Erstellung einer Topologie zur Durchführung von automatisierten Testszenarien in SDN-Umgebung (SDN: Software Defined Network)

Bachelorarbeit

zur Erlangung des akademischen Grades eines
Bachelor of Science (B.Sc.) in Electrical and Information Engineering

vorgelegt von
Ahmed Sammour
[redacted]

am 17.10.2022

Angefertigt im Studiengang Bachelor of Science (B.Sc.) in Electrical and Information Engineering an der Hochschule für Angewandte Wissenschaften Hamburg, Fakultät Technik

Erstprüferin: Prof. Dr. Dipl.-Ing. Freak Haase (HAW Hamburg)
Zweitprüfer: Dipl. Ing. Ali Golomohamady (Telefónica Germany GmbH & Co. OHG)

Diese Bachelorarbeit wurde betreut und erstellt in der Abteilung IP-Backbone Telefónica Germany GmbH & Co. OHG Hamburg.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Zusammenfassung

Der Einsatz von Software-Lösungen in der Netzwerktechnik ist heute nicht mehr wegzudenken. In dieser Arbeit wurde SDN für die Erstellung einer Testumgebung angewendet, mit dem Ziel verschiedene Testszenarien durchzuführen. Die erfolgreiche Erstellung der Testumgebung hat gezeigt, dass die Umstellung auf SDN realistisch und sinnvoll evaluiert werden kann. Weiterhin hat sich herausgestellt, dass die Erstellung von Topologien in NSO-Docker ebenso möglich ist. Somit können Softwareunternehmen anhand des Entwurfes solcher Testumgebungen profitieren. Diese Umgebungen können funktional den Bedingungen der realen Welt entsprechen und dennoch separat betrieben werden.

Abstract

The use of software solutions in network technology is indispensable today. In this work, SDN was applied to the creation of a test environment with the aim of performing various test scenarios. The successful creation of the test environment has shown that the transition to SDN can be evaluated realistically and meaningfully. Furthermore, it has been shown that the creation of topologies in NSO Docker is equally possible. Thus, software companies can benefit based on the design of such test environments. These environments can functionally correspond to real-world conditions and still be operated separately.

Inhaltverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AS	Autonomes System
API	Application Program Interface
BGP	Border Gateway Protocol
CDG	Crosswork Data Gateway
CI	Continuous Integration
CLI	Command-line interface,
CNC	Crosswork Network Controller
CSV	Validierung computergestützter Systeme
*csv	Comma-separated values
EU	Europäische Union
NED	Network Element Driver
NCS	Common Language Runtime
NSO	Network Services Orchestrator
PCE	Path Computational Element
UI	User Interface
VM	Virtuelle Maschine

1 Einleitung

Der Einsatz von Software-Lösungen in der Netzwerktechnik ist heute nicht mehr wegzudenken. Die drastische Zunahme der Netzwerkkomplexität in den Jahren der Existenz und Erweiterung von Computernetzwerken hat zu Schwierigkeiten bei der Verwaltung dieser Netzwerke geführt. Die Konfiguration von Computernetzsystemen nach vordefinierten Richtlinien ist oft schwierig. Denn solche Netze müssen neu konfiguriert werden, sobald Änderungen, Störungen oder Belastungen auftreten.

Durch die vertikale Integration der Netzwerksysteme sind die Steuerebene (Control plane) und die Datenebene (Data plane) miteinander verbunden. Software Defined Networking ist ein aufkommender Standard, der den aktuellen Zustand von Computernetzwerken verändert, indem er die Steuerebene des Netzes aus der Datenebene der Geräte von den grundlegenden Switches und Routern trennt. Software Defined Networking ermöglicht die einfache Erstellung und Einführung von neuen Abstraktionen in Computernetzwerken. Des Weiteren kann SDN die Netzwerkverwaltung vereinfachen und zur Evolution von Computernetzwerken beitragen.

2 Ziel der Arbeit

SDN unterscheidet sich von traditionellen Netzwerken dadurch, dass es in erster Linie softwarebasiert und nicht hardwarebasiert ist, wie es bei traditionellen Netzwerken der Fall ist. Somit ist SDN vielseitiger und ermöglicht es den Benutzern, Ressourcen virtuell über die Steuerungsebene mit größerer Freiheit und Bequemlichkeit zu verwalten.
Im Rahmen dieser Arbeit soll die Einsetzbarkeit von SDN durch eine Testumgebung untersucht werden. Dazu wird eine Topologie für die Durchführung mehrere Testszenarien im Bereich Network Services Orchestrator (NSO) erstellt.

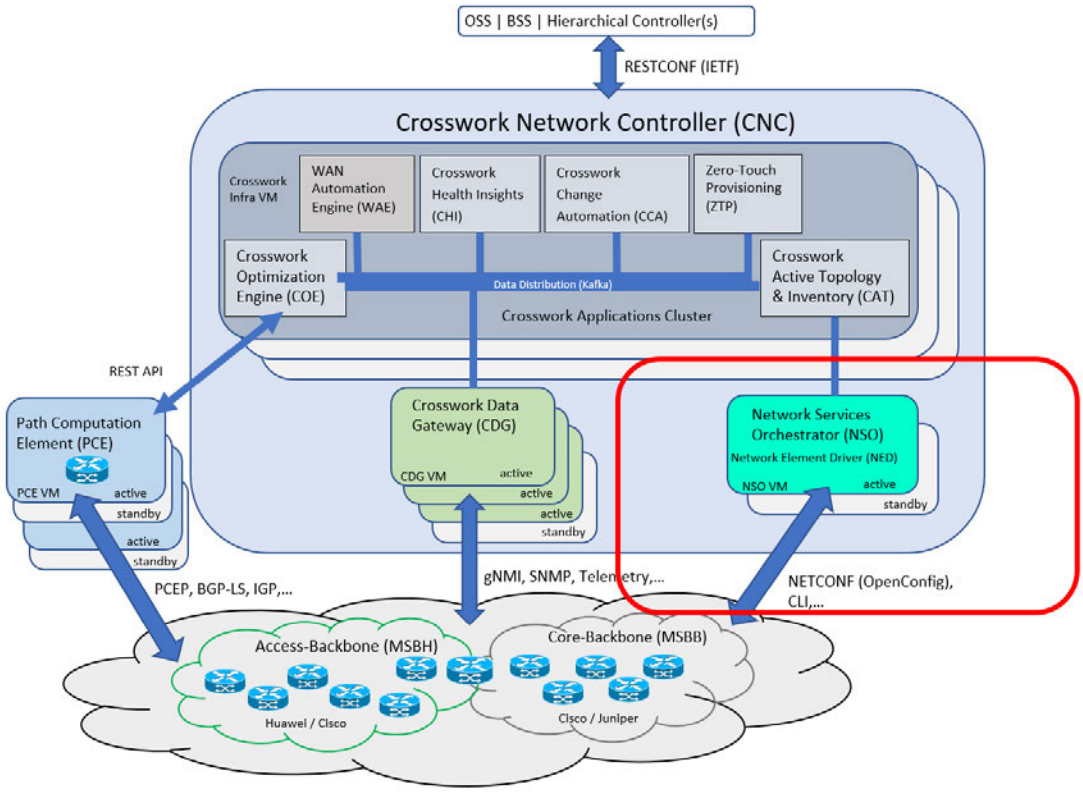

Abbildung 2.1: Hierarchical Controller(s)

Die komplette hierarchische Darstellung eines „Crosswork Network Controller“ ist aus Abbildung 2.1 zu entnehmen. Der Crosswork Data Gateway (CDG) ist für die Sammlung von Netzwerkdaten von Geräten verschiedener Hersteller zuständig [CISCO-a]. Das Path Computation Element (PCE) dagegen ist ein Pfadberechnungselement einer Systemkomponente, welche in der Lage ist, eine geeignete Route für die Übertragung von Daten zwischen einer Quelle und einem Ziel zu bestimmen und finden [Far06]. In dieser Arbeit wird auf den Network Services Orchestrator (NSO) eingegangen (Abbildung 2.2).

Diese Plattform ermöglicht die Verwaltung, Automatisierung und Orchestrierung von physischen Cisco-Geräten in Netzwerken. NSO bildet die Verbindung zwischen der Netzwerkautomatisierung und der Orchestrierung Software, um die physische und virtuelle Infrastruktur zu gründen [Cap22]. Diese Testszenarien dienen der Fehlersuche und der Sicherstellung der fehlerfreien Funktion des Netzes.

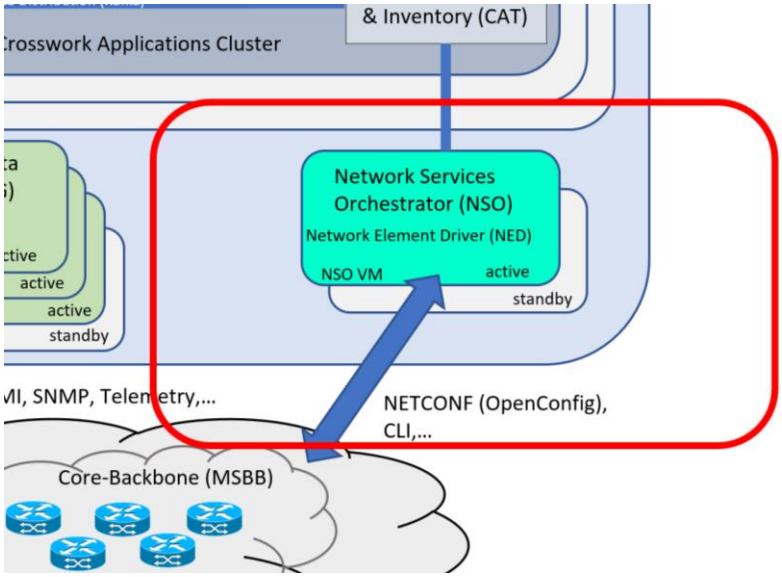

Abbildung 2.2: Network Services Orchestrator

3 Theoretischer Hintergrund

Ein tiefgründiges Verständnis in die Thematik fordert die Erläuterung der theoretischen Grundlagen zum Thema SDN und Computer Networking. Des Weiteren sollen hier Definitionen vorgestellt werden, die im Rahmen dieser Arbeit relevant sind.

3.1 Software Defined Networking

SDN ist eine Methode für die Planung, Aufbau und Wartung großer Netzwerke. Bei dieser Methode werden die Weiterleitungsentscheidungen von Kommunikations- und Netzwerkgeräten, die zwischen den Netzwerken vermittelt werden, in einer Software von einem zentralen Server aus gesteuert. Ebenfalls handelt sich um eine Methode zur Entwicklung von Hardware und Software für Computernetzwerke, bei der zwei wichtige Komponenten isoliert und abstrahiert werden. Es handelt sich dabei um die Steuerebene (Control Plane) bzw. die Datenebene (Data Plane). Im Jahr 2005 entwickelte die Stanford University die ersten Ideen dazu, anschließend haben zahlreiche Hersteller acht Jahre später ihre Geräte als SDN-fähig beworben, was durch Spekulationen zu einem Hype führte.

SDN erleichtert Netzmanagern die Verwaltung von Netzen, indem es untere Funktionsebenen zu virtuellen Diensten abstrahiert. Dies bedeutet, dass die Hardware nicht mehr manuell eingerichtet werden muss. Mit der Einführung der Virtualisierung muss ein größeres Rechenzentrum eine wachsende Anzahl virtueller Systeme im gesamten Netzwerk entwickelt und konfiguriert sowie entsprechende Firewall-Regeln und Netzwerkadressen erstellt werden. Es gibt auch andere Techniken, um virtuelle Netzwerke (VLANs) auf vergleichbare Weise zu generieren, was jedoch eine erhebliche Komplexität mit sich bringt. SDN ermöglicht es Netzwerkmanagern, den Netzwerkverkehr auf programmierbare, zentralisierte Weise abzuwickeln, ohne auf bestimmte physische Netzwerkkomponenten zugreifen zu müssen. SDN entkoppelt das System, das bestimmt, wohin Daten zu übertragen sind (die Steuerebene), von dem zugrunde liegenden System, das die Daten an das gewünschte Ziel leitet (die Datenebene). Den Erfindern und Anbietern dieser Systeme zufolge vereinfacht diese Methode die Netzwerkverwaltung und ermöglicht neue Anwendungen wie die Netzwerk Virtualisierung, bei der die Steuerungsebene von der Datenebene getrennt und als eigenständiges Programm implementiert wird.

3.2 Cisco NSO - Verbindung zwischen Intent und Action

NSO fungiert als starke Verbindung zwischen Netzwerkautomatisierungs- und Orchestrierungstechnologien und der zugrunde liegenden physischen und virtuellen Infrastruktur. Es enthält ein umfassendes Set von Software-Schnittstellen und APIs für die Nordrichtung, die eine einfache Integration ermöglichen. Da NSO über eine erweiterbare, nach Süden gerichtete Geräteabstraktionsschicht verfügt, kann es mit vielen Anbietern und Technologiebereichen interagieren.

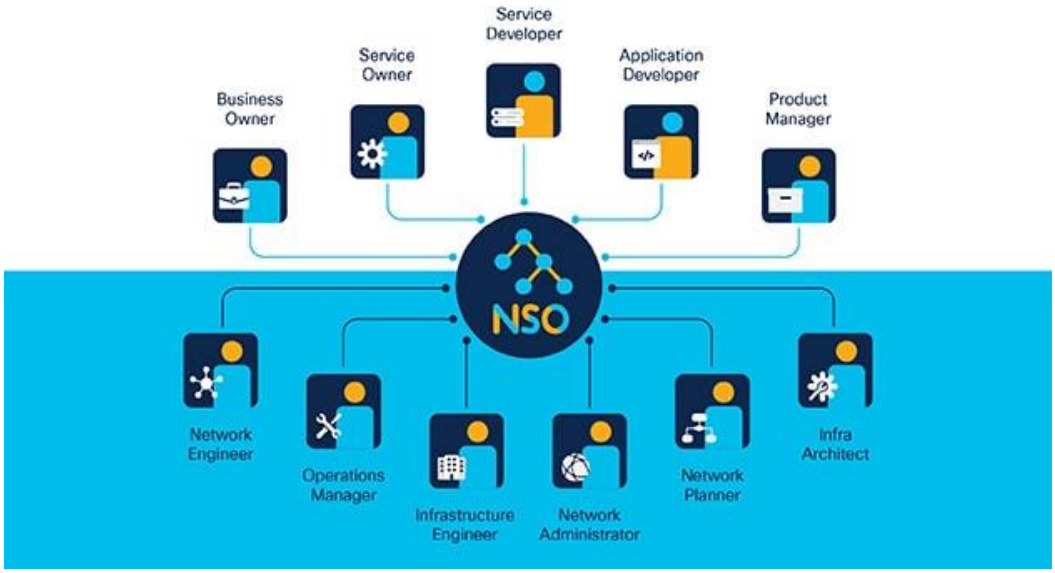

Abbildung 3.1: Cisco NSO [CISCO-b]

3.3 NSO-Docker

Die Open Networking Fundation wurde gegründet, um SDN-Standards voranzutreiben. Cloud Computing beispielsweise verwischt die Unterscheidung zwischen Netzwerk und Computing in einem technologischen Umfeld, indem SDN-Standards hilfreich erscheinen.
NSO in Docker ist ein von Cisco ins Leben gerufenes Projekt, das Benutzern von NSO ermöglicht, NSO einfach in Docker auszuführen.

Wie und warum?
Es gibt zahlreiche Gründe weshalb Docker und Container im Allgemeinen von Vorteil sein können. Die Hauptgründe für die Verwendung von Docker mit NSO sind die Paketierung, die Sicherstellung der Konsistenz beim Testen und in der Produktion, sowie die einfache und bequeme Erstellung von Testumgebungen.

- Einfache Erstellung von einem Docker-Image aus einer bestimmten Version von NSO
- NSO in einem Container zu haben, macht den Start einfach:
    - Zu testen
    - In CI zu betreiben
    - Für die Entwicklung zu verwenden
- Vorteile von NSO im Docker-Ökosystem:
    - Simple Ausführung von Netsim oder virtuellen Routern für Tests
    - Schaffung von Test- und Entwicklungsumgebungen (testenv), die gemeinsam genutzt werden können
    - Für ein effizientes Entwicklungsteam ist es wichtig, über organisierte und gemeinsam nutzbare Entwicklungsumgebungen zu verfügen.
- Es wird unter keinen Umständen Kubernetes, Docker Swarm oder eine andere ausgefallene Orchestrierung benötigt.
    - Ausnahme: Docker-Engine kann auf einem einzigen Rechner betrieben werden.

An dieser Stelle ist zu erwähnen, dass die Verwendung von Docker nicht bedeutet, dass die gesamte aktuelle Infrastruktur ersetzt werden muss. Wenn zu dem Zeitpunkt OpenStack oder ein anderes System verwendet wird, das in erster Linie mit virtuellen Maschinen arbeitet, muss diese nicht entfernt werden.

Anwendungsfälle & Leitfäden

NSO in Docker ist mehr als nur zwei Docker-Container-Images. Es handelt sich um ein Ökosystem -NID Ökosystem (NSO in Docker), das eine Art zu arbeiten ist und Probleme zu beseitigen. Das Konzept der gemeinsamen Entwicklungs- und Testumgebungen wird durch das NID-Ökosystem definiert. Es baut auf dem Fundament der NSO-Basis-Images auf:

- NED Repository
    - Einschließlich der Ausführung des NED als Netsim
    - Eine standardisierte Test- und Entwicklungsumgebung
- Paket Repository
    - Dienst oder ein anderes Paket-Repository
    - Inklusive Testen des Repositoriums
- NSO System Repository
    - Ein Repository, dass ein ganzes NSO-System umfasst
    - NSO-Dienstpakete verwendet
    - Einbindung von NEDs und anderen Paketen, die auf anderen Repositorien basieren
    - Tests auf einfache Weise zu schreiben

    - Eine standardisierte Test- und Entwicklungsumgebung haben
- Gemeinsam für alle NID-Skelette
    - Einfache Implementierung von Tests für Tests über mehrere NSO-Versionen hinweg

3.4 Docker VS virtuelle Maschine (VM)

Docker, Kubernetes und Container sind definitiv starke Technologien, die einem Unternehmen mehrere Vorteile bieten können. Sie sind jedoch nicht immer die beste Option für jede Arbeitslast. Unter bestimmten Umständen sind einfache virtuelle Maschinen vorzuziehen.

Je nach Arbeitsbelastung kann es jedoch erforderlich sein, stattdessen virtuelle Maschinen (VMs) oder eine Kombination aus Container und VMs zu verwenden. Aus diesem Grund wird auf die Frage eingegangen, ob Docker-Container oder VMs eingesetzt werden soll. Dabei liegt die Auswahl der Technologie auf dem Schwerpunkt der Art der Anwendungsfälle und Taktiken.

Zunächst werden die Gemeinsamkeiten und Unterschiede zwischen Docker und virtuellen Maschinen definiert. Docker-Container und virtuelle Maschinen sind beides Methoden zur Bereitstellung von Software in Umgebungen, die von der zugrundeliegenden Hardware isoliert sind. Der Hauptunterschied ist der Grad der Isolierung. Mit einer Container-Laufzeitumgebung wie Docker wird ihre Anwendung innerhalb der Isolationseigenschaften eines Containers in eine Sandbox eingeschlossen, während sie dennoch dasselbe Kernel wie andere Container auf demselben Host nutzt. Daher können Prozesse, die innerhalb von Containern ablaufen, vom Hostsystem gesehen werden (vorausgesetzt, die Berechtigungen reichen aus, um alle Prozesse aufzulisten). Wenn ein MongoDB-Container mit Docker gestartet und dann `ps -e|grep mongo` in einer normalen Shell auf dem Host (nicht in Docker) ausgeführt wird, wird der Prozess sichtbar. Da sich zahlreiche Container dasselbe Kernel teilen, kann der Endbenutzer eine große Anzahl von Containern auf demselben Computer mit nahezu sofortiger Startzeit unterbringen. Da Container kein ganzes Betriebssystem integrieren müssen, haben sie in der Regel etwa die Größe von 5-100 MB.

Im Gegensatz zu einer virtuellen Maschine, läuft alles unabhängig vom Host-Betriebssystem oder Hypervisor.

Die Plattform für virtuelle Maschinen initiiert einen Prozess (den sogenannten Virtual Machine Monitor oder VMM), um den Virtualisierung Prozess für eine bestimmte VM zu überwachen und das Host-System der VM einige seiner Hardware Ressourcen zuzuweisen.

Eine virtuelle Maschine hingegen bootet einen neuen speziellen Kernel für diese VM-Umgebung und beginnt beim Start eine (manchmal recht große) Sammlung von Betriebssystemprozessen. Daher ist die VM wesentlich größer als ein normaler Container, in dem lediglich die Anwendung untergebracht ist (vgl. Abbildung 3.2).

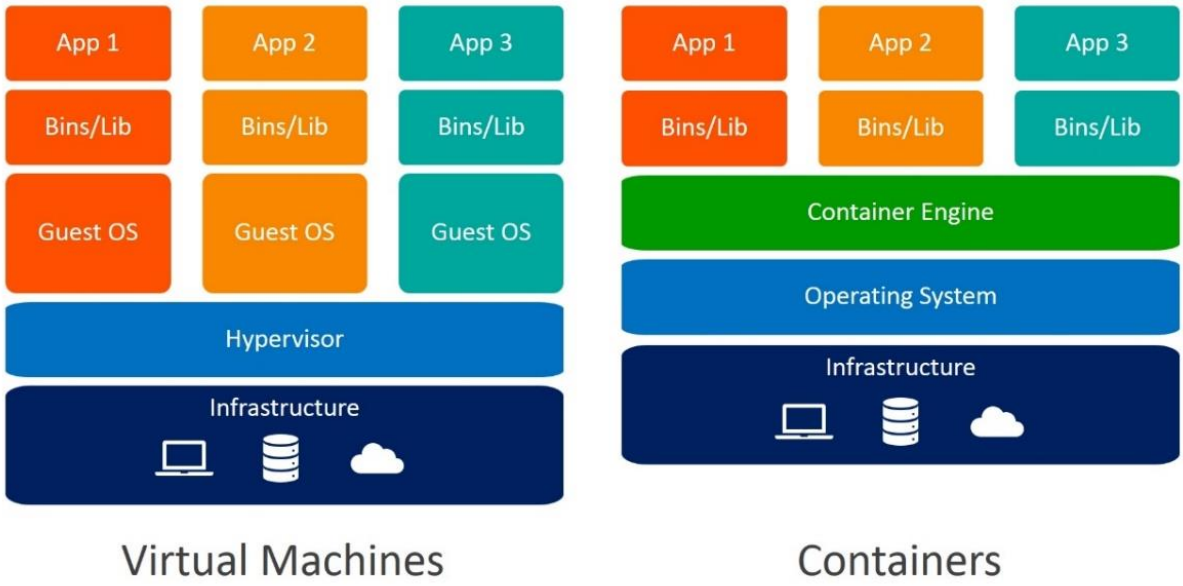

Abbildung 3.2: Containers vs virtual machines [WW22]

Die Verwendung eines speziellen Kernels und Betriebssystems bietet einige Vorteile. Einer der wichtigsten ist die Sicherheit der Isolierung. So kann das Hostsystem beispielsweise nicht wissen, was innerhalb der virtuellen Maschine vor sich geht. Da das Kernel von mehreren Containern auf demselben Host gemeinsam genutzt wird, besteht eine größere (wenn auch immer noch äußerst geringe) Gefahr, dass ein bösartiger Akteur aus seiner Eindämmung ausbricht und sich Zugang zum Host verschafft. Bei einer VM ist dies schwieriger, da das Kernel nicht gemeinsam genutzt wird und sich die Angriffsfläche auf die Hardware beschränkt.

Die Unterschiede zwischen einer Container-Laufzeitumgebung und einem Hypervisor für virtuelle Maschinen wurden im Detail dargestellt. Ebenso wurde auf die Implementierung der Snapshotting von Containern, unabhängig von der Art und Weise wie Daten in einer virtuellen Maschine gespeichert werden, eingegangen. Diese Unterscheidungen sind jedoch in der Regel weniger wichtig bei der Auswahl von Containern oder virtuellen Maschinen.

Zunächst wird erörtert, in welchen Fällen Docker eine gute Wahl sein könnte und in welchen Fällen virtuelle Maschinen besser geeignet sind oder beides verwendet werden soll.

3.4.1 Container oder virtuellen Maschinen

Für die große Mehrheit der Anwendungs-Workloads sind Container eine hervorragende Lösung. Container werden insbesondere dann in Betracht gezogen, wenn die folgenden Punkte von großer Bedeutung sind.

3.4.1.1 Start Time

Docker-Container starten in der Regel in wenigen Sekunden während virtuelle Maschinen mehrere Minuten zum Hochfahren benötigen können. Arbeitslasten, die schnell gestartet werden müssen oder bei denen Programme ständig hoch- und runtergefahren werden, sind möglicherweise für Docker geeignet.

3.4.1.2 Efficiency

Docker-Container erfordern weniger Installationen, da sie viele ihrer Ressourcen mit dem Host-System teilen. Ein Container benötigt oft weniger Platz und verbraucht weniger Arbeitsspeicher und CPU-Zeit als eine virtuelle Maschine. Daher kann mit Container häufig mehr Anwendungen auf einem einzigen Server als mit virtuellen Maschinen untergebracht werden. Aufgrund ihres geringeren Ressourcenverbrauchs können Container auch dazu beitragen, die Kosten für Cloud Computing zu senken.

3.4.1.3 Licensing

Die meisten Schlüsseltechnologien, die für den Einsatz von Docker-Containern erforderlich sind, wie z. B. Container-Laufzeiten und Orchestratoren wie Kubernetes, sind kostenlos und quelloffen. Dies kann sowohl zu Kosteneinsparungen als auch zu einer höheren Flexibilität führen. Es sei jedoch darauf hingewiesen, dass sich viele Unternehmen für eine kommerzielle Version von Docker oder Kubernetes entscheiden, um die Bereitstellung zu vereinfachen und professionelle Supportdienste in Anspruch zu nehmen.

3.4.1.4 Code reuse

Jeder Betriebscontainer basiert auf einem Container-Image, das die für die Ausführung einer bestimmten Anwendung erforderlichen Binärdateien und Bibliotheken enthält. Mit Docker Files lassen sich Container-Images einfach erstellen. Container-Registries, die im Wesentlichen Repositorien sind, in denen Container-Images gespeichert werden, ermöglichen deren gemeinsame Nutzung und Wiederverwendung. Nach der Einrichtung eines internen Registry kann ein Container innerhalb einer Organisation gemeinsam genutzt und wiederverwendet

werden. Tausende von vorgefertigten Images können kostenlos von öffentlichen Registern (wie Docker Hub oder Quay.io) heruntergeladen und für die Erstellung eigener containerisierter Anwendungen verwendet werden.
Natürlich können auch VMs in Images gepackt und diese Images können ebenfalls gemeinsam genutzt werden, aber nicht so effektiv oder einfach wie Container. Außerdem lassen sich die Abbilder virtueller Maschinen nicht so leicht automatisch erstellen und sind in der Regel größer. Da sie in der Regel auch Betriebssysteme enthalten, kann ihre Weitergabe rechtlich kompliziert werden.

3.4.2 Virtuellen Maschinen

Einige Gründe, warum auf Docker-Container verzichtet werden soll:

- Security
- Mixing and matching Linux and Windows portability
- Rollback Features

3.5 Python

Dank seiner einfachen Syntax und großen Flexibilität ist Python eine der am häufigsten verwendeten Programmiersprachen und wird oft als Goldstandard für Data Science und maschinelles Lernen angesehen, insbesondere für Deep Learning. Die Data-Science Bibliotheken und Tools von Python, wie Pandas, Matplotlib, Seaborn, und Numpy, haben zur Bildung einer großen Data-Science-Community geführt.
Python, einschließlich Keras, Tensorflow und Pytorch, verfügt über die meisten Bibliotheken für maschinelles und tiefes Lernen, die von bedeutenden Unternehmen wie Google und Facebook erstellt und genutzt werden. Somit kann Python einen sehr einfachen Einstieg in den Bereich der künstlichen Intelligenz ermöglichen und schnell fruchtbare Ergebnisse erzielen, die völlig neue Formen kultureller Bestrebungen ermöglichen.

In dieser Arbeit wird für die Testszenarien Yaml und Ansible, welche Python basiert sind, verwendet.

4 Problemstellung

In diesem Kapitel wird auf die Unterschiede zwischen SDN und den normalen Netzwerken eingegangen. Diese Gegenüberstellung soll einen überschlägigen Überblick über die Vor- und Nachteile von den beiden Systemen geben.

4.1 SDN vs Normales Netzwerk

Normale Netzwerke unterscheiden sich von SDN dadurch, dass sie softwarebasiert sind, während normale Netzwerke oft hardwarebasiert sind. Da SDN softwarebasiert ist, bietet es den Benutzern mehr Freiheit und Leichtigkeit bei der Steuerung von Ressourcen in der gesamten Steuerungsebene. Normale Netzwerke hingegen verwenden Switches, Router und andere physische Infrastrukturen zur Verbindung und Verwaltung des Netzwerks.

SDN-Controller sind über eine Northbound-Schnittstelle mit APIs verbunden. Aufgrund dieser Verbindung können Anwendungsentwickler das Netzwerk direkt programmieren, anstatt die für normale Netzwerke erforderlichen Protokolle zu verwenden. Da Benutzer neue Geräte über Software und nicht über die physische Infrastruktur bereitstellen können, ermöglicht SDN den IT-Managern, Netzwerkkanäle zu steuern und Netzwerkdienste proaktiv zu organisieren. Im Gegensatz zu normalen Switches kann SDN auch eine effektivere Verbindung zu vernetzten Geräten herstellen.

Die Virtualisierung ist ein Beispiel für den Hauptunterschied zwischen SDN und normalen Netzwerken. Sollte SDN einsetzen werden, um das gesamte Netzwerk zu virtualisieren, so wird ein abstrakter Klon des physischen Netzwerks erstellt, das von einem einzigen Punkt gesteuert werden kann.

Im Gegensatz dazu macht es die physische Positionierung der Steuerungsebene in einem normalen Netzwerk einem IT-Administrator unmöglich, den Verkehrsfluss zu regulieren.
Mit SDN wird die Steuerebene softwarebasiert und ist damit über ein angeschlossenes Gerät verfügbar. Dieser Zugriff ermöglicht es IT-Managern, den Verkehrsfluss über eine zentrale Benutzeroberfläche (UI) effektiver zu regulieren. Dank dieser zentralen Stelle haben die Benutzer mehr Kontrolle über die Leistung und Konfiguration ihrer Netzwerke. Die Möglichkeit, zahlreiche Netzwerkeinstellungen schnell über eine einzige Benutzeroberfläche zu verwalten, ist besonders wichtig für die Netzwerksegmentierung.

SDN ist zu einer beliebten Alternative zu herkömmlichen Netzwerken geworden, da es IT-Managern ermöglicht, Ressourcen und Bandbreiten nach Bedarf bereitzustellen, ohne in zusätzliche physische Infrastruktur investieren zu müssen. Normales Networking erfordert den Kauf von zusätzlicher Hardware, um die Netzwerkkapazität zu erhöhen.
Aus diesem Grund wird in dieser Arbeit die Testumgebung anhand SDN erstellt und durchgeführt.

5 Vorgehensweise

In diesem Kapitel werden die Vorbereitung, der Aufbau und die komplette Vorgehensweise bei der Erstellung der Topologie behandelt.

5.1 Vorbereitung des Servers

Für das Projekt musste zunächst ein externer Server eingerichtet werden, um mögliche Schäden am laufenden Netz zu vermeiden.
Der Server soll das reale Netz simulieren und einem Forscher ermöglichen, ein kompliziertes heterogenes Telekommunikationsnetz auf einem einzigen Host-Computer zu betreiben. Darüber hinaus soll der Server folgende Fähigkeiten besitzen.

1. Der Server soll ein simuliertes Netz auf einem einzigen physischen Computer mit vier oder mehr virtuellen Hosts und/oder virtuellen Router, die über virtuelle Verbindungen miteinander verbunden sind, aufbauen.
2. Er verbindet die Netzschnittstellen mit diesem virtuellen Router in verschiedenen vom Benutzer gewählten Konfigurationen.
3. Er wird Software enthalten, die auf jeden virtuellen Router arbeiten kann und die Aufgaben der Paketweiterleitung übernimmt.
4. Ferner enthält der Server Software, die auf jedem virtuellen Router läuft und Host- sowie Server-Netzwerkaktivitäten durchführt.
5. Außerdem wird er über Skripte verfügen, die es ermöglichen, die Konfiguration verschiedener Konfigurationen verbundener virtueller Maschinen zu automatisieren, um die Funktionalität auf jeder virtuellen Maschine anzupassen.
6. Des Weiteren bietet er die Möglichkeit Laborsituationen zu konstruieren, die gespeichert und in Zukunft wiederverwendet werden können.
7. Darüber hinaus ist es von Vorteil, wenn das Tool bereits über einige vordefinierte Laborsituationen verfügt, die es dem Benutzer ermöglichen, das Tool zu testen, ohne sich in die Komplexität der Anpassung einarbeiten zu müssen.
8. Somit kann der Status einer Lab-Sitzung gespeichert werden und die Arbeit kann bei dem letzten Zustand fortgesetzt werden.
9. Auf den virtuellen Hosts und Routern können eine Reihe von netzwerkbezogenen Anwendungen ausgeführt werden.
10. Die Softwareeinstellungen kann auf jedem virtuellen Host oder virtuellen Router über ein Terminal anzeigt und/oder geändert werden.

11. Die Leistung der Benutzeroberfläche muss "akzeptabel" sein. Das Starten eines neuen Routers sollte zum Beispiel weniger als eine Minute sowie das Einloggen in einen bestehenden Router weniger als 5 Sekunden und das Drücken der Eingabetaste in der Befehlszeile eines Routers weniger als eine Sekunde dauern.
12. Das Programm ist ein "nützliches" Hilfsmittel für die Vermittlung von Netzwerkideen an andere und/oder für das Selbststudium von Netzwerkkonzepten.

Ein passender Server, der diese Punkte erfüllt, wird wie folgt eingerichtet. Zunächst wurde der Server zurückgesetzt und mit dem Netzwerk von Telefónica verbunden. Anschließend wurde ein neues Betriebssystem installiert, das von Telefónica gesichert wurde. Abbildung 5.1 stellt die technischen Eigenschaften des Servers dar. Der Server verfügt über folgenden CPU-Einheit, sowie in Abbildung 5.2 dargestellt ist.

```
H/W path             Device      Class          Description
============================================================
                                 system         PowerEdge R630 (SKU=NotProvided;ModelName=PowerEdge R630)
/0                               bus            0CNCJW
/0/0                             memory         64KiB BIOS
/0/400                           processor      Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz
/0/400/700                       memory         640KiB L1 cache
/0/400/701                       memory         2560KiB L2 cache
/0/400/702                       memory         25MiB L3 cache
/0/401                           processor      Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz
/0/401/703                       memory         640KiB L1 cache
/0/401/704                       memory         2560KiB L2 cache
/0/401/705                       memory         25MiB L3 cache
/0/1000                          memory         192GiB System Memory
/0/1000/0                        memory         8GiB DIMM DDR4 Synchronous Registered (Buffered) 2666 MHz (0.4 ns)
```

Abbildung 5.1: Server Eigenschaften

```
 Socket Designation: CPU1
 Type: Central Processor
 Family: Xeon
 Manufacturer: Intel
 ID: F2 06 03 00 FF FB EB BF
 Signature: Type 0, Family 6, Model 63, Stepping 2
Version: Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz
Voltage: 1.3 V
External Clock: 9600 MHz
Max Speed: 4000 MHz
Current Speed: 2600 MHz
Status: Populated, Enabled
Upgrade: Socket LGA2011-3
L1 Cache Handle: 0x0700
L2 Cache Handle: 0x0701
```

Abbildung 5.2: Server CPU-Daten

5.2 Server-Betriebssystem

Die Server-Edition der beliebten Linux-Distribution Ubuntu bietet eine einfache Grundlage für zahlreiche Serversituationen wie E-Mail-Server, Web-hosting-Plattform oder File-Server. "Ubuntu Server Edition" verzichtet auf eine grafische Benutzeroberfläche und ist daher äußerst ressourcenschonend.
Eine 1-Gigahertz-CPU, 1 Gigabyte Arbeitsspeicher und 2 Gigabyte Festplattenspeicher sind vollkommen ausreichend für einen reibungslosen Betrieb [Ubuntu-a].

5.2.1 Installation

Die Installation erfolgt in den folgenden Schritte:

1. Ubuntu Server wird heruntergeladen (siehe Abbildung 5.3).

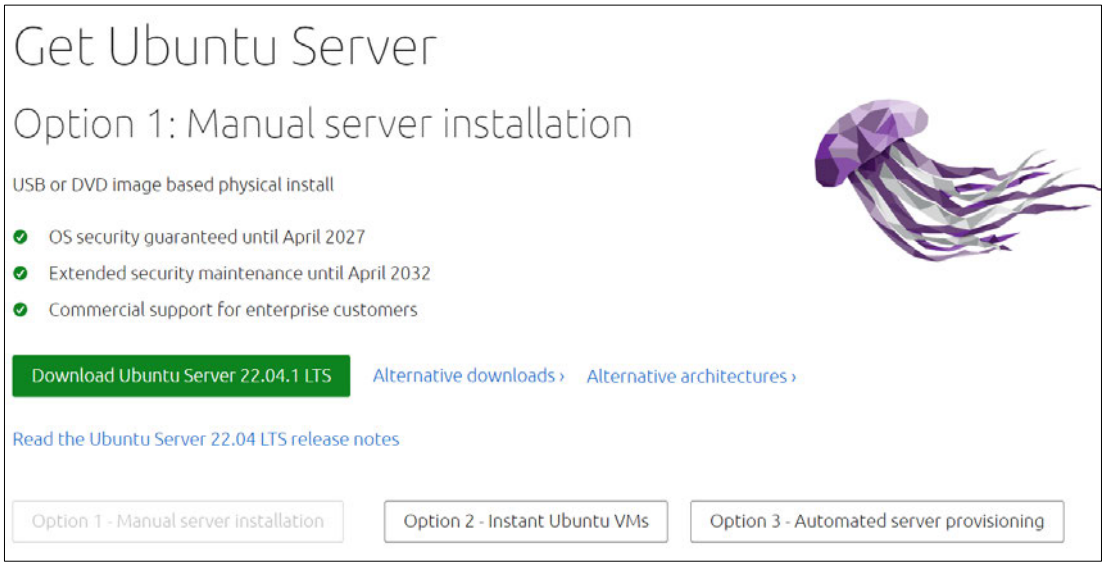

Abbildung 5.3: Ubuntu Server Download [Ubuntu-a]

2. Der Installationsvorgang wird wie folgt starten: USB-Stick oder ein anderes Installationsmedium wird am Server angeschlossen. Nach ein paar Sekunden sollte auf dem Bildschirm eine ähnliche Meldung wie in Abbildung 5.4 dargestellt ist.

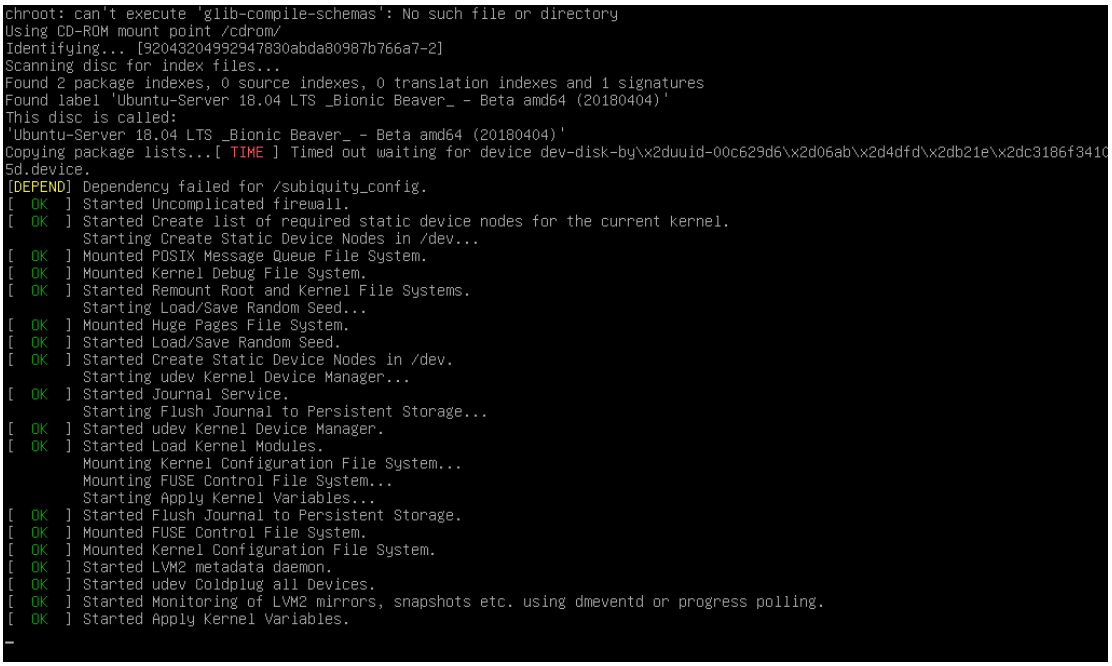

Abbildung 5.4: Ubuntu Server installieren [Ubuntu-b]

3. Nun kann die gewünschte Sprache ausgewählt werden.

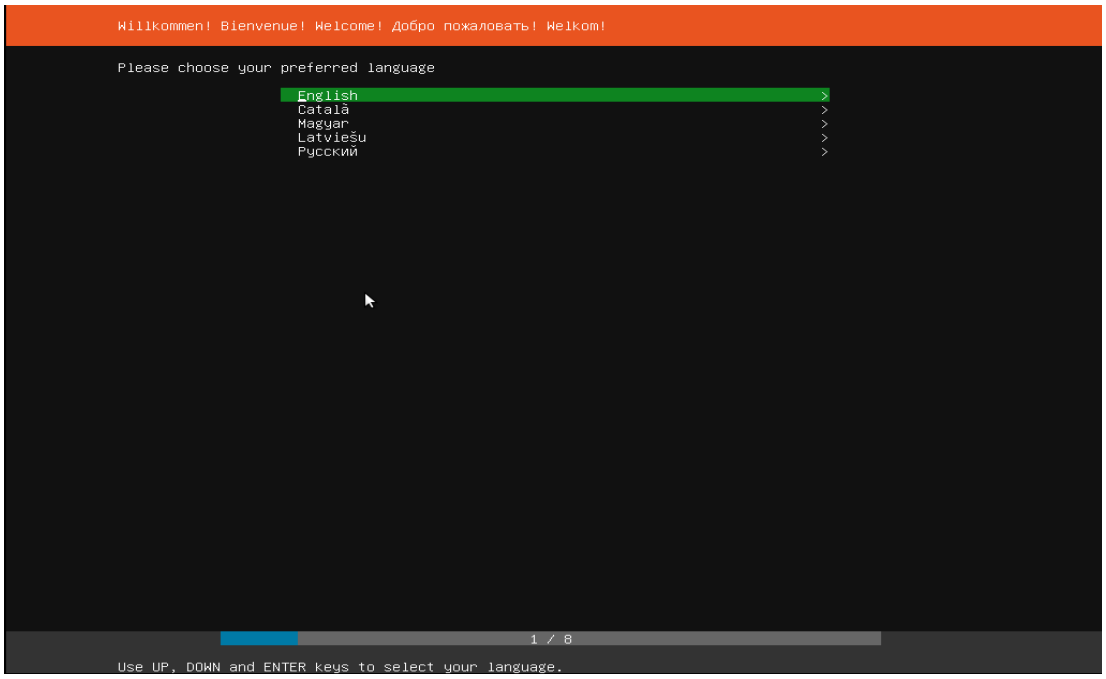

Abbildung 5.5: Ubuntu Server Installation - Sprachauswahl [Ubuntu-b]

4. Jetzt kann „Install Ubuntu“ ausgewählt werden.

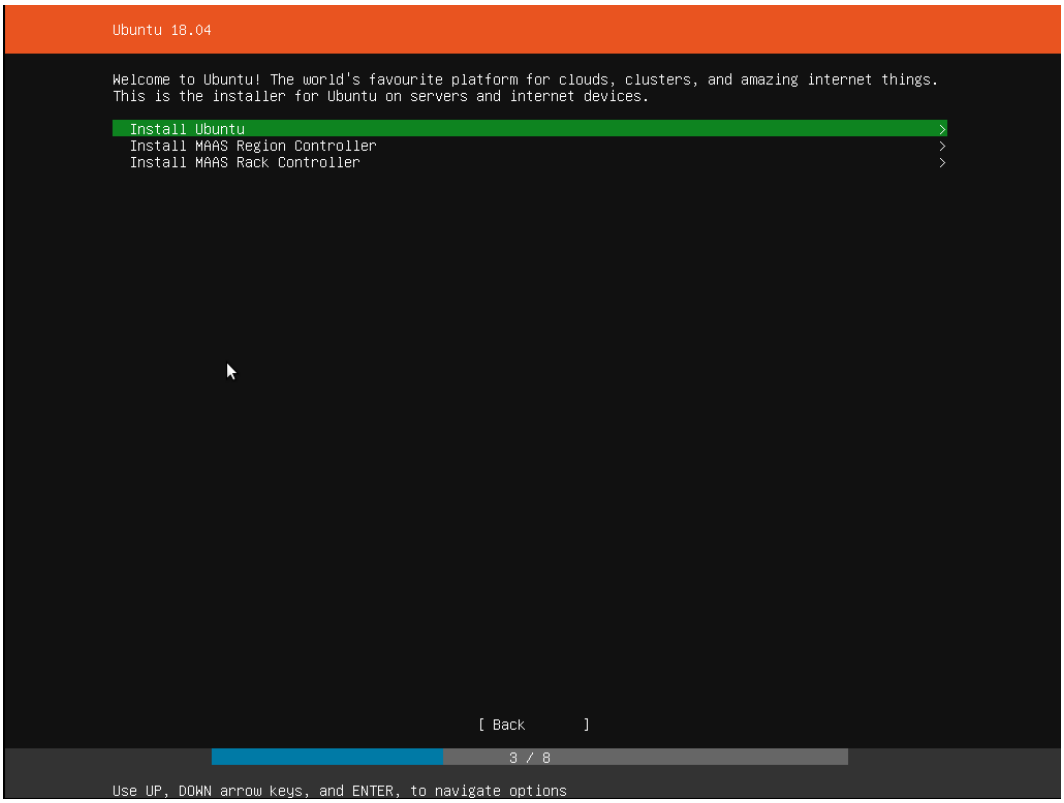

Abbildung 5.6: Ubuntu Server Installation - Version [Ubuntu-b]

Die beiden unteren Optionen werden für die Installation bestimmter Komponenten einer Metall As A Service (MAAS)-Installation verwendet.

5. Der nächste Schritt ist die Konfiguration des Speichers. Es wird empfohlen, eine ganze Festplatte oder Partition für die Ausführung von Ubuntu zu reservieren.

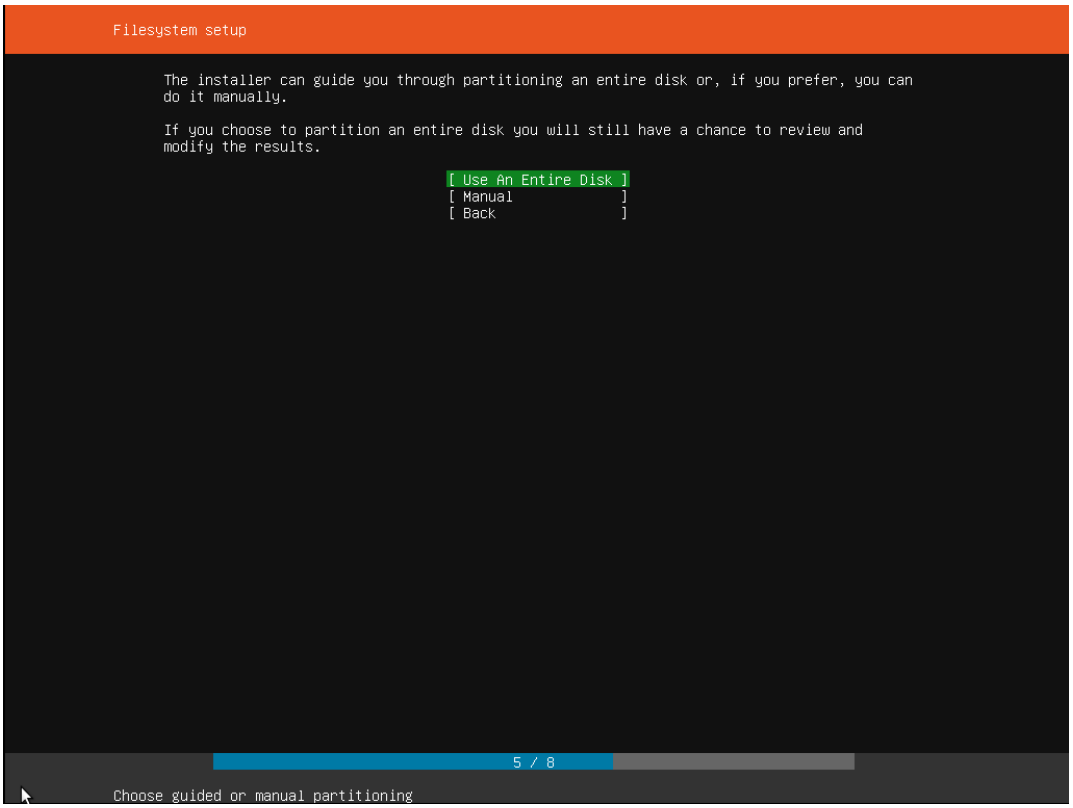

Abbildung 5.7: Ubuntu Server Installation - Speicher Festlegung [Ubuntu-b]

6. Die Software wird nun auf der Festplatte installiert, dennoch benötigt das Installationsprogramm noch einige weitere Informationen. Der Ubuntu Server benötigt mindestens einen bekannten Benutzer für das System und einen Hostnamen. Zu dem Benutzer soll ein Passwort festgelegt werden.

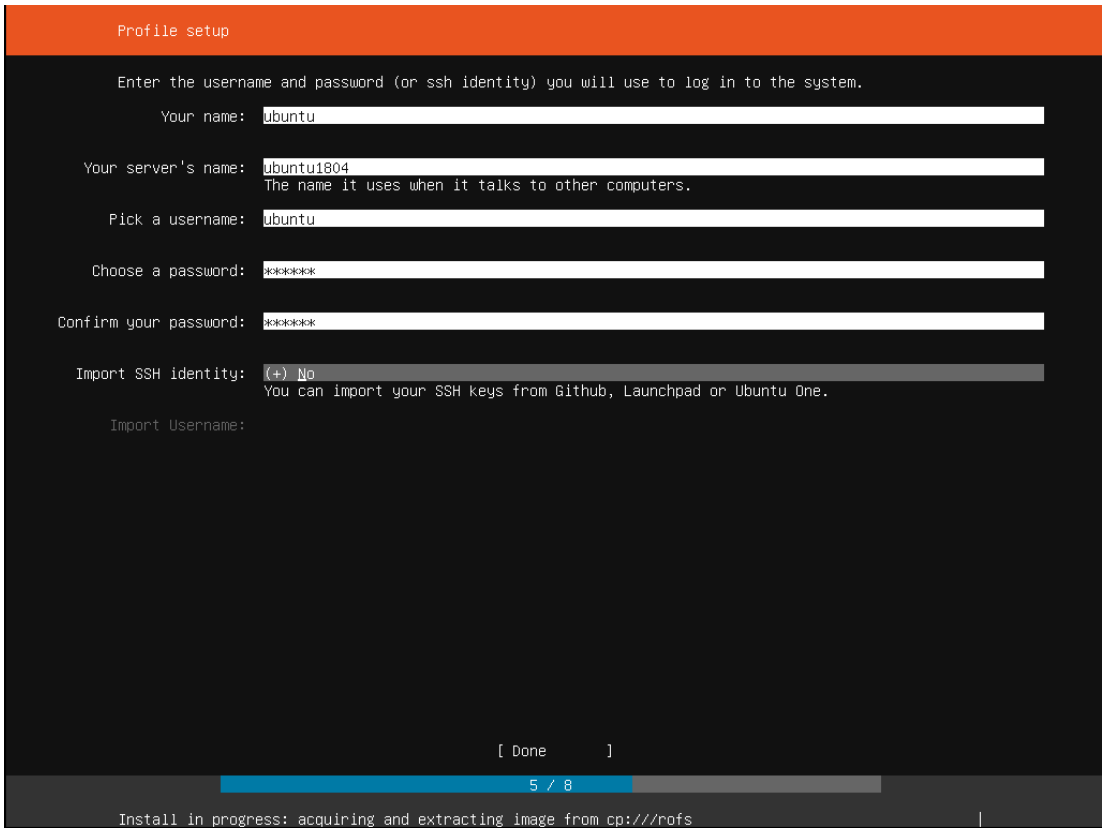

Abbildung 5.8: Ubuntu Server Profile einlegen [Ubuntu-b]

Im Anschluss werden einige Dateien automatisch installiert und Ubuntu ist somit einsatzbereit.

5.3 Docker-NSO Einrichtung

5.3.1.1 Voraussetzungen

Der NSO in Docker muss auf folgende Betriebssysteme laufen:

- Linux
- Mac OS
- Windows

Um ein Docker-Image aus einer bestimmten Version von NSO und Ihren Paketen zu bilden, werden ein Docker, ein Make und ein realpath installiert. Docker realpath wird dann anhand folgender Befehle installiert:

- `Mac OS X: apt install coreutils`
- `Debian: apt install coreutils brew install coreutils`

Die Ausführung von Testsuite fordert die Eingabe folgender Befehle:

- `expect`
- `ssshpass`

5.3.1.2 Verwendung

Idealerweise werden vorgefertigte Docker-Images mit NSO ausgeliefert, jedoch aufgrund rechtlicher Einschränkungen ist dies die nächstbeste Option. Die NSO Docker Repository enthält Rezepte für die Erstellung eigener Docker-Images.

5.3.1.3 Einrichtung

Die manuelle Erstellung von Docker-Images auf dem lokalen Rechner kann anhand folgender Schritte durchgeführt werden:

- Folgendes Verzeichnis sollte auf den lokalen Rechner geklont werden [Docker]
  - `git clone https://gitlab.com/nso-developer/nso-docker.git`

- Dann wird NSO Cisco heruntergeladen
  - `https://developer.cisco.com/docs/nso/#!getting-nso/getting-nso.`

- Wenn die Dateiendung `signed.bin` lautet, handelt es sich um ein selbstextrahierendes Archiv, das eine Signatur verifiziert. Dies wird ausgeführt, um das Installationsprogramm zu generieren.
  - Wenn beispielsweise `bash nso-5.7.1.linux.x86_64.signed.bin` ausgeführt wird, werden mehrere Dateien erzeugt, darunter die Installation `nso-5.7.1.linux.x86 64.installer.bin`.

- Die Datei `nso-5.x.linux.x86_64.installer.bin` wird in das Verzeichnis `nso-install-files/` gezogen.

- Der Befehl `make` wird im Hauptverzeichnis ausgeführt, um Docker-Images aus allen gefundenen NSO-Installationsdateien zu erstellen.
  - HINWEIS: Die Ausführung von Docker-Befehlen, die von `make` aufgerufen werden, erfordert in der Regel Root-Rechte oder die Mitgliedschaft in einer Docker-Gruppe.
  - Docker-Images werden mit einem Suffix versehen, z.B. cisco-nso-base:5.7.1-asammour, wenn der Benutzername asammour ist. Das Suffix soll verhindern, dass ein Versions-tag wie cisco-nso-base:5.7.1 überschrieben wird, bevor das Image getestet und für gut befunden wurde. Der Befehl `make tag-release` wird

ausgeführt, um ein Docker-Tag ohne Suffix hinzuzufügen, wie cisco-nso-base:5.7.1.

- `make tag-release` wird ausgeführt und die Version von tag mit der Variable NSO_VERSION wird angegeben.

5.3.1.4 Ausführung

NSO ist für Linux auf den Plattformen x8664 / amd64 kompiliert. Wenn eine andere CPU-Architektur verwendet wird, wie z. B. das Apple M1-Silizium, muss der Container mit dem Argument `-Plattform=linux/amd64` ausgeführt werden. Der Start von NSO Standalone zum Testen wäre zum Beispiel wie folgt:

```
docker run -itd --platform=linux/amd64 --name nso-dev1 my-prod-image:12345
```

Wenn ein Produktions-Image erstellt wird, d.h. das Basis-Image aus diesem Repo verwendet und ein eigenes Paket hinzugefügt wird, kann es zu Testzwecken eigenständig ausgeführt werden.

Eigenständiges starten eines Containers

Docker-Netzwerke werden verwendet, um sich mit anderen Docker-basierten Diensten wie Netsim zu verbinden.

```
nso-dev1 my-prod-image:12345 docker run -itd
```

Die Aufnahme des Quellcode-Verzeichnisses in den Container ist zu empfehlen, wenn dieser für die Entwicklung ausgeführt wird. Dies ermöglicht dem Container die Verwendung eines Compilers und anderer Tools. Die Installation von Compilern und anderen Tools direkt auf dem Computer soll vermieden werden.

```
nso-dev1 -v $(pwd):/src cisco-nso-dev:5.2 docker run -itd
```

Für die Produktion soll mit einem Produktions-Image begonnen werden, d.h. das Basis-Image aus diesem Repo und in eigenes Paket hinzugefügt werden. Ein Shared Volume wird verwendet, um die Daten zwischen den Reboots zu sichern. CDB (NSO-Datenbank) SSL- und SSH-Schlüssel NETCONF-Benachrichtigungswiedergabe inklusive Rollback-Backups und NSO-Protokolle sind optional.

Wenn die Fernprotokollierung (syslog) verwendet wird, gibt es kaum einen Grund, Protokolle zu führen. Wenn lokale Protokollierung verwendet wird, ist es eine gute Idee, Protokolle zu führen. Die Verwendung von `-net=host`, um eine IP-Adresse mit dem Host-Rechner zu teilen, erleichtert die Verwaltung der Konnektivität.

Die Option `-net=host` wird verwendet, um dem Container zu erlauben, im Netzwerk-Namensraum des Hosts zu leben. Das bedeutet, dass er sich mit der IP-Adresse des (virtuellen) Rechners verbindet, auf dem er läuft.
NSO ist so eingerichtet, dass die CLI über SSH auf Port 22 zugänglich ist. Die Verwendung von `-net=host` führt zu einer Kollision, wenn SSH auf der VM ausgeführt wird. NSO kann so umkonfiguriert werden, dass es auf einem anderen Port lauscht, um Port-Kollisionen zu vermeiden, indem die Umgebungsvariable SSH PORT gesetzt wird.

```
docker run -itd --name nso -v /data/nso:/nso -v /data/nso-logs:/log --net=host -e SSH_PORT=2024 my-prod-image:12345
```

NSO-Konfigurationsmanagement

Es gibt mehrere Ansätze für den Umgang mit ncs.conf in NSO in Docker; idiomatischer Container-Ansatz mit konfigurierbaren Optionen über Umgebungsvariablen eine von zwei Methoden kann verwendet werden, um eine vorhandene ncs.conf zu importieren. ncs.conf wird auf dem Volume platziert, dass im Container nach `/nso` gemountet ist, unter `/nso/etc/ncs.conf`. wird als direkt gemountete Datei unter `/etc/ncs/ncs.conf` injiziert. Diese Strategie ist sehr einfach. eine Konfigurationsdatei wird in `/etc/ncs/ncs.conf` installiert.

```
docker run -itd --name nso -v /data/nso-config/my-nso-config.conf:/etc/ncs/ncs.conf -v /data/nso:/nso -v /data/nso-logs:/log --net=host my-prod-image:12345
```

5.4 Cisco NSO Implementierung

Das Erstellen und Konfigurieren von Netzwerkdiensten ist ein schwieriger Prozess, der oft viele Konfigurationsänderungen auf allen Geräten erfordert, die den Dienst nutzen. Außerdem müssen die Änderungen in der Regel auf allen Geräten gleichzeitig vorgenommen werden und entweder vollständig erfolgreich sein oder zur ursprünglichen Einrichtung zurückkehren. Darüber hinaus muss die Konfiguration des Systems und der Netzwerkgeräte konsistent bleiben. NSO löst diese Probleme, indem es als Schnittstelle zwischen den Netzwerkgeräten und jeder Software oder jedem Benutzer, der konfiguriert wurde, dient.

Cisco NSO ist sowohl mit MacOS als auch mit Linux-Systemen kompatibel. Für dieses Projekt wurde NSO jedoch in einem Linux-Container installiert. Kleine Labornetzwerke (weniger als 10 Geräte) erfordern in der Regel nur eine einzige vCPU und 4-8 GB RAM. Sollten bei der Nutzung von NSO Probleme auftauchen, dann soll die Größe des Arbeitsspeichers erhöht werden, da es sich um eine speicherintensive Anwendung handelt.

Damit NSO funktioniert muss Java installiert werden. Hier sind einige gängige Methoden zur Installation von Java:

```
apt - apt install default-jdk
yum - yum install java-1.8.0-openjdk
```

Apache Ant ist eine Java-Bibliothek und ein Werkzeug zum Erstellen von Dateien und anderen Abhängigkeiten, die benötigt werden.

```
apt - apt install ant
yum - yum install ant
```

Durchführen einer lokalen Installation

Zunächst wird in einem Terminal das Installationsverzeichnis ausgewählt. Die Datei signed.bin wird ausgeführt damit das Zertifikat überprüft werden kann und somit die Binärdatei des Installationsprogramms und andere Dateien extrahiert wird.
Nachdem die letzten Schritte erfolgreich abgeschlossen sind, kann die Installation gestartet werden. Abbildung 5.9 stellt eine Liste der entpackten Dateien dar.

```
1.8K Oct 09 06:05 README.signature
 12K Oct 09 06:05 cisco_x509_verify_release.py
199M Oct 09 05:55 nso-5.7.1.darwin.x86_64.installer.bin
256B Oct 09 06:05 nso-5.7.1.darwin.x86_64.installer.bin.signature
199M Oct 09 11:45 nso-5.7.1.darwin.x86_64.signed.bin
1.4K Oct 09 06:05 tailf.cer
```

Abbildung 5.9: entpackte Dateien

`nso-VERSION.OS.ARCH.installer.bin` ist das NSO-Installationsprogramm.
`nso-VERSION.OS.ARCH.installer.bin.signature` wurde für das NSO-Image generiert.

Ein von Cisco signiertes tailf.cer x.509 Endbenutzerzertifikat mit dem öffentlichen Schlüssel, der zur Verifizierung der Signatur verwendet wird, ist beigefügt. Die Datei README.signature enthält weitere Informationen über den entpackten Inhalt sowie eine Anleitung zum Ausführen des Signaturprüfungsprogramms. Die Schritte in dieser Datei können befolgt werden, falls die Signatur manuell überprüft werden soll. Das Python-Programm cisco x509 verify release.py kann verwendet werden, um die dreistufige x.509-Zertifikatskette und die Signatur zu überprüfen.

Zunächst wird folgender Befehl verwendet, um die Hilfedatei für das Installationsprogramm

```
sh nso-5.7.1darwin.x86 64.installer.bin -help
```

zu erhalten. Es soll auf die Optionen -local-install und -system-install geachtet. Das Installationsverzeichnis, oder LocalInstallDir, sollte in dem $HOME-Verzeichnis in einem Ordner namens /nso-VERSION installiert werden. Wenn die installierte Version beispielsweise 5.7.1 ist, lautet das Verzeichnis /nso-5.7.1. Damit die Installation in das Heimatverzeichnis erfolgen kann, wird das Installationsprogramm mit dem Argument -local-install /nso-5.7.1 ausgeführt.

5.5 NCS & NEDS

NSO verwendet NEDs als Gerätetreiber für verschiedene Gerätetypen, um mit dem Netzwerk zu "sprechen". Kunden können NEDs für Hunderte von verschiedenen Geräten von Cisco beziehen und einige sind im Installationsprogramm im Verzeichnis `/nso-5.7.1/packages/neds` enthalten.

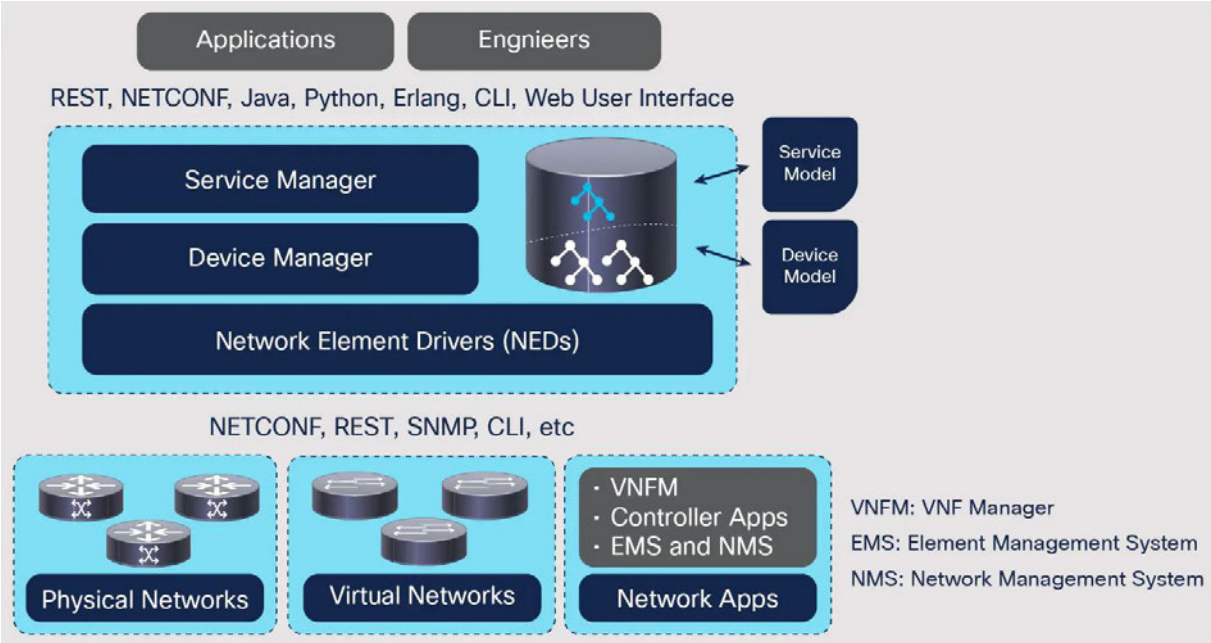

Abbildung 5.10: NED-Übersicht

Der netzseitige Teil von NSO besteht aus Netzelementtreibern. Sie tauschen Daten unter Verwendung des geräteeigenen Protokolls aus, z. B. Network Configuration Protocol (NETCONF), Representational State Transfer (REST), Extensible Markup Language (XML), CLI und Simple Network Management Protocol (SNMP).

Die NEDs sind wie das NSO-Installationsprogramm signed.bin Dateien, die ausgeführt werden müssen, um den Download zu validieren und den neuen Code zu extrahieren. Zunächst wird

die heruntergeladenen Dateien gesucht. Um das Zertifikat zu überprüfen und die NED tar.gz und andere Dateien zu entpacken, kann den Befehl `sh` verwendet werden um die Datei signed.bin auszuführen. Es sollten nun drei Tar-Dateien (. tar.gz) geben. Dies sind NED-komprimierte Versionen. Zum Verzeichnis `local-packages/neds-Installation` wird navigiert. Im Verzeichnis `/nso-5.7.1/packages/neds` soll der Befehl `tar` mit dem Pfad zu den komprimierten NED verwendet werden, um die Tar in dieses Verzeichnis zu entpacken.

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Arbeit dargestellt. Zunächst können die Daten und die Erfolgreiche Implementierung von Docker den Abbildung 6.1 entnehmen werden.

```
asammour@toastie:~$ ssh boip-deham-95-0004
asammour@boip-deham-95-0004's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-117-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Oct 10 19:41:16 UTC 2022

  System load:    2.97               Users logged in:          1
  Usage of /home: 10.2% of 255.88GB  IPv4 address for br_test: 192.168.11.254
  Memory usage:   12%                IPv4 address for docker0: 172.17.0.1
  Swap usage:     0%                 IPv4 address for eno1:    172.31.4.127
  Temperature:    57.0 C             IPv4 address for virbr0:  192.168.122.1
  Processes:      634

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Mon Oct 10 16:44:28 2022 from 172.30.1.201
asammour@boip:~$
```

Abbildung 6.1: Ausschnitt Docker Daten

Aus der Abbildung 6.2 kann die in dem Docker installierte Images entnommen werden. Die Images werden benötigt, um Containers erstellen zu können.

```
asammour@boip:~$ docker images
REPOSITORY                                TAG              IMAGE ID       CREATED         SIZE
busybox                                   latest           c98db043bed9   5 weeks ago     1.24MB
nginx                                     latest           2b7d6430f78d   6 weeks ago     142MB
alpine                                    latest           9c6f07244728   2 months ago    5.54MB
asammour-test-cisco-nso-5.7.1-asammour    latest           a9f14f77391e   2 months ago    657MB
<none>                                    <none>           f03c17d5677f   2 months ago    1.4GB
cisco-nso-dev                             5.7.1-asammour   5cc13928e02b   2 months ago    1.4GB
cisco-nso-base                            5.7.1-asammour   fd302a3b7f71   2 months ago    657MB
<none>                                    <none>           0c22d6d2547b   2 months ago    657MB
<none>                                    <none>           72da0964cff4   2 months ago    1.4GB
ubuntu                                    latest           df5de72bdb3b   2 months ago    77.8MB
cisco-nso-base                            5.7.1            e4ae5cc98a1e   2 months ago    657MB
cisco-nso-dev                             5.7.1            5fc2dde6f0ea   2 months ago    1.4GB
centos                                    latest           5d0da3dc9764   13 months ago   231MB
asammour@boip:~$
```

Abbildung 6.2: installierte Docker-Images

In Abbildung 6.3 werden die erstellten Container für die Testumgebung dargestellt, die test2 benannt ist.

```
asammour@boip:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED       STATUS       PORTS     NAMES
71c3ccf4d5b4   5cc13928e02b   "/enter-shell.sh"        4 weeks ago   Up 4 weeks             test2
2288cd41d61b   nginx          "/docker-entrypoint.…"   4 weeks ago   Up 4 weeks   80/tcp    thor3
16deeb9dd221   busybox        "sh"                     4 weeks ago   Up 4 weeks             thor2
fcc4efdcbcf2   busybox        "sh"                     4 weeks ago   Up 4 weeks             thor
asammour@boip:~$
```

Abbildung 6.3: erstellte Docker Containers

Nun wird der Container test2 mit dem Befehl bash als OS gestartet. Danach wird zu NCS-Ordner navigiert, um NCS zum Laufen zu bringen (siehe Abbildung 6.4).

```
asammour@boip:~$ docker exec -it test2 bash
root@71c3ccf4d5b4:/# ls
 README.signature               dev             lib64   nso-5.7.1.linux.x86_64.installer.bin            root         sys        '~nso-5.7.1'
 bin                            enter-shell.sh  media   nso-5.7.1.linux.x86_64.installer.bin.signature   run          tailf.cer
 boot                           etc             mnt     nso-5.7.1.linux.x86_64.signed.bin                run-nso.sh   tmp
 cisco_x509_verify_release.py   home            nid     opt                                              sbin         usr
 cisco_x509_verify_release.py3  lib             nso     proc                                             srv          var
root@71c3ccf4d5b4:/# cd '~nso-5.7.1'
root@71c3ccf4d5b4:/~nso-5.7.1# ls
CHANGES  README   bin  erlang  examples.ncs  java  man    ncsrc.tcsh  packages  src      var
LICENSE  VERSION  doc  etc     include       lib   ncsrc  netsim      scripts   support
root@71c3ccf4d5b4:/~nso-5.7.1# cd src
root@71c3ccf4d5b4:/~nso-5.7.1/src# ls
ncs
root@71c3ccf4d5b4:/~nso-5.7.1/src# cd ncs
root@71c3ccf4d5b4:/~nso-5.7.1/src/ncs# ls
aaa    builtin_yang            cli                emacs   ncs_config  netsim            project-skeletons  snmp   yang
build  builtin_yang_cli_generic  configuration_policy  errors  netconf     package-skeletons  pyapi              tools
root@71c3ccf4d5b4:/~nso-5.7.1/src/ncs# ncs
Bad configuration: /opt/ncs/ncs-5.7.1/etc/ncs/ncs.conf:0: "./state/packages-in-use: Failed to create symlink: no such file or directory"
Daemon died status=21
root@71c3ccf4d5b4:/~nso-5.7.1/src/ncs#
```

Abbildung 6.4: NCS-Ordner

In Abbildung 6.5 wird der Status von NCS zur Kontrolle abgefragt.

```
root@71c3ccf4d5b4:/~nso-5.7.1/src/ncs# ncs --status | grep status
status: started
        db=running id=33 priority=1 path=/ncs:devices/device/live-status-protocol/device-type
root@71c3ccf4d5b4:/~nso-5.7.1/src/ncs#
```

Abbildung 6.5: erfolgreiche Verbindung zum NCS

Nachdem NCS gestartet wurde, kann die Testphase der Umgebung begonnen werden. Nun kann ein simuliertes Netzwerk mit dem Befehl `ncs-netsim --create-network` erstellt werden. Mit dem Befehl `cisco-ios-cli-3.8 3 ios` werden drei virtuelle Router von Type ios definiert (siehe Abbildung 6.6).

```
root@71c3ccf4d5b4:/~nso-5.7.1/packages/neds/nso-instance# ncs-netsim create-network cisco-ios-cli-3.8 3 ios
DEVICE ios0 CREATED
DEVICE ios1 CREATED
DEVICE ios2 CREATED
```

Abbildung 6.6:Virtuelle Router werden erstellt

Nun kann, wie in Abbildung 6.7 zu sehen ist, das Netz gestartet werden.

```
root@71c3ccf4d5b4:/~nso-5.7.1/packages/neds/nso-instance# ncs-netsim start
DEVICE ios0 OK STARTED
DEVICE ios1 OK STARTED
DEVICE ios2 OK STARTED
```

Abbildung 6.7: Virtuelle Router werden gestartet

Dabei muss beachtet werden, dass sowohl die NSO-Software (ncs) als auch die simulierten Netzwerkgeräte auf dem lokalen Server laufen.

Jetzt können die CLI auf einem der simulierten Geräte ausgeführt werden.

```
root@71c3ccf4d5b4:/~nso-5.7.1/packages/neds/nso-instance# ncs-netsim cli-i ios0

admin connected from 127.0.0.1 using console on 71c3ccf4d5b4
```

Abbildung 6.8: CLI-Zugriff auf einem Virtuellen Router

Abbildung 6.9 stellt die Verbindung zwischen NSO und der drei virtuellen Router dar. Außerdem sind die Cisco CLI ebenso zu erkennen.

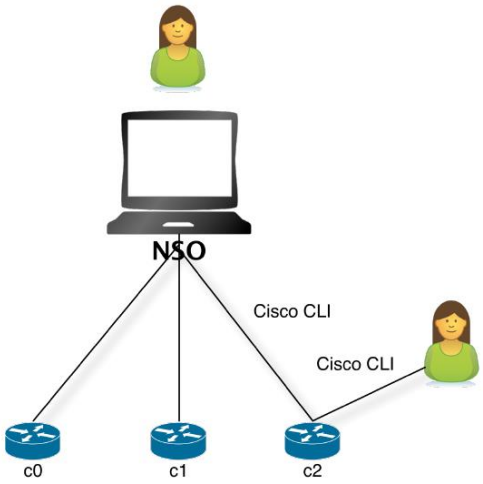

Abbildung 6.9: Netzwerk Simulation

Mit `enable` kann der CLI gestartet werden. Die Konfigurationen sind in Abbildung 6.10 dargestellt.

```
ios0> enable
ios0# show running-config
tailfned device netsim
tailfned police cirmode
no service password-encryption
no cable admission-control preempt priority-voice
no cable qos permission create
no cable qos permission update
no cable qos permission modems
ip source-route
no ip cef
no ip forward-protocol nd
no ip http server
no ip http secure-server
ip vrf my-forward
 bgp next-hop Loopback1
!
ip community-list 1 permit
ip community-list 2 deny
ip community-list standard s permit
no ipv6 source-route
no ipv6 cef
class-map match-all a
!
class-map match-all cmap1
 match mpls experimental topmost 1
 match packet length max 255
 match packet length min 2
 match qos-group 1
!
policy-map a
!
policy-map map1
 class c1
  drop
  estimate bandwidth delay-one-in 500 milliseconds 100
  priority percent 33
 !
!
interface Loopback0
 no shutdown
exit
interface FastEthernet0/0
 no shutdown
```

Abbildung 6.10: Virtuelle Router Konfigurationen

Nun ist die Topologie fertigerstellt und kann für die Durchführung von Testszenarien beginnen.

7 Fazit und Ausblick

SDN ist eine hervorragende technologische Entwicklung im Vergleich zum normalen Netzwerk, da sie softwarebasiert ist. Sie bietet eine breite Palette von Funktionen an und kann sowohl die Sicherheit verbessern als auch die Einrichtung von Testumgebungen vereinfachen. In dieser Arbeit wurde SDN für die Erstellung einer Testumgebung angewendet, mit dem Ziel verschiedene Testszenarien durchzuführen. Allerdings wurden nicht alle Bestandteile der SDN-Netze berücksichtigt.

Die erfolgreiche Erstellung der Testumgebung hat gezeigt, dass die Umstellung auf SDN realistisch und sinnvoll evaluiert werden kann. Weiterhin hat sich herausgestellt, dass die Erstellung von Topologien in NSO-Docker ebenso möglich ist. Somit können Softwareunternehmen anhand des Entwurfes solcher Testumgebungen profitieren. Diese Umgebungen können funktional den Bedingungen der realen Welt entsprechen und dennoch separat betrieben werden.

Das Projekt kann so erweitert werden, dass die Topologie auch auf weitere SDN-Aspekte angewendet werden kann, mit dem Hintergrund ein komplettes simuliertes Netzwerk zu realisieren. Die Umstellung erfolgt Schritt für Schritt im Live-Netz, bis sie vollständig automatisiert ist.

Literaturverzeichnis

[CISCO-a] https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/crosswork-infrastructure/4-1/AdminGuide/b_CiscoCrossworkAdminGuide_4_1/m-crosswork-data-gateway.html

[CISCO-b] Cisco Network Services Orchestrator (NSO) https://www.cisco.com/c/en/us/products/cloud-systems-management/network-services-orchestrator/index.html#~features (28.09.2022)

[Far06] https://www.ietf.org/rfc/rfc4655.txt

[Cap22] https://www.capterra.com.de/software/1030249/cisco-network-services-orchestrator-nso

[WW22] A Practical Guide to Docker Containers and VMs https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms (28.09.2022)

[Ubuntu-a] Ubuntu Requirements. https://ubuntu.com/tutorials/install-ubuntu-server#2-requirements(28.09.2022)

[Ubuntu-b] Ubuntu Installation guide. https://ubuntu.com/tutorials/install-ubuntu-server# (28.09.2022)

[Docker] Docker NSO https://github.com/NSO-developer/nso-docker (29.09.2022)

Versicherung

„Ich versichere, dass ich die vorstehende Arbeit selbständig angefertigt und mich fremder Hilfe nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichtem oder nicht veröffentlichtem Schrifttum entnommen sind, habe ich als solche kenntlich gemacht.“

_