht, würden aber den Nutzen für Netzwerkingenieure deutlich verbessern.

5.1.1 Client Software

NUTS kann als Client Software weiter entwickelt werden. Das bedeutet, dass die Software auf einem Client installiert werden muss. Dadurch muss nichts an der Architektur von NUTS erweitert werden. Es ist aufwändig für Netzwerkingenieure die Software auf neuen Clients zur Verfügung zu stellen, das kann eine Einstiegshürde sein. Die Clients auf denen NUTS läuft müssen vom aktuellen Netzwerk Zugang zu den zu testenden Netzwerkgeräten haben, diese Verbindung könnte möglicherweise durch eine Firewall blockiert werden.

5.1.2 Server Software

NUTS kann als Server Software konzipiert werden. Dafür gäbe es zwei Möglichkeiten. Ein zentraler Server oder ein Master/Slave Modell. Die Architektur für NUTS muss umgebaut und noch mit einem GUI, bestenfalls einem Webinterface, erweitert werden.

Zentraler Server

NUTS als zentraler Server würde einen Server benötigen. Dieser Server müsste zu allen Netzwerkgeräten, die getestet werden möchten, SSH Zugriff haben.

Der Vorteil dieser Variante ist, dass es nur einen Server braucht und der Umbau der Architektur sich in Grenzen hält. Der Server kann Netzwerke jedoch nicht aus Client-Sicht testen.

Dieser Ansatz skaliert bei grossen Netzwerk schlecht, da weder Redundanz, noch gute Perfomance garantiert sind. Die Perfomance kann durch Latenz und Congestion im Netzwerk erheblich beeinträchtigt werden.

Master / Slave

Als zweite Servervariante könnte ein Master/Slave Modell in Betracht gezogen werden. Es gibt einen Master Server auf welchen über ein GUI zugegriffen wird. Für Tests kontaktiert der Masterserver die Slaves, die sich in den Netzwerken befinden und die Tests ausführen. Die Resultate liefern sie wieder an den Masterserver.

Dadurch können die Netzwerke aus Client-Sicht getestet werden. Der Masterserver braucht lediglich einen Kommunikationskanal auf die einzelnen Slaves und nicht auf alle Netzwerkgeräte. Diese Kommunikation kann

SSH, HTTPS, gRPC oder Ähnliches sein. Für diese Variante braucht es einen grösseren Umbau der Architektur. Zusätzlich werden viele Slaves[17] benötigt, die in den Netzwerken verteilt sind.

Dieser Ansatz skaliert gut, auch bei grossen Netzwerken. Zu beachten ist aber das initale Investment in die Infrastruktur und die Kosten beim Betrieb der Testinfrastruktur. Diese Variante hat die Möglichkeit eine grosse Transparenz in das Netzwerk zu bringen.

5.2 Software-Architektur

5.2.1 Aktueller Ablauf

Um das aktuelle Problem mit der Weiterentwicklung von NUTS zu verstehen, muss zuerst verstanden werden wie das Ausführen eines Test-Bundles mit NUTS abläuft.

Die Netzwerkingenieure definieren die gewünschten Tests in YAML Dateien, in denen das zu testende Netzwerkgerät definiert wird. Die Informationen über das Netzwerkgerät, wie zum Beispiel das Betriebssystem, die IP und die Login Daten werden von den Netzwerkingenieuren in YAML Dateien definiert, die dann von Nornir in das Inventar eingelesen werden. Ein Beispiel dieser zwei YAML-Datei Strukturen kann in der Dokumentation von NUTS [18] gefunden werden.

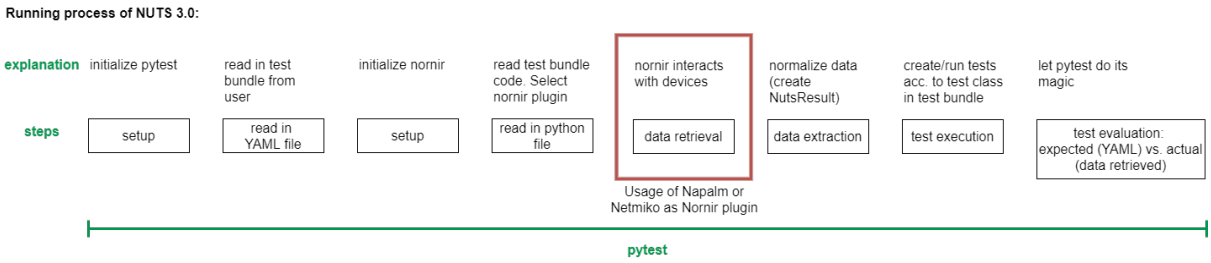


Abbildung 7: Ablauf NUTS 3.0, Grafik des NUTS-Entwicklerteams, angepasst

NUTS wird nicht direkt, sondern mit pytest gestartet und als Parameter wird die YAML Datei mit den definierten Tests mitgegeben. NUTS liest als erstes die YAML Datei des Test-Bundle ein und initialisiert Nornir, mit welchem auch das Inventar mit den Netzwerkgeräten eingelesen wird. Als nächstes wird der Code des Test-Bundles ausgeführt. Im Code der jeweiligen Test-Bundles ist definiert, welches Nornir-Plugin verwendet wird um die Daten von den Netzwerkgeräten zu holen. Nornir fragt die benötigten Daten mit Hilfe des Plugin von den betroffenen Netzwerkgeräten ab und gibt diese an NUTS zurück. NUTS extrahiert die Daten und bringt diese in die gewünschte Struktur. Zum Schluss vergleicht pytest die Daten anhand der definierten Test-Cases im Test-Bundle Code mit den Soll-Daten, die im YAML definiert sind.

5.2.2 Aktuelles Problem

Wie im Kapitel Frameworks beschrieben, verwendet NUTS diese Frameworks um die Tests auszuführen. Die Analyse der Frameworks hat ergeben, dass das aktuell genutzte Framework Napalm, welches als Nornir-Plugin verwendet wird, nicht alle definierten Test-Bundles im Testportfolio abdeckt. Dies liegt daran, dass noch nicht so viele "getter" implementiert sind, die die benötigten Daten von den Netzwerkgeräten herstellerunabhängig holen können.

Als Alternative gibt es aktuell auch noch Netmiko, das als Nornir-Plugin in NUTS eingebunden ist. Die Netmiko Test-Bundle sind jedoch nicht herstellerunabhängig, da im Test-Bundle Code das benötigte Kommando für Cisco IOS Geräte definiert ist um die Daten zu holen und das Resultat nur für die Antwortstruktur von Cisco geparsed wird. Somit können die Netmiko Test-Bundle nur mit Cisco IOS Geräten verwendet werden.

[17] Als Slaves könnten zum Beispiel Raspberry Pi verwendet werden.
[18] Zugriff, 14.12.2021: https://nuts.readthedocs.io/en/latest/tutorial/firststeps.html

Dies ist ein grosses Problem für die Weiterentwicklung von NUTS. Alle Test-Bundles, die Informationen von Netzwerkgeräten benötigen, die nicht mit Napalm geholt werden können, müssen anderweitig implementiert werden. Dabei muss jedoch berücksichtigt werden, dass NUTS herstellerunabhängig Tests ausführen können muss. Die Netzwerkingenieure definieren die Tests in YAML Dateien für alle Betriebssysteme der Netzwerkgeräte gleich. Die aktuelle Software-Struktur von NUTS weist eine hohe Kopplung auf. Dadurch ist es schwierig, die Architektur zu erweitern, da bei bereits kleinen Änderungen viele Teile der Software angepasst werden müssen. Ausserdem ist NUTS als pytest-Plugin auf pytest angewiesen. Viele der aktuellen Abläufe benutzten pytest und sind darauf angewiesen. Änderungen an pytest können auch NUTS betreffen. Mit dieser Gegebenheit ist es folglich auch schwierig, neue Test-Bundles zu implementieren, die nicht dem aktuellen Schema entsprechen. Das heisst, Test-Bundles die nicht Napalm verwenden und herstellerunabhängig Test-Bundles mit Netmiko.

In diesem Kapitel werden verschiedene Lösungsansätze aufgezeigt, wie dieses Problem gelöst werden kann und was für Vor- und Nachteile sich dadurch ergeben.

5.2.3 Lösungsansätze

Napalm erweitern

Wenn die Architektur nicht anpassen werden soll, ist die beste Lösung neue "getter" für Napalm zu implementieren. "Getter" in Napalm sind Funktionen, die herstellerunabhängig Daten von Netzwerkgeräten abfragen können. Da Napalm Open-Source ist, können neue "getter" von jedem implementiert werden, was auch der Open-Source-Community etwas bringt. Napalm bietet auch bereits die Struktur für den ganzen herstellerunabhängigen Bereich, dadurch können relativ unkompliziert neue "getter" pro Betriebssystem hinzugefügt werden. Dieser Teil muss in Napalm erweitert werden, damit die Test-Bundles in NUTS über die neuen "getter" die benötigten Informationen abfragen können.

Der Nachteil dieser Lösung ist, dass es sich um ein Open-Source Projekt handelt. Das INS hat nicht die volle Kontrolle über Napalm und neuer Code muss zuerst von den Core-Maintainern von Napalm akzeptiert werden. Als neue Partei neuen Code für ein Open-Source Projekt in den nächsten Release zu bekommen, kann knifflig sein. Dies wurde auch im Buch "Working in Public: The Making and Maintenance of Open Source Software"[19] so beschrieben. Die Autorin hat EntwicklerInnen nach ihren Erfahrungen mit Open-Source Projekten gefragt und konnte feststellen, dass Entwickler nicht immer schnell aufgenommen werden.

Eigenes Nornir-Plugin

Eine weitere Möglichkeit ist ein eigenes, neues Nornir-Plugin zu erstellen. Dadurch müsste die Architektur von NUTS ebenfalls nicht angepasst werden.

Ein neues Nornir-Plugin kann ähnlich wie das Napalm Plugin für Nornir in NUTS eingebunden werden und herstellerunabhängig die Daten über das neue Plugin abfragen. Strukturell kann sich auch an Napalm orientieren werden. Die Vorteile dadurch sind, dass die volle Kontrolle über das Plugin beim Entiwckler ist und nach eigenen Bedürfnissen an NUTS anpassen kann.

Ein neues Plugin ist wahrscheinlich ähnlich wie das Napalm Plugin und ein grosser initialer Aufwand.

Preprocessing

Die aktuelle Struktur von NUTS ist wie bereits erwähnt kompliziert. Für neue Entwickler ist es schwierig, sich in NUTS einzulesen und Änderungen haben grosse Auswirkungen. Für NUTS ist es in Zukunft sinnvoll, wenn die Architektur weniger von pytest abhängt und wo sinnvoll selbstständig arbeiten verrichtet.

[19] Zugriff 14.12.2021, [Egh]

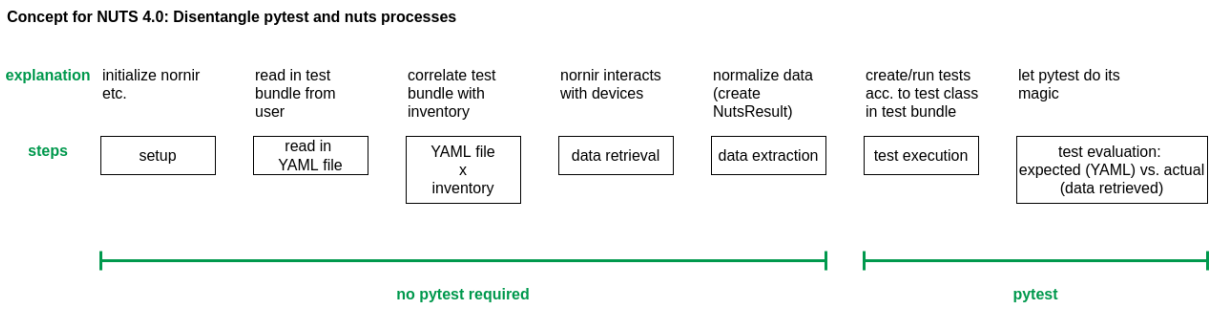


Abbildung 8: Konzept für NUTS 4.0, ZVG NUTS-Entwicklungsteam, 2021

Für ein Preprocessing muss die Architektur angepasst werden. Ein möglicher Ablauf sieht folgt aus:

NUTS initialisiert als erstes Nornir und ladet die definierten Tests und das Nornir Inventar mit den Netzwerkgeräten. Danach werden die Informationen der zu testenden Netzwerkgeräte aus dem Inventar geladen und mit den definierten Tests korreliert. Dadurch ergibt sich eine Datenstruktur, in der alle benötigten Daten vorhanden sind. Mit dieser Datenstruktur kann dann zum Beispiel mit dem Netmiko Plugin für Nornir herstellerspezifisch die gewünschten Daten von den Netzwerkgeräten abfragen. Die Information, welches herstellerspezifische Kommando an Netmiko mitgegeben werden muss, kann über diese korrelierte Datenstruktur bestimmt werden. Das Parsen der Ergebnisse kann mit Tools wie textfsm[20] durchgeführt werden. Pytest kommt am Schluss nur zum Einsatz, um die erwarteten und tatsächlichen Daten zu vergleichen.

Diese Variante gibt den Entwicklern die grösste Freiheit. Erforderliche Daten könnten geladen und zusammengeführt werden und unkompliziert erweitert werden. Ausserdem ist NUTS dadurch nicht mehr so stark von anderen Frameworks wie pytest und Napalm beeinflusst. Zusätzlich kann die neue Architektur nach Software-Engineering-Best-Practices geplant werden.

Ein offensichtlicher Nachteil dieser Variante ist der grosse Aufwand. Viele Teile der Applikation müssten neu geschrieben werden. Das Umbauen der Architektur wird zwangsweise auch “breaking changes” beinhalten und könnte den Upgrade einer alten Version auf eine neue erheblich erschweren.

5.3 Inventar

Die Netzwerkgeräte werden im “simple inventory” von Nornir eingetragen. Danach kann in der Testdefinitionssprache auf die Geräte zugegriffen werden. Netzwerke von nationalen und internationalen Firmen sind riesig. Diese von Hand in das Inventar zu übertragen, ist ein grosser Aufwand und kann potentielle Benutzer von NUTS abschrecken.

Hier ist eine Automatisierung wünschenswert, um die Netzwerkgeäte aus einem bestehenden Assetmanagement in das Inventar von Nornir zu übernehmen. Die interviewten Industriepartner besitzen alle ein Inventar für ihre Netzwerkgeräte. Teilweise in einer oder mehreren Applikationen oder in Listen.

Daher wird es schwierig, eine einheitliche Lösung für alle zu definieren. Alle Applikation der Industriepartner haben die Funktion, die Netzwerkgeräte zu exportieren. Mit Individuallösungen könnte das automatisiert werden.

Eine weitere Möglichkeit ist eine Import Funktion in NUTS, die ein CSV mit einer bestimmten Struktur einliest. Die Exporte aus den Inventar-Applikationen der Firmen in eine vorgegebene Struktur zu bringen ist simpler, als diese in YAML Struktur zu bringen.

[20] Link zum Repository: https://github.com/google/textfsm

6 Interviews

6.1 Organsation und Informatik Stadt Zürich - OIZ

6.1.1 Vorstellung OIZ

Die Stadt Zürich ist mit einer Wohnbevölkerung von über 410'000 Personen die grösste Stadt der Schweiz. Das Besondere an Zürich: die rund 28'000 städtischen Mitarbeitenden und 1'000 Lernende betreuen nicht nur die klassischen Verwaltungsaufgaben, sondern betreiben auch zwei Spitäler, die Wasser- und Stromversorgung und die städtischen Tram- und Buslinien.

Die Informatik der Stadt Zürich ist dabei das Nervenzentrum. Mit dem Betrieb von 2'500 Applikationen in zwei der modernsten Rechenzentren der Schweiz stellt sie sicher, dass die Stadt Zürich ihre Dienstleistungen reibungslos erbringen kann. Dabei gestalten die 600 ITMitarbeitenden die digitale Transformation unserer Gesellschaft mit und ermöglichen so, dass die Bevölkerung städtische Dienste online beziehen kann und die Zürcher Kinder in den Schulen für die digitale Zukunft gerüstet sind:
Wir machen Zürich - IT der Stadt Zürich. [21]

6.1.2 Interview OIZ

Beim Interview mit der Organisation und Informatik der Stadt Zürich erklärten die Verantwortlichen der Netzwerkabteilung das allgemeine Vorgehen im Bezug auf Netzwerktesting. Aus diesem Vorgehen und den Testprozessen wurde das Testportfolio ergänzt. Sobald diese Test-Bundles in der Liste umgesetzt wären, könnte das Tool die manuelle Arbeit eines Funktionschecks, nach zum Beispiel einer OS Aktualisierung, im OIZ ersetzen.

Aus dem Interview sticht auch klar heraus, dass das OIZ mehr in das Testing von neuen Cisco OS Versionen investiert, als in das Testing von Netzwerken nach einem Rollout. Es werden neue OS Versionen anhand einer Checkliste getestet, um zu sehen, ob sich irgendwelche Bugs eingeschlichen haben. Diese Tests zu automatisieren ist aber schwierig, da Layer 1 Funktionscheck nicht ersetzt werden können, bei denen Kabel ein- und ausgesteckt werden müssen. Als alternative würden sich virtuelle Images von Cisco bieten, bei diesen ist aber die Maturität nicht gegeben. Die virtuellen Images sind grösstenteils nicht identisch zu den Releases der physischen Komponenten oder nicht vorhanden.

6.2 MAN Energy Solutions Schweiz AG

6.2.1 Vorstellung MAN

MAN Energy Solutions ebnet den Weg in eine klimaneutrale Weltwirtschaft. Ob Industrieproduktion oder Energiewirtschaft: Wir denken ganzheitlich und packen schon heute die Herausforderungen von morgen an – für eine nachhaltige Wertschöpfung unserer Kunden. Am Standort Zürich sind wir rund 800 Mitarbeiter und 60 Lernende. Wir sind das einzige Unternehmen der Maschinenindustrie, das im Herzen der Stadt Zürich seine Produktion hat – und dies seit über 200 Jahren. MAN Energy Solutions hat seinen Hauptsitz in Deutschland und beschäftigt rund 14.000 Mitarbeiter an mehr als 120 Standorten weltweit. Ein Unternehmen, das an Lösungen arbeitet, die die Gesellschaft wirklich braucht. [22]

6.2.2 Interview MAN

Das Interview mit den Netzwerkverantwortlichen der MAN Energy Solutions Schweiz AG gab Einsicht in den Arbeitsvorgang der Firma. Die MAN setzt hauptsächlich Netzwerkgeräte von Cisco ein. Im WAN wird eine MPLS Leitung, die von Verizon betrieben wird, eingesetzt, sowie eine SD-WAN Lösung von Fortinet. Das Netzwerkdesign ist klassisch hierarchisch, ohne SDN und mit wenigen dynamischen Routing Protokollen. Bei MAN wird kontinuierlich die Infrastruktur überwacht. Die Überwachung erfolgt durch eine vielzahl von

[21] Stadt Zürich, Zugriff: 05.11.2021, [Zür]
[22] MAN Energy Solutions, Zugriff: 05.11.2021, [MAN]

Monitoringsystemen. Es wird PRTG, Solarwinds und Cacti verwendet. Bei Umstellungen im Netzwerk wird nach Ermessen des Netzwerkingenieurs getestet. Bei kritischen Systemen gibt es Key-User Tests, die Ihre Applikationen testen.
Weiter zeigte die MAN das interne Tool zur Verwaltung aller Hardware. Ihre Lösung ist selbst programmiert, erfasst alle Daten von Hard- und Software aller Netzwerkgeräte. In dieser Datenbank werden alle Informationen manuell eingetragen. Die Angaben beinhalten zum Beispiel Hostname, Softwareversion, Ansprechpartner, Service-Level und weiteres. Diese Datenbank ist die Single-Source-of-Truth für die MAN. Das Monitoring in Solarwinds wird über diese Datenbank eingerichtet. Sobald die Geräte in diesem Inventar eingetragen sind, werden diese mit Solarwinds synchronisiert und überwacht.

6.3 Migros Genossenschafts Bund

6.3.1 Vorstellung MGB

> *Die Migros (Kurzform von Migros-Genossenschaft) gehört zu den grössten Detailhandelsunternehmen der Schweiz. Sie ist als Verbund von Genossenschaften mit fast 2,3 Millionen Genossenschaftern organisiert. Der Kern besteht aus zehn regionalen Genossenschaften, die zusammen den Migros-Genossenschafts-Bund (MGB) bilden. Dem MGB wiederum sind zahlreiche Tochtergesellschaften und Beteiligungen untergeordnet. Dazu gehören unter anderem der Detailhändler Denner, die Convenience-Kette Migrolino, zahlreiche Produktionsbetriebe zur Herstellung von Eigenmarken, der Reiseveranstalter Hotelplan, der Onlinehändler Digitec Galaxus und die Migros Bank. Von 1997 bis 2020 gehörte auch die Warenhausgruppe Globus zum MGB.* [23]

6.3.2 Interview MGB

In den Interviews mit verschiedenen Abteilungen zeigten die verantwortlichen Personen die Netzstruktur, sowie auch die administrativen Prozesse auf.
Das Netzwerk besteht aus verschiedenen Elementen, die durch verschiedene Parteien betrieben werden. Dadurch werden die Büros und Datacenter durch den Migros Genossenschaft Bund verwaltet, die WAN Strecke durch Swisscom und die Filialen durch die einzelnen Genossenschaft.
Dementsprechend besteht noch kein klarer Standard. Es wird eine vielzahl von Routingprotokollen und Technologien eingesetzt. Da neue Technologien, wie Software-Defined-Networking, aber auch alte Technologien vorhanden sind, gestaltet sich das Testen der Infrastruktur schwer. Es wird in der Regel nach Ermessen des Ingenieurs getestet. Alle Arbeiten die durch externe Dienstleister verrichtet werden, müssen durch die interne Infrastruktur Abteilungen nicht getestet werden.
Somit hat Migros die Priorität nicht beim Testing, sondern beim Erschaffen einer einheitlichen Infrastruktur. Viele Lösungen die doppelt angeboten werden oder veraltet sind, sollen beseitigt werden. Dann wird auch ein automatisiertes Testing möglich sein. Zurzeit scheint der Technologie-Stack viel zu umfänglich zu sein, um sauber testen zu können.
Momentan wird mit diversen Checklisten gearbeitet, die hauptsächlich administrative Prozesse abbilden und auch nicht automatisiert werden können.
Eines der grössten Problem beim erstellen einer Testautomatisierung wäre auch das Inventar. Dieses müsste vollständig sein und damit auch an ein Testingframework angebunden werden. Ein solches Inventar gibt es jedoch zur Zeit nicht.
Das Interview hat auch ergeben, dass Testing in vielen Bereichen Sinn macht und auch erwünscht wird, aber nicht in allen Bereich machbar ist. Bei WLAN wurde im Interview schnell klar, dass Testing hier viel zu schlecht skaliert.
Beim Testing von WLAN müsste jeder Accesspoint getestet werden. Nicht nur jeder Accesspoint, sondern auch durch jedes mögliche Gerät. Bei einer Vielfalt wie es bei einem Konzern der Fall ist, ist das eine Sache der Unmöglichkeit.
Die Migros testet WLAN zur Zeit durch Rolloutteams, die das Netzwerk auch aufbauen. Es scheint auch in Zukunft die beste Lösung zu sein.

[23] Migros, Zugriff: 13.11.2021, [Wik]

III. Implementation

1 Test Infrastruktur

1.1 Lab Topology Builder

Der Lab Topology Builder, kurz LTB, ist ein Produkt des INS. Das LTB erlaubt es virtuell Laborumgebungen aufzusetzen. Im Labor ist es möglich, eine Vielzahl an Geräten wie Router, Switches, Server und weiteres, zu erstellen. Um NUTS weiter entwickeln zu können wird ein Netzwerk benötigt, das möglichst ähnlich zu einem realen, produktiv verwendeten Netzwerk ist.

1.2 Setup des virtuellen Labors

Das Setup der initialen Infrastruktur wird vom INS übernommen. Die Umgebung beinhaltet 4 Router, 2 Switches und 2 Ubuntu Server. Alle Geräte sind gemäss Abbildung 9 miteinander verbunden. Dabei zu beachten ist, dass der Internet Gateway eine technische Vorgabe vom LTB ist und in unserer Lab Umgebung nicht aktiv verwendet wird.

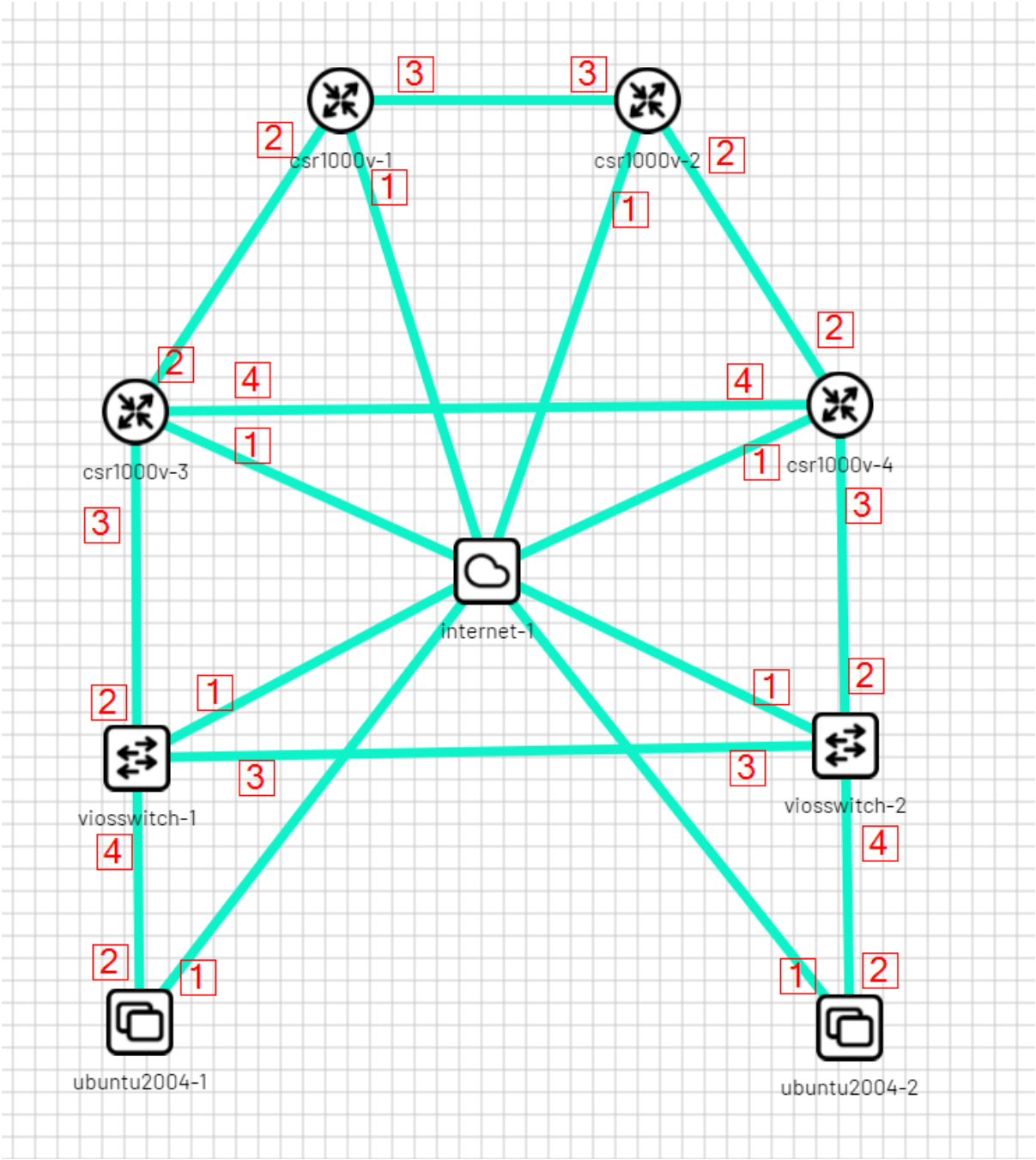


Abbildung 9: Labor Umgebung, ZVG Janik Zwicker, 12.10.2021

1.3 Logische Topologie

Das Ziel in diesem Labor ist es eine möglichst diverse Netzwerkumgebung zu erstellen, um viele verschieden Test-Bundles ausführen zu können. Somit wurde entschieden zwischen den Switches und den Clients zwei VLANs zu nutzen. Zwischen den Switches und den Edge-Routern "csr1000v-3 & csr1000v-4" sind dot1q Trunks im Einsatz. Die beiden Edge-Router betreiben die Gateway-Adresse des Netzwerkes mit First-Hop-Redundancy. Alle Router sind in der gleichen OSPF Area. Die Area 0 ist somit auch die Backbone Area. Alle Routen werden an alle Geräte propagiert. Alle Verbindungen zwischen den Router sind auf Layer 3 Basis. Weil das Lab überschaulich ist, wurden alle Switches und Router via CLI konfiguriert. Die Ubuntu Maschinen wurden auch auf dem CLI konfiguriert, für die Netzwerkkonfiguration wurde netplan verwendet.

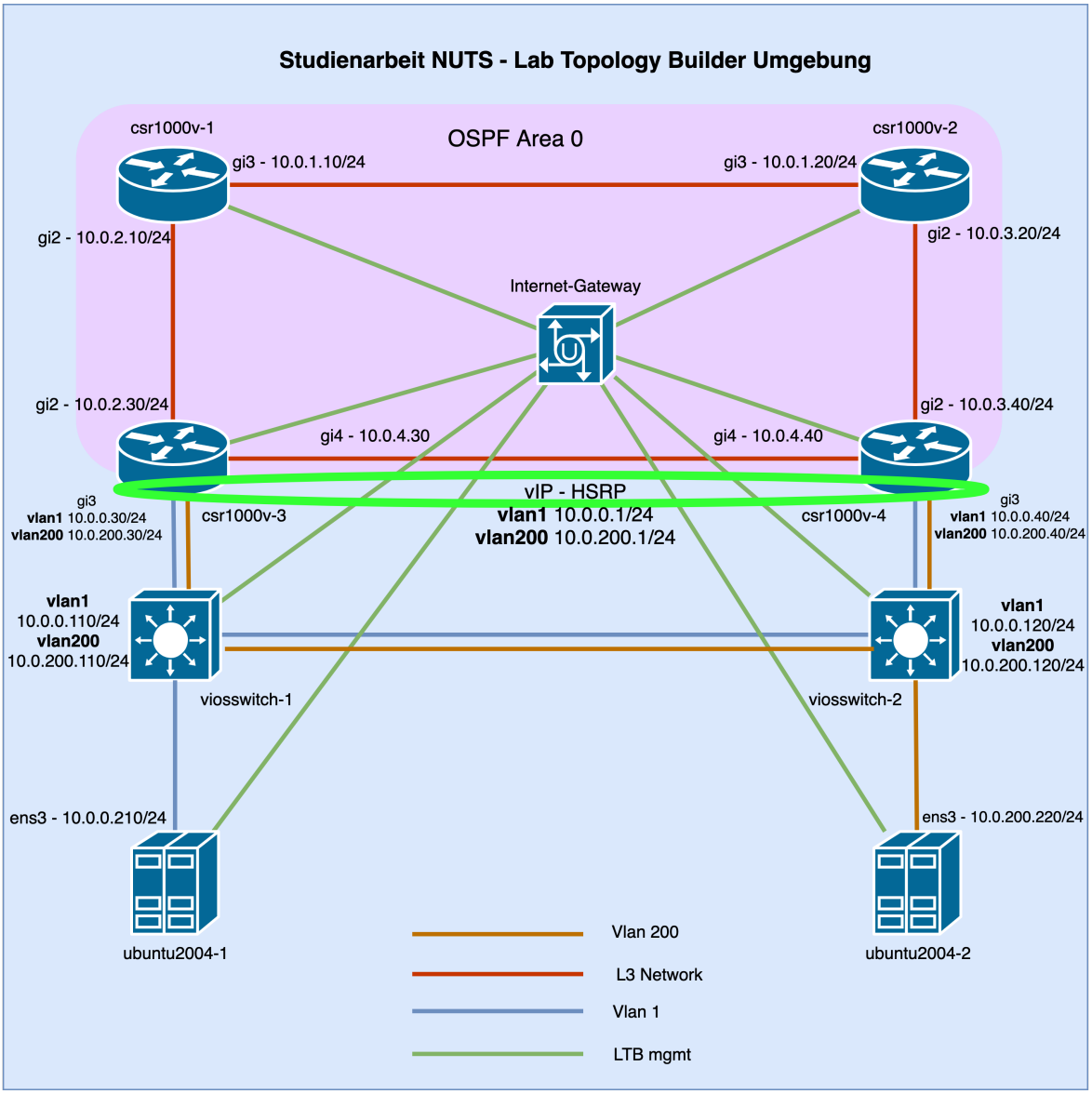


Abbildung 10: Logische Topologie

2 Test-Bundles

2.1 GT02 - Stimmen die VLANs und Tags auf Switch

Test-Bundle: Dieses Test-Bundle kann auf einem Switch erkennen, ob definierte VLANs vorhanden sind. Zusätzlich kann die Zuordnung VLAN Name zu VLAN Tag überprüft werden.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_get_vlans
    test_class: TestNapalmVlans
    test_data:
      - host: <hostname, required>
        vlan_tag: <tag>
        vlan_name: <vlan name>
      - host: <hostname, required>
        vlan_tag: <tag>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob definierte VLANs auf einem Gerät sind:"host, vlan_tag"
- Testet ob definiertes VLAN mit Namen und Tag vorhanden ist: "host, vlan_name, vlan_tag"

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_get_vlans
    test_class: TestNapalmVlans
    test_data:
      - host: S1
        vlan_tag: 1
        vlan_name: default
      - host: S1
        vlan_tag: 200
```

2.2 T04 - Virtual Routing & Forwarding VRF

Test-Bundle: Dieses Test-Bundle kann überprüfen, ob auf einem Gerät die richtigen VRF konfiguriert sind. Ausserdem kann auch überprüft werden, ob die richtigen Interfaces einem VRF zugewiesen sind.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_get_vrf
  test_class: TestNapalmVrf
  test_data:
    - host: <hostname, required>
      name: <vrf name>
      interfaces:
        - <interface name>
        - <interface name>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob ein VRF auf einem Gerät existiert: "host, name"
- Testet ob ein VRF auf einem Gerät auf den richtigen Interfaces konfiguriert ist: "host, name, interfaces"

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_get_vrf
  test_class: TestNapalmVrf
  test_data:
    - host: S1
      name: "default"
      interfaces:
        - "GigabitEthernet0/1"
        - "GigabitEthernet0/2"
        - "GigabitEthernet0/3"
        - "Vlan1"
```

2.3 T27 - ARP Einträge

Test-Bundle: Mit diesem Test-Bundle kann man überprüfen, ob gewünschte ARP Einträge auf einem Geräte verfügbar sind. Es prüft ob vorgegebene Einträge, welche das Interface und die IP enthalten, in der ARP Tabelle des Gerätes vorhanden sind.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_get_arp
  test_class: TestNapalmArp
  test_data:
    - host: <hostname, required>
      interface: <interface name>
      ip: <IP adresse>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob ein ARP Eintrag auf einem Interface mit einer gegebenen Adresse vorhanden ist: "host, interface, ip"

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_get_arp
  test_class: TestNapalmArp
  test_data:
    - host: S1
      interface: Vlan1
      ip: "10.0.0.30"
```

2.4 T28 - ARP Einträge im vorgegebenen Bereich

Test-Bundle: Dieses Test-Bundle überprüft, ob die Anzahl ARP Einträge auf einem Geräte im richtigen Bereich ist.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_get_arp
  test_class: TestNapalmArpRange
  test_data:
    - host: <hostname, required>
      min: <number>
      max: <number>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob die Anzahl ARP Einträge zwischen min und max ist: "host, min, max"

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_get_arp
  test_class: TestNapalmArpRange
  test_data:
    - host: S1
      min: 13
      max: 20
```

2.5 T32 - CDP Nachbar Anzahl

Test-Bundle: Dieses Test-Bundle ist eine Erweiterung auf das bestehende Test-Bundle "CDP Neighbor". Mit diesem Test-Bundle kann man überprüfen, ob die Anzahl an CDP Neighbors eines Gerätes stimmt.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.netmiko_cdp_Nachbarn
  test_class: TestNetmikoCdpNeighborsAmount
  test_data:
    - host: <hostname, required>
      amount: <number>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob das Gerät eine vorgegebene Anzahl CDP Nachbarn hat: "host, amount"

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.netmiko_cdp_neighbors
  test_class: TestNetmikoCdpNeighborsAmount
  test_data:
    - host: S1
      amount: 3
```

2.6 T33 - LLDP Nachbar Anzahl

Test-Bundle: Dieses Test-Bundle ist eine Erweiterung auf das bestehende Test-Bundle "LLDP Neighbor". Mit diesem Test-Bundle kann man überprüfen, ob die Anzahl von LLDP Nachbarn eines Gerätes stimmt.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_lldp_neighbors
  test_class: TestNapalmLldpNeighborsAmount
  test_data:
    - host: <hostname, required>
      amount: <number>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob das Gerät die vorgegebene Anzahl LLDP Nachbarn hat: "host, amount"

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_lldp_neighbors
  test_class: TestNapalmLldpNeighborsAmount
  test_data:
    - host: S1
      amount: 3
```

2.7 T35 - Interface in VLAN

Test-Bundle: Dieses Test-Bundle kann auf einem Switch erkennen, ob eine Zuweisung von einem VLAN auf ein Interface korrekt ist.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_get_vlans
    test_class: TestNapalmInterfaceInVlan
    test_data:
      - host: <hostname, required>
        vlan_tag: <tag>
        interface: <interface name>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob ein Interface dem richtigen VLAN zugewiesen ist:“host, vlan_tag, interface”

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_get_vlans
    test_class: TestNapalmInterfaceInVlan
    test_data:
      - host: S2
        vlan_tag: 200
        interface: GigabitEthernet0/3
```

2.8 T37 - Nur definierte VLANs auf Switch.

Test-Bundle: Dieses Test-Bundle kann auf einem Switch erkennen, ob nur vordefinierte VLANs vorhanden sind.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_get_vlans
    test_class: TestNapalmOnlyDefinedVlansExist
    test_data:
      - host: <hostname, required>
        vlan_tags:
          - <tag>
          - <tag>
          - <tag>
          - <tag>
          - <tag>
          - <tag>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob nur definierte VLANs auf einem Switch sind:“host, vlan_tags”

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_get_vlans
    test_class: TestNapalmOnlyDefinedVlansExist
    test_data:
      - host: S1
        vlan_tags:
          - 1
          - 200
          - 1002
          - 1003
          - 1004
          - 1005
```

2.9 T55 - Konfiguration

Test-Bundle: Dieses Test-Bundle kann auf einem Netzwerkgerät erkennen, ob die "running configuration" & die "startup configuration" identisch sind. Es kann auch geprüft werden ob die Konfiguration nicht identisch ist und somit eindeutig einen "config drift" erkennen.

Test-Bundle Struktur:

```
- test_module: nuts.base_tests.napalm_get_config
    test_class: TestNapalmConfig
    test_data:
      - host: <hostname, required>
        startup_equals_running_config: <True|False>
```

Mögliche Felder um spezifische Tests zu erstellen:

- Testet ob die running config gleich ist wie die startup config: "host, startup_equals_running_config"

Test-Bundle Beispiel:

```
- test_module: nuts.base_tests.napalm_get_config
    test_class: TestNapalmConfig
    test_data:
      - host: S1
        startup_equals_running_config: True
```

IV. Anhang

Literatur

[Byea] Kirk Byers. *Napalm Dokumentation*. URL: https://napalm.readthedocs.io/.

[Byeb] Kirk Byers. *Netmiko Dokumentation*. URL: https://ktbyers.github.io/netmiko/#api-documentation.

[Egh] Nadia Eghbal. *Working in Public: The Making and Maintenance of Open Source Software*. URL: https://www.amazon.com/dp/0578675862/.

[Hat] Red Hat. *How Ansible Works*. URL: https://www.ansible.com/overview/how-ansible-works.

[Inc] VMware Inc. *SaltStack Get Started*. URL: https://docs.saltproject.io/en/getstarted/.

[Käl] Thomas Kälin. *Testing*. FS20, [Vorlesung Software Engineering 2, Ostschweizer Fachhochschule].

[Kre] Holger Krekel. *pytest*. URL: https://docs.pytest.org/en/6.2.x/#.

[MAN] MAN. *MAN Energy Solutions*. URL: www.man-es.com.

[Sie] Méline Sieber. "Definition von Begriffen im Zusammenhang mit NUTS". In: [E-Mail].

[Tec] Nornir Tech. *Nornir Projektdokumentation*. URL: https://nornir.tech/nornir/.

[Wik] Wikipedia. *Migros*. URL: https://de.wikipedia.org/wiki/Migros.

[Zür] Stadt Zürich. *stadt-zuerich.ch*. URL: https://www.stadt-zuerich.ch/fd/de/index/das_departement/organisation/oiz.html.

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

API - Application Programming Interface
ARP - Address Resolution Protokoll
BGP - Border Gateway Protokoll
CD - Continuous Deployment
CI - Continuous Integration
CLI - Command Line Interface
CMDB - Computer Management Database
CSV - Comma-separated values
dot1x - Port Based Authentication Protokoll
GUI - Graphical User Interface
INS - Institute for Networked Solutions
IS-IS - Intermediate System to Intermediate System Protokoll
LAN - Local Area Network
LTB - Lab Topology Builder
MAB - MAC Authentication Bypass Protokoll
MAN-ES - MAN Energy Solutions
MGB - Migros Genossenschaft Bund
MPLS - Multiprotocol Label Switching
NUTS - NetTowel Network Unit Testing System
NVE - Network Virtual Interface
OIZ - Organisation und Informatik Zürich
OS - Operating System
OSPF - Open Shortest Path First Protokoll
OST - Ostschweizer Fachhochschule
PIM - Protocol Independent Multicast
RJ OST - Standort Rapperswil Jona, Ostschweizer Fachhochschule
SDN - Software Defined Networks
SNMP - Simple Network Management Protokoll
SSH - Secure Shell
UI - User Interface
VCS - Version Control System
VLAN - Virtual LAN
VNI - Virtual Network Identifier
VPN - Virtual Private Network
VSS - Virtual Switching Systems
VTEP - Virtual Tunnel Endpoint
VXLAN - Virtual Extensible LAN
WLC - Wireless Controller
ZVG - Zur Verfügung gestellt