HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

# Lab Topology Builder mit ACI

# Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2018


| | |
|---|---|
| Autor(en): | Yannick Zwicker \| Pirmin Wenk |
| Betreuer: | Urs Baumann |
| Projektpartner: | INS |
| Examinator: | Prof. Laurent Metzger |

**Abstract**

Ziel der Arbeit ist es, hybride Lab-Netzwerke, bestehend aus virtuellen und physischen Netzwerkgeräten, automatisiert zu erstellen und mittels der Cisco Application Centric Infrastructure (ACI) dynamisch zu verbinden.

Als Architektur und Design Pattern wurde die Twelve-Factor-App Methodik angewandt, die Patterns und Methodiken für moderne SaaS (Software as a Service) Applikationen aufzeigt.
Aufgrund diesen Architektur Patterns entstand eine skalierende, erweiterbare und auf Docker Containern basierende Microservice Architektur.

Der Kernservice der Architektur, der Orchestration Service, basiert auf dem Python Netzwerkautomatisierungs-Framework Nornir. Die Business Logik ist in parallelisierbaren, wiederverwendbaren Tasks gekapselt, die in einem Deployment und Removement Workflow organisiert sind.

Über eine Vue.js SPA (Single Page Application) werden die Workflows des Orchestration Services angestossen, sowie die Lab Topology Templates verwaltet.
Die Lab-Netzwerke werden im YAML Format spezifiziert und als Lab Topology Templates über die SPA im Templatestore Service persistiert. Dieser basiert auf einer Flask API, der die Templates lokal speichert.

Die provisionierten Ressourcen werden vom Orchestration Service fortlaufend im Runninglabstore gespeichert. Dieser Service basiert ebenfalls auf einer Flask API, der die Informationen in einer relationalen Postgres Datenbank persistiert.
Als «Single Source of Truth» für die Informationsbeschaffung über die physikalische Infrastruktur, nutzt der Orchestration Service das OpenSource Tool NetBox.

Durch die Microservice Architektur basierend auf Docker Containern, können weitere Services hinzugefügt werden.

# Inhaltsverzeichnis

# 1 Aufgabenstellung

## Lab Topology Builder mit ACI

## 1 Motivation

Das INS (Institute for Networked Solutions) bietet verschiedene Lehrgänge im Netzwerkbereich an, welche physische und virtuelle Infrastrukturen benötigen. Im Moment besteht ein Tool für das Deployment von CCIE Labs auf einer statischen Topologie.
Mit dem Lab Topology Builder soll eine Lösung erarbeitet werden, mit dem für aller Lehrgänge generisch Lab verwaltet werden können. Virtuelle und physikalische Geräte sollen miteinander verbunden werden können. Der Kursinstruktor soll die Topologie zusammenstellen können. Studenten sollen in der Lage sein, Labs zu deployen und darauf zuzugreifen.

## 2 Ziel

Das Ziel der Arbeit ist eine erweiterbare Architektur für die generische Verwaltung von Device-Typen, Lab-Templates und Lab-Konfigurationen. Die physische Hardware sowie virtuelle Instanzen sollen dabei berücksichtigt werden.
Währenddessen werden die erfassten physischen Geräte auf Verfügbarkeit geprüft.
Am Schluss wird das Lab aus einem Template deployed, wobei die Config auf die Geräte gespielt wird und das Netzwerk funktionstüchtig aufgesetzt wird. Der Student soll über die nötigen Zugangsdaten erhalten, damit er sich auf den Geräten anmelden und arbeiten kann.

### 2.1 Kriterien

Soll:

- Erweiterbare Architektur für Lab-Netzwerk Deployments
- Konzept und Use Cases
- Verwendung der Cisco ACI zur Erstellung der Verbindungen zwischen virtuellen und physischen Geräten
- Lab Templates sollen durch den User bearbeitbar sein
- Lab Snapshots
- Automatische Gerätekonfiguration

Kann:

- vCable+ (virtuelle Verbindung mit Jitter, Packet-Drop und Delay)
- Security Konzept, Multitenancy
- Frontend
- Accounting

L. METZGER

# 2 Management Summary

## 2.1 Ausgangslage

Der Aufbau von Lab-Netzwerken für Praktika, Übungen und Kurse am Institute for Networked Solutions der HSR stellt sich als zeitintensive Aufgabe dar.
Bis anhin gibt es entweder statisch verkabelte Lab-Netzwerke, auf denen nur begrenzte Netzwerktopologien realisiert werden können, oder virtuelle, die mühsam von Hand provisioniert werden müssen.
Im Zeitalter der Automatisierung liegt es deshalb nahe, gewisse Schritte zu vereinfachen und zu automatisieren. Genau bei dieser Automatisierung setzt diese Arbeit an.

Ziel dieser Umsetzungsarbeit ist es, hybride Lab-Netzwerke, bestehend aus virtuellen und physischen Netzwerkgeräten, automatisiert zu erstellen, verwalten und löschen. Die dynamischen Netzwerkverbindungen zwischen den physischen und virtuellen Geräten werden mithilfe der Cisco Application Centric Infrastructure (ACI) Lösung umgesetzt.

Abbildung 1: Cisco ACI

## 2.2 Vorgehen, Technologien

Während der Arbeit wurde zuerst ein Konzept entwickelt, das aufzeigt, welche Komponenten für eine solche Automatisierungslösung nötig sind, mit welchen externen Systemen interagiert werden muss und schlussendlich, welche Technologien eingesetzt werden sollen.

In einem weiteren Schritt wurde ein Prototyp erstellt, welcher die Konzepte bestätigt. Daraufhin folgte die Umsetzung der Architektur-Analyse in die Praxis.
Es hat sich herausgestellt, dass es Sinn macht, die Komponenten in Microservices aufzuteilen, welche jeweils in eigenständigen Docker Containern arbeiten – getreu dem Moto «one process per container».
Für wichtige Architekturentscheidungen haben wir uns an der «Twelve-Factor App» [59] Methodik orientiert, die von Martin Fowler inspiriert ist.

Neben der Speicherung der Daten in relationalen Datenbanken, stehen vor allem die Hardware Netzwerkgeräte, Virtualisierungs- und Automatisierungslösung im Vordergrund. Für letztere wurde das Python Framework Nornir verwendet, welches die Aufgaben in sogenannten Tasks parallelisiert ausführen kann, die wiederum in ganze Workflows gegliedert sind.

## 2.3 Ergebnisse

Aufgrund der Architektur-Analyse entstanden fünf zentrale Microservices, die über einen Loadbalancer, der ebenfalls als Docker Container läuft, erreichbar sind.

### Frontend

Für das Erstellen und Verwalten von Lab-Netzwerken, wurde eine Single Page Application mit Vue.js entworfen. Das Frontend dient dazu, die Lab Topology Templates und Runniglabs zu administrieren.
Für die Provisionierung und Löschung der Lab-Netzwerke wird die Orchestration Komponente angesteuert.
Über die drei Stores werden alle für das Frontend benötigten Informationen bezogen.
Es läuft als eigenständiger stateless Service und kann dadurch ohne Probleme horizontal skaliert werden.

### Orchestration

Die Orchestration Komponente stellt über eine Web API Deployment und Removement Workflows zur Provisionierung und Löschung von Lab-Netzwerken zur Verfügung. Das Python Automation-Framework Nornir stellt die Funktionalität für parallelisierbare Workflows zur Verfügung.

### Template Store

Die Lab-Netzwerke werden in einer für Mensch und Maschine einfach lesbaren Daten Serialisierungs-Sprache YAML Ain't Markup Language (YAML) spezifiziert. Diese Lab-Netzwerk-Spezifikationen werden als sogenannte Lab Topology Templates in den Template Store importiert und persistiert.

### Running Lab Store

Wird ein neues Lab aufgrund eines Lab Topology Templates deployt, werden während des Workflows von der Orchestration Komponente alle wichtigen Informationen zu einem Lab-Netzwerk im Running-Lab-Store gespeichert. Dadurch entsteht eine Zuordnung von Geräten im Lab Topology Template zu den effektiv provisionierten Geräten.

### Physical Infrastructure Store

Die physische Infrastruktur des Lab Equipments wird in einem speziell dafür geeigneten open-source Tool Netbox verwaltet. Neben des IP-Adressmanagements (IPAM) werden die ganzen physikalischen Netzwerk-, Strom- und Konsolen-Verkabelungen (Data-Center Infrastructure Management (DCIM)) verwaltet. Diese Informationen werden sowohl für die Provisionierung von der Orchestration Komponente, als auch für den Zugriff auf die Geräte vom Frontend benötigt.

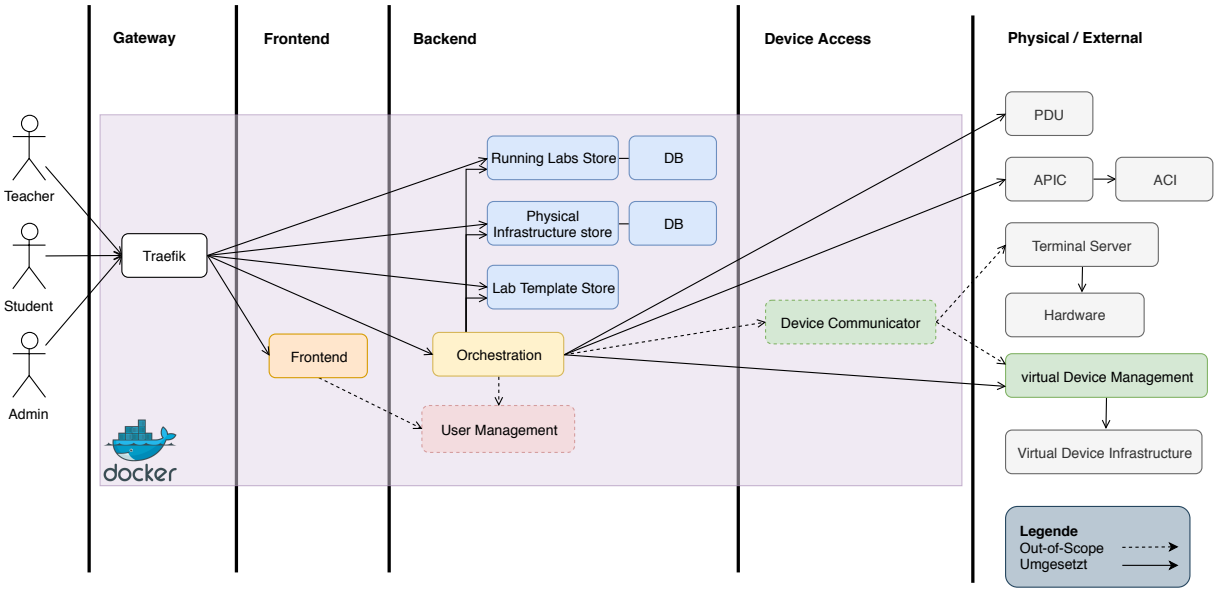



Abbildung 2: Architektur Übersicht

## 2.4 Ausblick

Das Potential für Erweiterungen dieser Software ist sehr gross, da aufgrund der Docker Architektur mit Microservices ohne grossen Aufwand einzelne Services ausgetauscht oder weitere hinzugefügt werden können.

In einem weiteren Schritt kann das User Interface so erweitert werden, dass das Lab Topology Template nicht mehr von Hand geschrieben werden muss, sondern per drag-and-drop ein Netzwerk zusammengestellt werden kann. Ebenfalls ist ein Buchungssystem angedacht, über welches physische Ressourcen in einem bestimmten Zeitraum reserviert werden können.
Neben der Erweiterung mit neuen Features muss ein Sicherheitskonzept erarbeitet werden, bei dem die einzelnen User beim Zugriff auf die Lab-Netzwerke authentisiert und autorisiert werden und dadurch nur auf ihre zugeteilten Labs Zugriff erhalten.

# 3 Ausgangslage

Bis anhin wurden die physischen Lab-Netzwerke am Institute for Networked Solutions (INS) immer von Hand auf- und wieder abgebaut. Eine Ausnahme stellen die CCIE Lab-Netzwerke dar, die über ein bestehendes Tool automatisiert ein- und ausgeschaltet werden können.
Pro User steht ein physikalischer Pod zur Verfügung, der in der Verkabelung und Anzahl Geräte nicht änderbar ist. Zudem sind die physikalischen Geräte mit einer festen Anzahl an virtuellen Geräten auf einem VMware ESXi Hypervisor verbunden. Aufgrund der aktuellen Architektur, müssen immer alle virtuellen Geräte laufen, das heisst sie werden nicht automatisch provisioniert.

Damit trotzdem eine Grosszahl von Lab-Netzwerken durchgeführt werden können, müssen tiefere Netzwerkkenntnise vorhanden sein, um z.B. über Sub-Interfaces die ganze physische Installation logisch zu erweitern. Dies ist vor allem für Netzwerk-Einsteigerkurse und die HSR Computernetze 1 und 2 Module des Studiengangs Informatik nicht praktikabel.

## 3.1 Constraints

Für die Arbeit sind folgende Technologien vorgegeben:

**Cisco ACI** Die Cisco ACI mit dem Cisco Application Policy Infrastructure Controller (APIC) sind am INS vorhanden und sollen für diese Arbeit verwendet werden

**Physische Netzwerkgeräte** Als Hardware stehen physische Cisco Router und Switches zur Verfügung, die an die ACI angeschlossen werden können

**Server** Als Server für die virtuellen Router stehen zwei Fujitsu Server zur Verfügung

Das Produkt der Arbeit soll sich schlussendlich sauber im INS-Netzwerk eingliedern.

# 4 Requirements

## 4.1 Use Cases

### 4.1.1 Diagramm

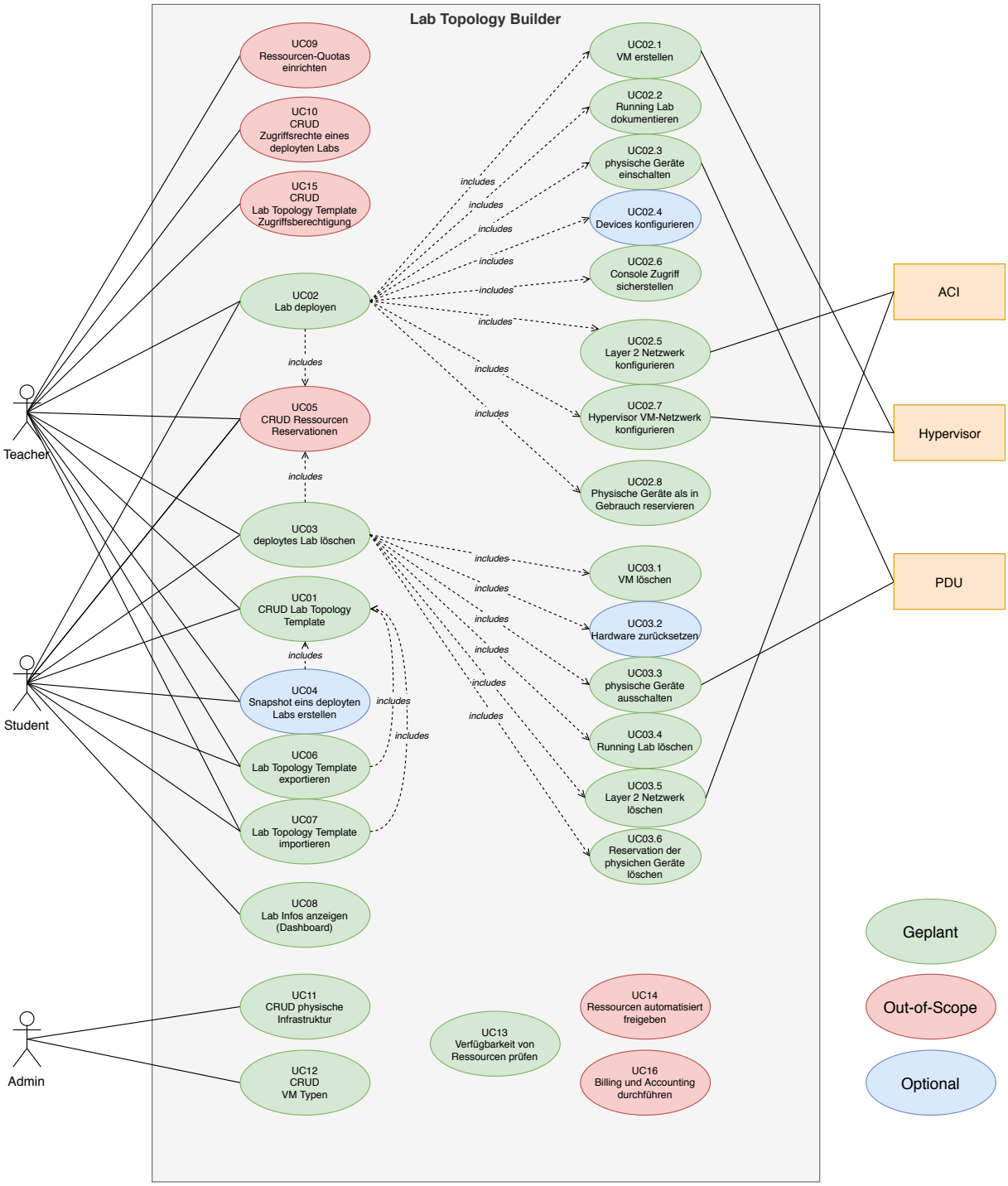



Abbildung 3: Use Case Diagramm

### 4.1.2 Aktoren

Es wird zwischen Teacher, Student und Admin als Aktoren unterschieden, die mit dem System interagieren:

**Teacher** Aktor, der über mehr Rechte verfügt als ein Student und einen Kurs, Übung oder Praktikum leitet.
**Student** Aktor, der an einem Kurs, Übung oder Praktikum teilnimmt.
**Admin** Aktor, der die physische Infrastruktur verwaltet, neue Gerätetypen und Templates erstellen kann.

### 4.1.3 Stakeholder

- INS
- Dozenten eines Kurses
- Studenten der HSR
- INS Kursteilnehmer
- Infrastruktur Administratoren
- Softwareentwickler

### 4.1.4 Beschreibung

#### UC01 - CRUD Lab Topology Template

Ein User (Teacher und Student) kann ein Lab Topology Template erstellen, anzeigen, bearbeiten und löschen. Ein Lab Topology Template beinhaltet unter anderem folgende Komponenten, die der User angeben kann:

**Description** Beschreibung des Lab-Netzwerkes
**Nodes** virtuelle Netzwerkgeräte, physische Netzwerkgeräte, virtuelle Maschinen
**Connections** Verbindungen der Geräte im Lab Topology Template
**Configurations** spezifische Konfiguration eines Nodes, z.B. Running-Config eines Routers

#### UC02 - Lab deployen

Ein User (Teacher und Student) kann ein zuvor erstelltes Lab Topology Template deployen. Dabei werden verschiedene Schritte automatisiert ausgeführt, wobei die Reihenfolge wegen den Abhängigkeiten beachtet werden muss:

- Verfügbarkeit der physischen Geräte prüfen und reservieren (siehe UC05 - Ressourcen reservieren, Seite 16)
- Physische Geräte einschalten
- Virtual machines (VMs) erstellen
- VMs inventarisieren
- Layer 2 Topologie (ACI) konfigurieren
- Hypervisor VM-Netzwerk konfigurieren
- Konsolenzugriff sicherstellen
- VMs und physische Geräte konfigurieren

Studenten können allerdings nur Labs deployen, welche von einem Teacher freigegeben wurden (siehe UC15 - Lab Topology Template für Student-Deployment freigeben, Seite 18).

### UC02.1 - VM erstellen

Durch das Deployen eines Labs wird aus dem Lab Topology Template ersichtlich, ob VMs deployt werden müssen.
In erster Linie sollen die virtuellen Router Cisco CSR 1000v [13] und Cisco IOS XRv [14] erstellt werden können. Zudem sollen Windows und Linux VMs erstellt werden können.
Die Erstellung erfolgt entweder direkt über den Hypervisor oder das Virtual Device Management. Es soll auf eine Ressourcen schonende und schnelle Lösung geachtet werden.

### UC02.2 - Running Lab dokumentieren

Ein laufendes Lab muss über eine eindeutige ID identifizierbar sein. Dadurch können alle damit verbundenen Ressourcen zugeordnet und bei Bedarf gelöscht werden.
Damit der User im Endeffekt über das Device im Lab Topology Template auf das effektive Device zugreifen kann, muss bekannt sein, welches Device des Lab Topology Templates welchem deployten Device entspricht.
Das Running Lab hat ein Ablaufdatum, damit der Garbage-Collector dieses automatisiert löschen kann.

### UC02.3 - physische Geräte einschalten

Die physischen Geräte werden einzeln über eine HTTP-API der Power Distribution Unit (PDU) eingeschaltet. Die Informationen welches Gerät über welche PDU eingeschaltet werden kann, muss zentral abrufbar sein (siehe UC11 - Hardware erfassen, Seite 17).

### UC02.4 - Devices konfigurieren

Im Lab Topology Template kann dem physischen, sowie dem virtuellen Gerät eine Konfiguration mitgegeben werden.
Die physischen Geräte sind per Konsolenkabel auf einen Terminalserver verbunden und werden über diesen automatisiert mit der Konfiguration beschrieben, sobald diese Einsatzbereit sind.

Es muss zentral gespeichert werden, welches Gerät über welche IP-Adresse und Port angesprochen werden kann (siehe UC11 - Hardware erfassen, Seite 17).
Die virtuellen Geräte werden über ein virtuelles, serielles Konsolenkabel ebenfalls über eine IP-Adresse und Port angesteuert und auf diese Weise mit der Konfiguration versehen. Über das Running Lab müssen dazu ebenfalls die benötigten Informationen abrufbar sein, um auf die entsprechende Konsole zugreifen zu können.

### UC02.5 - Layer 2 Netzwerk konfigurieren

Über die HTTP-API der ACI wird ein Layer 2 Netzwerk jeweils für die im Lab Topology Template erfassten Connections erstellt.
Jede Connection entspricht einer, auf dem Layer 2 des Open Systems Interconnection Modells (OSI Modells), getrennten Verbindung, die zwischen virtuellen Geräten oder auch in Kombination mit physischen Geräten erstellt wird.

### UC02.6 - Konsolen Zugriff für User sicherstellen

Für den User (Teacher und Student) müssen die Daten aufbereitet werden, wie auf die physischen und virtuellen Geräte zugegriffen werden kann (IP-Adresse, Port etc.).
Zudem müssen die entsprechenden Zugriffsberechtigungen konfiguriert werden.
Die Informationen um auf die Geräte zuzugreifen werden dem User dann angezeigt.

### UC02.7 - Hypervisor Netzwerk konfigurieren
Nachdem die VMs und das Layer 2 Netzwerk erstellt wurde, können die VMs den entsprechenden Netzen zugewiesen werden. Zudem muss auf den Hypervisor Servern die entsprechenden neuen Netze verfügbar gemacht werden.

### UC02.8 - Physische Geräte als in Gebrauch reservieren
Zu den physischen Geräten muss zentral ein Attribut hinterlegt werden können, welches beschreibt, ob das Gerät noch verfügbar ist oder bereits in einem Lab deployt wurde, um zu verhindern, dass ein Gerät für zwei Labs gleichzeitig provisioniert wird.

### UC03 - Deploytes Lab löschen
Wenn ein deploytes Lab wieder gestoppt werden soll, wird die Infrastruktur in folgender Weise abgebaut:

1. Physische Geräte zurücksetzen
2. VMs löschen
3. Virtuelle Netzwerke über ACI entfernen
4. Netzwerke von den Hypervisor Hosts entfernen
5. Hardware ausschalten
6. Reservation der physischen Geräte löschen
7. Running Lab entfernen

Ein Student hat nur die Berechtigung, seine eigenen Labs zu entfernen. Teacher hingegen können alle laufenden Labs löschen.

### UC03.1 - VM löschen
Die VMs müssen alle wieder heruntergefahren und gelöscht werden. Ebenfalls muss der Speicherplatz der VM wieder freigegeben werden.

### UC03.2 - Hardware zurücksetzen
Damit die Konfiguration auf den physischen Geräten zurückgesetzt wird, muss diese per serieller Verbindung gelöscht werden. Dabei müssen verschiedene Fehler beachtet werden, wie wenn jemand das Passwort geändert hat.

### UC03.3 - Physische Geräte ausschalten
Über die PDU werden die physischen Geräte wieder vom Strom getrennt, sobald sie zurückgesetzt wurden (siehe UC03.2 - Hardware zurücksetzen, Seite 15).

### UC03.4 - Running Lab löschen
Zum Schluss kann das Running Lab gelöscht werden, womit alle Einstellungen und Referenzen zu den zuvor gelöschten Ressourcen des deployten Labs entfernt werden.

### UC03.5 - Layer 2 Netzwerk löschen
Über den APIC werden die Lab-Netzwerkverbindungen wieder entfernt.

### UC03.6 - Reservation der physischen Geräte löschen
Die physischen Geräte müssen wieder dem Inventar als Verfügbar hinzugefügt werden, damit sie für neue Labs genutzt werden können.

### UC04 - Snapshot eines deployten Labs erstellen

Um einen gewissen Arbeitsstand zu sichern kann der User (Teacher und Student) eine laufende Lab-Umgebung in ein neues Lab Topology Template ablegen, welches er zu einem späteren Zeitpunkt wieder deployen kann.
Über das Running Lab kennt man die deployten Nodes. Es muss zwischen virtuellen Hosts (Linux/Windows) und physischen/virtuellen Netzwerkgeräten für den Snapshot unterschieden werden:

**physische/virtuelle Netzwerkgeräte** Zugriff via Device Communicator und auslesen der Konfiguration. Die Konfiguration wird dem entsprechenden Node im neuen Lab Topology Template zugeordnet.

**virtuelle Hosts** Snapshot oder klonen via virtual Device Manager. Im neuen Lab Topology Template muss auf den Snapshot oder Klon verwiesen werden.

Die Connections vom Lab Topology Template des deployten Running Labs werden in das Neue übernommen.

Ein Snapshot eines Labs wird als neues Lab Topology Template erstellt:

1. ID: wird eine neue vergeben
2. Netzwerkgeräte (physisch und virtuell): neue Configs werden übernommen
3. VMs: Verweis auf Snapshot oder Klon wird hinzugefügt
4. Connections werden übernommen

### UC05 - Ressourcen reservieren

Ein physisches Gerät muss frei verfügbar sein, bevor es für ein neues Lab reserviert werden kann. Unter Reservation wird hier das Gerät als «in Gebrauch» verstanden und es kann somit nicht mehr für ein weiteres Lab vergeben werden.

### UC06 - Lab Topology Template exportieren

Bei einem Export eines Lab Topology Templates werden alle benötigten Informationen in ein leserliches Format exportiert. Dieser Export kann dann mit einem Text-Editor angepasst werden. Der Export beinhaltet folgende Elemente:

- Description
- Physische Netzwerkgeräte (Node)
- Virtuelle Netzwerkgeräte (Node)
- VMs (Node)
- Verbindungen zwischen den Nodes (Connections)
- Configs der Nodes

### UC07 - Lab Topology Template importieren

Bei einem Import kann das angepasste Export-File wieder zu einem neuen Lab Topology Template importiert werden. Das Format muss dabei exakt dem Format des Exports entsprechen.
Zusätzlich muss beim Import noch eine Syntaxprüfung des Templates erfolgen.

### UC08 - Info Dashboard

Das Info Dashboard ist die graphische Schnittstelle zwischen Anwender und dem System. Es können verschiedene Interaktionen durchgeführt werden:

- Import / Export der Lab Topology Templates
- Anzeige der vorhandenen Lab Topology Templates
- Anzeige der vorhandenen Running Labs
- Details und Informationen eines Running Labs
- Starten eines Lab Topology Templates
- Stoppen eines Running Labs

### UC09 - Ressourcen Quotas

Ein Teacher kann Ressourcen Quotas für Studenten einrichten. Ressourcen Quotas basieren auf der Anzahl an physischen Devices und virtuellen Devices, die ein Student maximal reservieren bzw. erstellen darf.

### UC10 - Zugriffsrechte auf deployes Lab vergeben

Ein Teacher kann Zugriffsrechte eines deployten Labs verwalten. Damit kann weiteren Usern der Zugriff auf die provisionierten Geräte gegeben oder entzogen werden.

### UC11 - Hardware erfassen

Die Hardware muss in einem zentralen Backend erfasst werden können, von wo sie über eine Schnittstelle abrufbar ist. Folgende Informationen sind dabei für physische Geräte wichtig:

- Gerätebezeichnung
- Physical Device Type und Version
- Interface (gehört zu einem Physical Device)
- Connection (verbindet Physical Device Interfaces)
- Verfügbarkeit (in Gebrauch oder frei)
- Gerätelabel (Rolle des Gerätes)
- Zugehörige PDU (IP-Adresse) und Anschluss (Port)
- Zugehöriger Terminal Server (IP-Adresse) mit Anschluss (Port)

### UC12 - VM Typen erfassen

Die VM Typen müssen erfasst werden können, damit eine VM basierend auf einem Template automatisiert erstellt werden kann. Folgende Attribute sind dabei miteinzubeziehen:

- VM Typ Description
- VM Typ Version
- VM Typ Label
- Name des VM-Templates

### UC13 - Ressourcen prüfen

Die Hardware ist nur in begrenzter Anzahl verfügbar. Deshalb muss vor dem Deployment überprüft werden, ob genügend physische Geräte für das Lab frei sind. Die Überprüfung wird von der Orchestrierungs-Komponente (Lab Deployment) angestossen und sucht im Backend nach verfügbaren physischen Geräten.
Dabei wird anschliessend eine Reservation der physischen Ressourcen durchgeführt (siehe UC05 - Ressourcen reservieren, Seite 16).

Auch für die VMs gibt es nicht endlos Ressourcen auf den physischen Hosts. Die Virtualisierung muss diese Ressourcen unter Kontrolle haben können, damit die Orchestrierungskomponente diese auf verfügbare Ressourcen abfragen kann. Sind nicht genügend Ressourcen vorhanden, muss der Workflow ebenfalls abgebrochen werden.

### UC14 - Ressourcen automatisiert freigeben

Ein Lab besitzt ein Ablaufdatum. Wenn ein Lab dieses überschreitet, kann es von der Garbage Collector Komponente gelöscht werden, um Ressourcen zu sparen.

### UC15 - Lab Topology Template für Student-Deployment freigeben

Ein Teacher kann Deploy-Rechte für ein Lab Topology Template erstellen. Danach können die Studenten aufgrund des Lab Topology Templates ein Lab deployen.

### UC16 - Billing und Accounting durchführen

Für einen externen Benutzer, der ein Lab gemietet hat, werden die Kosten aufgrund des Verbrauchs berechnet. Es ist denkbar, dass die Berechnung auf verschiedenen Kriterien basiert:

- Anzahl an gebrauchten Netzwerkgeräten
- Dauer der Nutzung des Labs
- Benutzte Systemressourcen (CPU, RAM, Netzwerktraffic etc.)

## 4.2 Nichtfunktionale Anforderungen

### 4.2.1 Performance Efficiency

**NFR 1**

Die Tasks der Orchestrierung müssen parallel ablaufen können, um eine lange Wartezeit bei grösseren Labs zu vermeiden.

**NFR 2**

Ressourcen sind nur begrenzt vorhanden. Deshalb ist es wichtig, sparsam mit Festplattenspeicher und Arbeitsspeicher umzugehen und wo möglich Duplizierung sogar ganz zu vermeiden.

**NFR 3**

Das Ziel ist es, die Architektur so zu wählen, um 10 Labs mit jeweils bis zu 40 Devices deployen zu können. Das Deployment eines Lab Topology Template soll in einer angemessenen Zeit vonstattengehen.

- 40 Devices pro Lab
- 10 Labs gleichzeitig
- 50 Connections pro Lab
- max. 5 min Wartezeit pro Lab Deployment

### 4.2.2 Operability (Usability)

**NFR 4**

Die Architektur muss robust gegen Fehler in der Automatisierung und Kommunikation sein.
Das System muss im Falle eines Fehlers immer in einem «sauberen» Zustand hinterlassen werden (keine reservierten Ressourcen oder laufende Geräte hinterlassen).

### 4.2.3 Security

**NFR 5**

Der Lab Topology Builder verwendet HTTPS für die Kommunikation mit den Usern.

Die User müssen einzeln authentisiert und autorisiert werden, bevor sie den Lab Topology Builder benutzen können.

**NFR 6**

Der Zugriff auf die virtuellen und physischen Geräte muss pro deploytem Lab geschützt werden, damit ein User nicht auf Geräte eines anderen Users zugreifen kann.

### 4.2.4 Compatibility

**NFR 7**

Die Tools sollen bei Bedarf ausgetauscht werden können. Es ist wichtig, die Schnittstellen zwischen den Tools zu beschreiben und möglichst generisch zu halten. Durch eine Modularität der Komponenten sollen diese schnell und unproblematisch austauschbar sein.

### 4.2.5 Maintainability

**NFR 8**

Die Architektur muss leichtgewichtig sein, damit bei Updates oder Issues von verwendeten Komponenten schnell auf eine neue Version gewechselt werden kann. Der Administrationsaufwand bzw. der Unterhalt der Architektur muss möglichst klein sein.

### 4.2.6 Transferability

**NFR 9**

Die einzelnen Komponenten müssen über Umgebungsvariablen konfigurierbar, anpassbar und von einem zentralen Repository installierbar sein.

# 5 Risikoanalyse

## 5.1 Definition

**R1** Virtualization - Es dauert länger als geplant, um die Virtualisierungslösung zu evaluieren und aufzusetzen. Die Einarbeitung in die API macht Schwierigkeiten

**R2** Hardware - Vorhandene Hardware verhält sich nicht wie gedacht (Physische Server, PDU, ACI, Router, Terminal Server). Es muss Hardware ersetzt oder weggelassen und gemocked[19] werden.
*Konkrete Risiken*:

- **R2.1** Die ACI hat ein Problem, wenn zwei Bridge Domains die gleichen Subnetze zugewiesen haben. Als Alternative müsste pro Connection ein eigenes VRF erstellt werden. Jedoch kann es ebenfalls eine Beschränkung der Anzahl VRFs geben.
- **R2.2** Um 10 CCIE Labs mit je 50 Connections (inkl. vCable+ 100 Connections) betreiben zu können, werden 1000 verschiedene VLANs benötigt. Kann ein so grosser VLAN Pool reserviert werden und hat die ACI irgendwelche VLAN Limitationen?
- **R2.3** Wenn man ein statisches Endpoint Group (EPG) auf einem Leaf Interface deployt, muss ein VLAN für die «Port Encap» angegeben werden. Es würde einen grossen Aufwand bedeuten, wenn dieses für jeden physischen Port unterschiedlich sein muss, bzw. manuell verwaltet werden muss.
- **R2.4** Die zurzeit vorhandene ACI Hardware unterstützt keine richtigen Layer 2 Tunnels. Layer 2 Protokolle funktionieren nicht über die ACI.
- **R2.5** VMM Domain und die verwendete Software zur Administration der virtuellen Maschinen (Hypervisor Software) unterstützt nur VLANs und keine VXLANs bis auf den Hypervisor.
- **R2.6** Die PDU API unterstützt nicht alle Features, die wir benötigen.

**R3** Orchestration Tool Evaluation - Das zentrale Orchestrationtool macht bei der Evaluierung und beim Prototyping Schwierigkeiten, sodass es zu Verzögerungen kommen kann.

**R4** Projektmanagement - Jira Cloud unterstützt nicht alle Features oder hat einen Ausfall

**R5** Automatisierter Konsolen Zugriff - Der Zugriff via SSH und Telnet macht Probleme

### 5.1.1 Risikomatrix

Das Risiko ist folgend in zwei verschiedenen Versionen vorhanden, welche jeweils eine Analyse zu einem gewissen Meilenstein darstellen.
Die Eintrittswahrscheinlichkeiten sind so zu lesen:

- 0 - 20%: sehr tiefe Eintrittswahrscheinlichkeit (0 bis 1 erwartete Ereignisse)
- 20 - 40%: geringe Eintrittswahrscheinlichkeit (ca. 1 erwartetes Ereignis)
- 40 - 60%: mittelgrosse Eintrittswahrscheinlichkeit (2 bis 3 erwartete Ereignisse)
- 60 - 80%: grosse Eintrittswahrscheinlichkeit (4 bis 6 erwartete Ereignisse)
- 80 - 100%: sehr hohe Eintrittswahrscheinlichkeit (mehr als 6 erwartete Ereignisse)

Die Maximale Auswirkung des Ereignisses zeigt das höchstgeschätzte Schadenpotential in Stunden. Die Grösse der Blase zeigt den gewichteten Schaden im Verhältnis zu den anderen Risiken.

## 5.2 Version 1 - Meilenstein 1

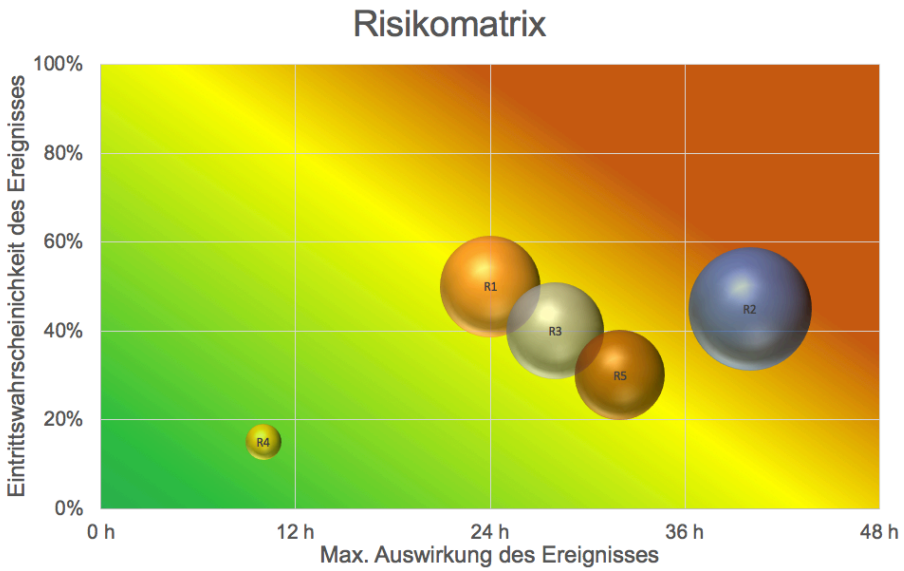



Abbildung 4: Risikomatrix Version 1

## 5.3 Version 2 - Meilenstein 4

Zum Zeitpunkt der End of Elaboration sind die Risiken möglichst beseitigt. Auffällig ist das Risiko R5, welches auch zum Schluss der End of Elaboration noch nicht beseitigt ist. Der Grund dafür liegt in der Kommunikation vom Device Communicator mit dem SSH-Proxy der Virtualisierungslösung.

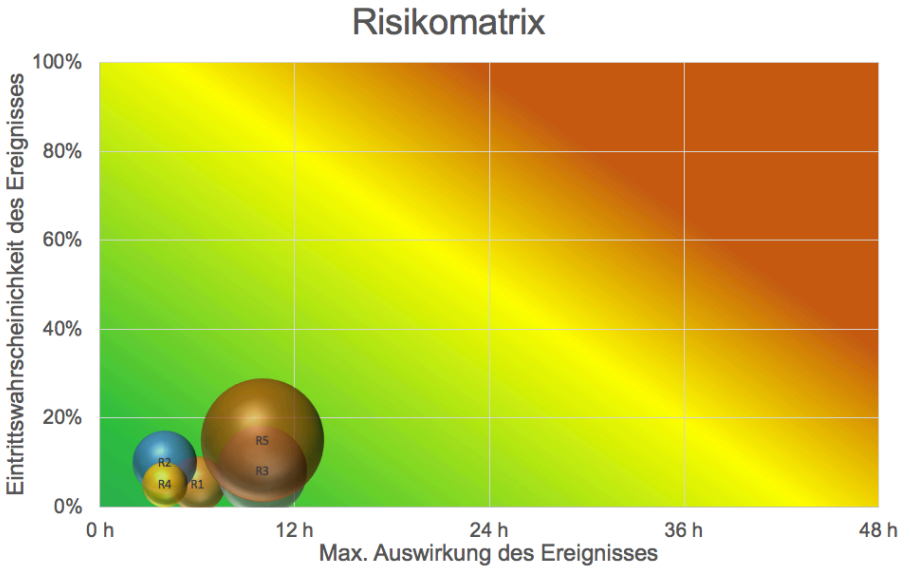



Abbildung 5: Risikomatrix Version 2

### 5.3.1 Abklärungen

**R1** Virtualization - Die Virtualisierungslösung mit der Administrationssoftware (oVirt) konnte nach anfänglichen Schwierigkeiten für den Prototyp gut aufgesetzt werden. Die API Tests verhalten sich wie in der Dokumentation beschrieben. Das Erstellen von VMs über die API konnte bereits erfolgreich getestet werden.

**R2** Hardware:

*Konkrete Risiken*:

**R2.1** Eine Bridge Domain (BD) pro Connection funktioniert in unserem Fall erfolgreich, da sie einem VLAN entspricht und deshalb für uns eine eigene Broadcast Domain ist. Es wird kein VRF benötigt, da wir der BD auf der ACI kein Subnetz zuweisen. Getestet wurde es in einem Setup mit 4 virtuellen Routern vom Typ CSR1000v, die sich jeweils als Paar in einer Bridge Domain befunden haben. Pings konnten ohne Probleme versendet werden. Ebenfalls funktionierte das OSPF Routing Protokoll ohne Probleme.
Pro Leaf können maximal 400 VRFs erstellt werden. Die Anzahl BDs pro Leaf ist auf 3500 beschränkt (siehe Abbildung 6, Seite 24). Dies genügt dem NFR 3 (siehe NFR 3, Seite 19).

**R2.2** Die Leafs haben ein VLAN Maximum von 3960 (siehe Abbildung 6, Seite 24).

**R2.3** Es kann jedes Mal das gleiche VLAN genutzt werden. Dieses VLAN hat keinen Einfluss auf die Funktionsweise. Dies würde nur benötigt, wenn beim ACI Leaf der Port als Trunk konfiguriert wäre, als *physical Network extension*.

**R2.4** Die zurzeit vorhandene ACI Hardware unterstützt keine richtigen Layer 2 Tunnels. Layer 2 Protokolle (CDP, LLDP) funktionieren nicht über die ACI. Auf die Studienarbeit hat dies keinen Einfluss.
Es werden neue ACI Leafs bestellt und sobald diese geliefert werden, kann das Tool erweitert werden. Es muss zusätzlich noch ein Q-in-Q Tunnel erstellt werden.
Je nach Liefertermin ist dies Out of Scope.

**R2.5** Grundsätzlich unterstützt die ACI nur VLANs in Kombination mit einer Virtual Machine Manager (VMM) Domain. Es gibt jedoch die Möglichkeit eine Cisco spezifische virtuelle Bridge zu verwenden auf dem Hypervisor, dann könnte man die VXLANs bis in den Hypervisor ziehen. Unbekannt ist, ob dies oVirt unterstützen würde.
Die VXLANs werden für die Studienarbeit nicht benötigt, da die ACI genügend VLANs unterstützt.

**R2.6** Die HTTP API der PDU wurde mit dem Postman getestet und entspricht vollumfänglich der Dokumentation. Es muss deshalb nicht auf eine andere Technologie oder sonstige zusätzliche Logik für aktuelle Zustandserkennung der jeweiligen Ports zurückgegriffen werden.

**R3** Orchestration Tool Evaluation - Das Prototyping verlief erfolgreich. Unbekannt ist zurzeit die Anbindung an ein externes Inventar (z.B. Netbox). Gemäss Source Code [3] sollte dies jedoch funktionieren bzw. wäre mit relativ geringem Aufwand selbst von Hand zu implementieren [4].

**R4** Projektmanagement - Jira Cloud unterstützt alle benötigten Features für ein erfolgreiches Projektmanagement

**R5** Automatisierter Konsolen Zugriff - Der Zugriff via SSH über den Proxy der virtual Device Management Software bereitet noch etwas an Schwierigkeiten. Grundsätzlich wurde eine Lösung gefunden. Diese verwendet Netmiko[8] als Library. Der Connection-Aufbau muss von Hand gesteuert werden bzw. nach dem Verbinden auf den SSH Proxy der oVirt Engine, wird ein *redispatch* durchgeführt, der vom Devicetyp Linux auf Cisco IOS umstellt (siehe Device Communicator, Seite 70). Den weiteren Projektphasen stellt dies jedoch kein grosses Hindernis mehr dar.

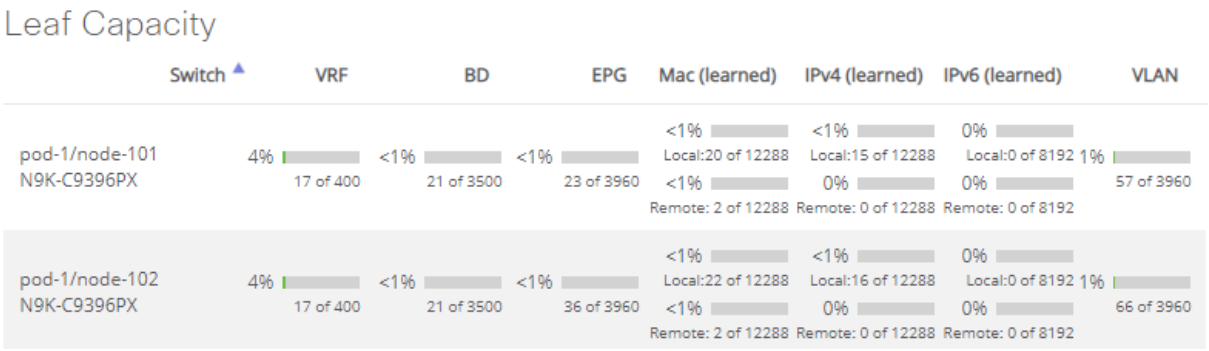



Abbildung 6: INS ACI Leaf Capacity per Kategorie

# 6 Software Architektur und Design

## 6.1 Solution Strategy

Es ist essentiell, welche Technologieentscheidungen wir für unsere Architektur treffen, deshalb wird im Folgenden auf verschiedene Patterns und bekannte Funktionsweisen zurückgegriffen.

### 6.1.1 Technology Decisions

#### Docker

Wir haben uns für eine auf Docker Container basierte Architektur aufgrund der folgenden Punkte entschieden:

- Die *Twelve-Factor-App* Architectural/Design Patterns (siehe Architectural/Design Patterns, Seite 25) können mittels einer Docker Microservices Architektur erfolgreich umgesetzt werden.
- Durch die modulare Microservice Architektur mit Docker Containern kann das NFR 7 (siehe Compatibility, Seite 19) erreicht werden.
- Die Architektur ist einfach und sehr leichtgewichtig. Updates von einzelnen Services kann über die Docker Container unkompliziert vonstattengehen, da sie *self-contained* sind. Damit wird NFR 8 (siehe Maintainability, Seite 20) erreicht.
- Durch den Einsatz einer Docker Registry, die die gebuildeten Software Services als Docker Images speichert, können die einzelnen Komponenten von dieser zentralen Registry installiert werden, ohne weitere Dependencies installieren zu müssen. Durch Umgebungsvariablen sind die Services konfigurierbar. Damit wird NFR 9 (siehe Transferability, Seite 20) erreicht.
- Weitere zentrale Faktoren, die zur Entscheidung beigetragen haben, sind in den Architectural/Design Patterns (Seite 25) festgehalten.

#### Python

Folgende Punkte waren ausschlaggebend für den Einsatz von Python.

- Know-how für die Weiterentwicklung ist am INS vorhanden
- Viele bekannte Libraries im Bereich Netzwerk basieren auf Python (Napalm, Netmiko, Paramiko, Nornir) [39]
- Leichtgewichtige Architektur ist möglich

#### Vue.js

Für Vue.js [29] haben wir uns aufgrund der folgenden Punkte entschieden.

- Sehr gut geeignet für Prototypen, da man in kurzer Zeit schnell etwas erreicht
- Das Team hat bereits aufgrund von mehreren Arbeiten Erfahrung damit
- Für einen Prototypen kann auf komplexe Features verzichtet werden (z.B. State Management Pattern Vuex [58])

### 6.1.2 Architectural/Design Patterns

Für die Architektur wird die *Twelve-Factor-App* [59] Methodik angewandt. Diese beschreibt im Wesentlichen 12 Faktoren, die beachtet werden sollen, wenn man eine moderne Software-as-a-Service Applikation entwickelt, die auch für Cloud Deployments vorbereitet ist.
Das erleichtert nicht nur das Deployment, sondern vor allem auch die Weiterentwicklung der Applikation.

### I. Codebase
*«One codebase tracked in revision control, many deploys»*

Für jeden Service wird ein eigenes Code Repository erstellt, da bei Änderungen des Codes eines Services, dieser durch das Continuous Integration / Continuous Delivery (CI/CD) automatisch getestet, gebuildet und deployt wird, unabhängig von den anderen Services.

Dadurch kann nachvollzogen werden, was wann geändert wurde und es muss auch nur dieser eine Service vom CI/CD neu gebuildet werden.

### II. Dependencies
*«Explicitly declare and isolate dependencies»*

Die Code Dependencies (Libraries etc.) werden bei Python in einer *requirements.txt* Datei, bei Vue.js in einer *packages.json* Datei gespeichert.

Weitere für die Services benötigten System-Libraries, werden im *Dockerfile* deklariert und beim Builden der Docker Container installiert.

### III. Config
*«Store config in the environment»*

Pro Service und Umgebung (Development/Production) gibt es ein einzelnes *Environment* File, das je nach dem für welche Umgebung die Software gebuildet wird, benutzt wird.

Dadurch werden globale Variablen im Code verhindert und alle benötigten Einstellungen, können von aussen über die Umgebungsvariablen gesteuert werden.

### IV. Backing services
*«Treat backing services as attached resources - The code for a twelve-factor app makes no distinction between local and third party services»*

Alle Ressourcen und Services können über Umgebungsvariablen konfiguriert werden. Dadurch kann eine Datenbank als Docker Service laufen, auf einem Datenbank Cluster betrieben oder aus der Cloud angebunden werden.

### V. Build, release, run
*«Strictly separate build and run stages»*

Im *Build Stage* wird die Applikation gebuildet und als Docker-Image auf die Docker-Registry gepushed.
Im *Run Stage* werden die Docker-Services durch ein *Docker-Compose* File instanziiert. Die gebuildeten Docker-Images werden von der Docker-Registry geladen und mit den Umgebungsvariablen gestartet.

### VI. Processes
*«Execute the app as one or more stateless processes»*

Durch die Verwendung von Docker als die darunterliegende Architektur, können die Services einfach horizontal skaliert werden. Gemäss der *Twelve-Factor-App* Beschreibung werden alle persistierten Daten in einen *Backing Service (z.B. Datenbank)* ausgelagert, damit der Service stateless bleibt.

### VII. Port binding
*«Export services via port binding»*

Die Docker-Container sind *self-contained* und können über das Port-Binding angesprochen werden (siehe Network, Seite 58).

### VIII. Concurrency
*«Scale out via the process model»*

Gemäss den Docker Best Practices soll jeder Container nur über einen Prozess verfügen - «one process per container», damit ist die *Separation of Concerns* sichergestellt. Wenn jeder Service nur über einen Prozess verfügt, kann dieser über die Docker-Architektur skaliert werden.

### IX. Disposability
*«Maximize robustness with fast startup and graceful shutdown»*

Durch die Docker-Architektur und die sehr leichtgewichtigen auf Python basierenden Services (siehe Services, Seite 62), haben sie eine kurze Startzeit und sind durch die stateless Architektur robust gegen Fehler.

### X. Dev/prod parity
*«Keep development, staging, and production as similar as possible»*

Damit die Development und Production Umgebungen möglichst ähnlich sind, kann über ein separates Docker-Compose File direkt in den Docker Containern entwickelt werden.
Das Dateisystem mit dem Code wird direkt als Volume in den Docker Container gemappt. Dadurch ist man unabhängig vom Betriebssystem und hat die gleiche Umgebung, wie in der Production.

Listing 1: Development Volumes - docker-compose.yaml

```
orchestration:
   build:
     context: ../orchestration
     dockerfile: Dockerfile
   volumes:
     - ../orchestration:/usr/src/app
   env_file: ./services/orchestration/orchestration.env
   networks:
     - internal_traefik
   labels:
     traefik.port: 5000
     traefik.enable: true
     traefik.backend: orchestration
     traefik.frontend.rule: Host:orchestration.ltb.ins
     traefik.docker.network: labbuilder_internal_traefik
```

### XI. Logs

*«Treat logs as event streams»*

Gemäss diesem *Twelve-Factor-App* Faktor soll die Applikation sich nicht um das Routing und Speichern der Logs kümmern müssen - «*A twelve-factor app never concerns itself with routing or storage of its output stream.*»

Die Logs der Services werden direkt in den *stdout / stderr* auf den Konsolen-Output gesendet. Dadurch können die Logs von aussen über die Docker-Architektur weiterverarbeitet werden und z.B. in ein Elasticsearch [7] eingeliefert werden, um die Logs zu analysieren.

### XII. Admin processes

*«Run admin/management tasks as one-off processes»*

Durch die Docker Container können Administrations-Prozesse als Startup-Scripts mitgegeben werden. Zum Beispiel können Datenbankmigrationen nach einem geänderten Model direkt in einem Startup-Script gemacht werden.
In einer produktiven Umgebung kann man ebenfalls durch SSH direkt auf die Docker Container zugreifen und gewisse Änderung dort durchführen. Dies sollte jedoch nur in Ausnahmefällen gemacht werden, bzw. ansonsten soll eher eine neue Version eines Services deployt werden.

## 6.2 Building Block View

### 6.2.1 Context View

Die Context View zeigt die Abhängigkeiten zwischen den Tools oder Usern, die den Lab Topology Builder ansteuern und jene, die von dem Lab Topology Builder angesprochen werden.

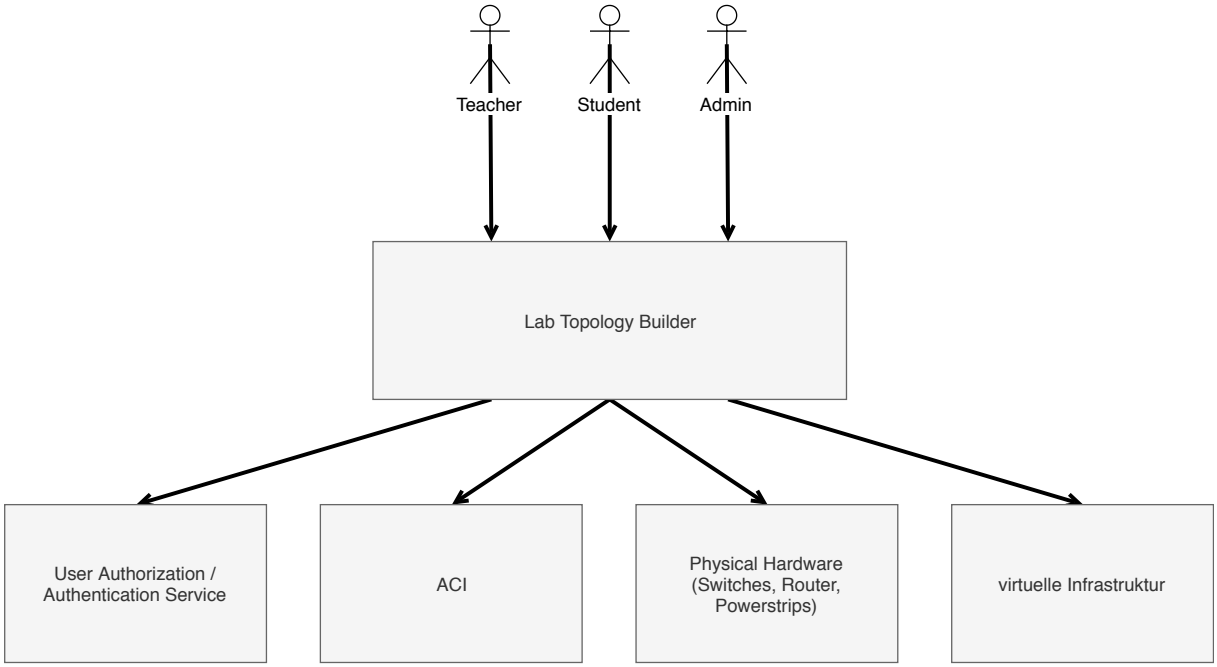



Abbildung 7: Context Diagram

### 6.2.2 Container View

Die Container View entspricht der Deployment View und zeigt, wie die Software physisch verteilt wird und wie die Container untereinander kommunizieren. Ein Container entspricht direkt auch einem Docker Container. Dadurch werden die Architekturpatterns eingehalten (siehe Architectural/Design Patterns, Seite 25).

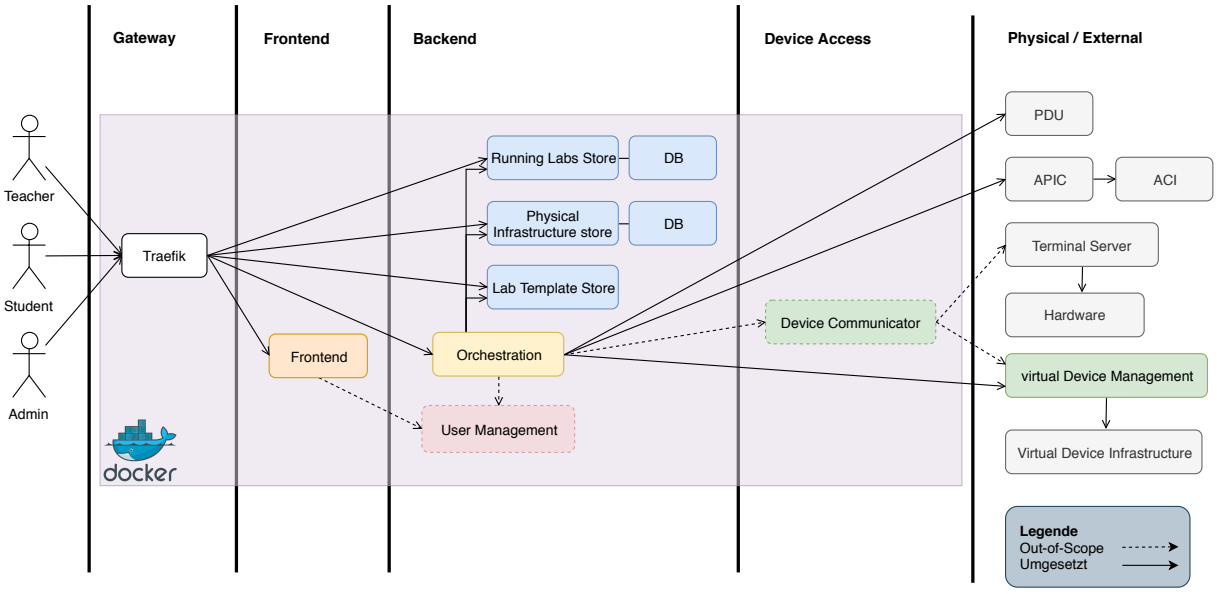



Abbildung 8: Container/Deployment Diagram

### 6.2.3 Component View

Durch die Component View sieht man in die einzelnen Container, wie sie aufgebaut sind und was für Komponenten enthalten sind.

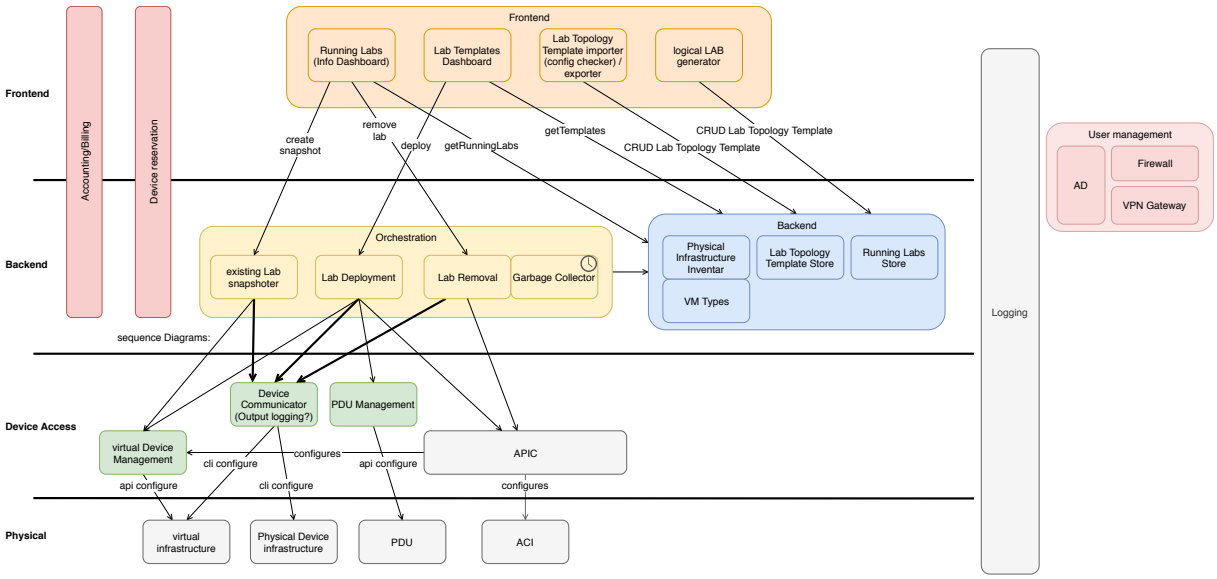



Abbildung 9: Component Diagram

Die CRC Card (Class-Responsibility-Collaboration Karte) beschreibt die jeweiligen Verantwortlichkeiten (Responsibilities) und die Abhängigkeiten (Collaborators) einer Komponente und zeigt mögliche Umsetzungsvarianten oder bereits bekannte Implementationen auf.

#### Backend

#### Backend - Lab Topology Template Store

| **Component: Lab Topology Template Store** | |
|---|---|
| **Responsibilities:**<br>- speichert Lab Topology Templates<br>- Template ID<br>- Description<br>- Nodes (HW Typ oder VM Typ, mit Verweis auf Snapshot / Klon, mit virtual Interfaces)<br>- Connections<br>- running Configs | **Collaborators:**<br>Interfaces from:<br>- UI<br>- Task Orchestration |
| **Candidate implementations technologies (and know uses):**<br>- Flask<br>- Django<br>- Persistence: File based \| NoSQL \| SQL | |

### Backend - Running Labs Store

| Component: Running Labs Store | |
|---|---|
| **Responsibilities:**<br>- speichert/liefert Informationen zu den running Labs<br>- Referenz zum Lab Topology Template<br>- Mapping zwischen Template und den explizit deployten devices (Physical Device, VM ID)<br>- Ablaufdatum des Labs | **Collaborators:**<br>Interfaces from:<br>- UI<br>- Task Orchestration |
| **Candidate implementations technologies (and know uses):**<br>- Flask<br>- Django<br>- Persistence: File based \| NoSQL \| SQL | |

Damit beim Abbau des Labs alle nötigen Informationen vorhanden sind, welche Geräte dafür im Einsatz sind, benötigen wir einen einfachen «Runninglab Store».
Weil während das Lab deployt wurde, das Lab Topology Template angepasst oder gelöscht werden könnte, darf man sich nicht darauf verlassen, dass im Template alle nötigen Informationen vorhanden sind.
Im Folgenden sind die Abhängigkeiten der Lab-Komponenten beschrieben, so wie sie benötigt werden, um ein laufendes Lab zu beschreiben. Für die Verwaltung und den Abbau des Labs stehen hier alle benötigten Informationen zur Verfügung.

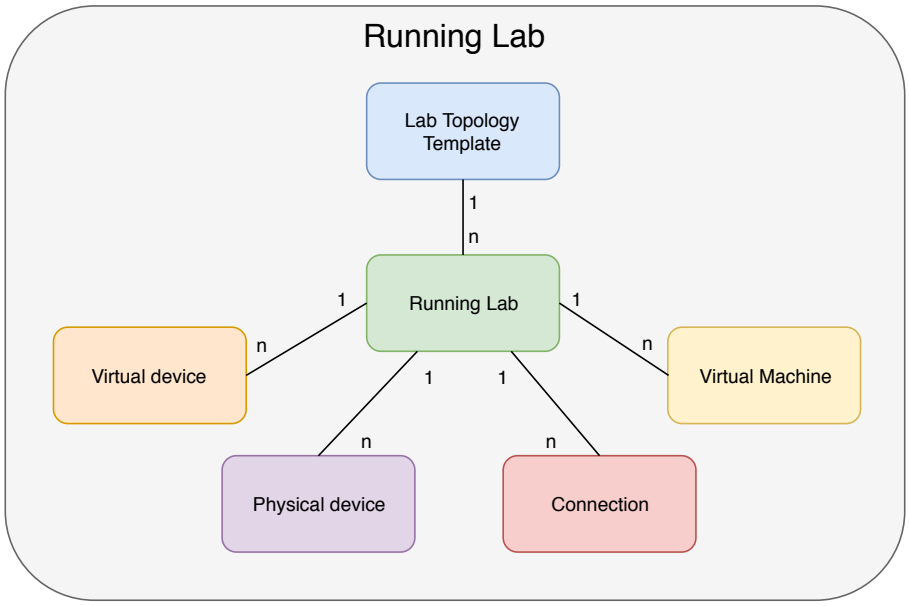



Abbildung 10: Running Lab Domainmodel

### Backend - Physical / Virtual Infrastructure Store

**Anforderungen**

- Physical Device darf nur einmal gleichzeitig vergeben werden können

Damit das physische Device nur an ein Lab vergeben wird, muss das Asynchronitäts-Problem gelöst werden. Es könnte sein, dass mehrere verschiedene Deployments gleichzeitig die Ressourcen anfragen und zufälligerweise dieselben freien physischen Geräte zugeteilt erhalten. Um das Problem zu lösen gibt es verschiedene Lösungsansätze:

1. Es gibt eine Zwischenschicht, die dafür sorgt, dass immer nur ein Deployment eine Anfrage starten kann. Die anderen werden in eine Queue gestellt.
2. Das Backend kümmert sich um den Status der Geräte. Es wird z.B. ein Zwischenzustand «Blockiert» auf dem Device gespeichert, sodass verhindert wird, dass ein zweites Deployment dieselben Geräte erhält, bevor das Erste die Devices definitiv «Reserviert» hat.
3. Das Orchestration-Tool speichert sich den Zustand der vergebenen physischen Devices in einem Singleton-Zwischenspeicher ab. Dies hat den Nachteil, dass die Orchestrierung nicht mehr stateless ist und deshalb nicht skaliert werden kann.
4. Die Orchestrierung fragt vom Backend die freien Devices ab und schreibt vor der definitiven Reservation die Device-ID in eine relationale Datenbank. Diese sorgt über einen Primärschlüssel für Uniqueness und Atomarität der Transaktionen und somit dafür, dass ein Gerät zur selben Zeit nur von einem Lab verwendet werden kann.

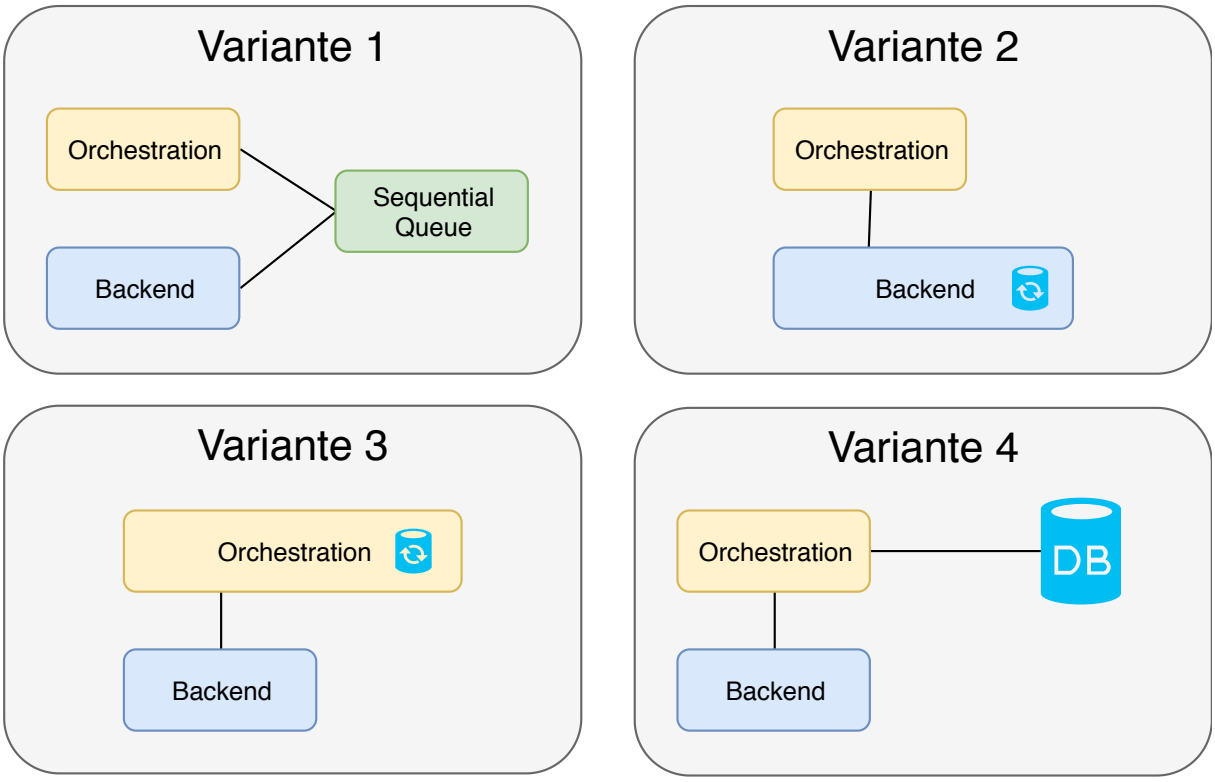



Abbildung 11: Arten der physischen Geräteverwaltung und Reservation

| Component: Physical / Virtual Infrastructure Store | |
|---|---|
| **Responsibilities:**<br>- speichert VM Templates (VM Typ)<br>- speichert HW Topologie:<br>- Gerätebezeichnung<br>- Physical Device (z.B. Router, Leaf etc.)<br>- Physical Device Type und Version<br>- Interface (zu einem Physical Device)<br>- Connection (verbindet Phys. Dev. Interf.)<br>- Verfügbarkeit (in Gebrauch oder frei)<br>- Gerätelabel (Rolle des Gerätes)<br>- Zugehörige PDU (IP, Port)<br>- Zugehöriger Terminal Server (IP, Port) | **Collaborators:**<br>Interfaces from:<br>- UI<br>- Task Orchestration |
| **Candidate implementations technologies (and know uses):**<br>- Netbox<br>- NSoT<br>- File based (YAML)<br>- eigene Implementation (Flask, Django etc.) | |

### Task Orchestration

**Anforderungen**

- Skalierbarkeit: Gleichzeitiges abarbeiten von min. 10 Workflows
- Parallelisierte Tasks im Workflow: Join, Fork (definierbare Anzahl)
- Möglichkeit zur Statusabfrage der Workflows
- Operability: Ease-of-use (einfacher Einstieg; gute Dokumentation),
- Maintainability: Modular (erweiterbar mittels Plugins), Wartbarkeit (wenige Abhängigkeiten zu anderen Tools/Frameworks)
- HTTP Web API ansteuern
- Workflows müssen ineinander verschachtelbar sein.

### Task Orchestration - Lab deployment

| Component: Task Orchestration - Lab deployment | |
|---|---|
| **Responsibilities:**<br>- verfügbare physische Geräte vom Backend abfragen<br>- physische Geräte reservieren (sequentiell über mehrere Workflows)<br>- Informationen zu running Lab in Backend abspeichern<br>- Lab deployen:<br>- physical / virtual Devices (klonen und / oder starten)<br>- APIC anstossen<br>- Devices Konfigurieren | **Collaborators:**<br>Interfaces from:<br>- UI<br>Interfaces to:<br>- Backend<br>- virtual Device Manager<br>- physical Device Power Management<br>- Device Communicator<br>- APIC |
| **Candidate implementations technologies (and know uses):**<br>- Ansible, StackStorm, OpenStack, eigene Implementation mit Flask und Nornir | |

### Task Orchestration - Lab removal

| Component: Task Orchestration - Lab removal | |
|---|---|
| **Responsibilities:**<br>- Running Config von physischen Geräten löschen<br>- Reservation zu Physischen Geräten freigeben<br>- Physische Geräte ausschalten<br>- VMs löschen<br>- virtuelle Netzwerkkonfiguration löschen | **Collaborators:**<br>Interfaces from:<br>- UI<br>- Garbage collector<br>Interfaces to:<br>- Backend<br>- virtual Device Manager<br>- physical Device Power Management<br>- APIC |
| **Candidate implementations technologies (and know uses):**<br>- Ansible, StackStorm, OpenStack, eigene Implementation mit Flask und Nornir | |

### Task Orchestration - Lab snapshoter

| Component: Task Orchestration - Lab snapshoter | |
|---|---|
| **Responsibilities:**<br>- Running lab snapshoten:<br>- Neues Lab Topology Template erstellen<br>- siehe UC04 - Snapshot eines deployten Labs erstellen, Seite 16 | **Collaborators:**<br>Interfaces from:<br>- UI<br>Interfaces to:<br>- Backend<br>- virtual Device Manager<br>- Device Communicator |
| **Candidate implementations technologies (and know uses):**<br>- Ansible, StackStorm, OpenStack, eigene Implementation mit Flask und Nornir | |

### Device Communicator

**Anforderungen**

- Muss via Terminal Server mit den physischen Devices kommunizieren können
- Muss via Proxy Server (virtual Device Manager) mit den virtuellen Devices kommunizieren können
- Einfache Erweiterbarkeit
- Error Handling muss konfigurierbar und implementierbar sein, damit die Nachvollziehbarkeit bei Fehlern gewährleistet ist

| Component: Device Communicator | |
|---|---|
| **Responsibilities:**<br>- Kommuniziert mit physischen und virtuellen Netzwerkgeräte per SSH und Telnet:<br>- Read/Write Config<br>- Reboot<br>- Reset Device (löschen startup-config, vlan.dat)<br>- Error handling mit Logging | **Collaborators:**<br>Interfaces from:<br>- Task Orchestration - Lab Snapshoter<br>- Task Orchestratoin - Lab Deployment<br>Interfaces to:<br>- virtual Devices:<br>- SSH Cisco CSR1000v<br>- SSH Cisco IOS-XRv<br>- vASA<br>- phyiscal Devices:<br>- Telnet/SSH Cisco IOS via Terminal Server |
| **Candidate implementations technologies (and know uses):**<br>- Ansible<br>- StackStorm (Napalm Plugin)<br>- OpenStack<br>- eigene Implementation mit Nornir / Napalm / Netmiko / Paramiko | |

### Physical Device Power Management (PDU)

| Component: Physical Device Power Management (PDU) | |
|---|---|
| **Responsibilities:**<br>- Power on / off von einzelnen phy-ischen Geräten mittels Powerstrip API | **Collaborators:**<br>Interfaces from:<br>- Orchestration |
| **Candidate implementations technologies (and know uses):**<br>- PDUs mit Web APIs: Value, Rittal, MyStrom, WTI, NetIO<br>- Eigene Implementation einer API, wenn PDU keine Web API hat | |

### Virtual Device Management

| Component: virtual Device Management | |
|---|---|
| **Responsibilities:**<br>- VMs klonen / snapshot<br>- VMs starten/stoppen<br>- VMs löschen<br>- VMs erstellen<br>- VMs inventarisieren<br>- vSwitches administrieren<br>- VM Inventar<br>- (Dynamic IP Configuration/Manage-ment Interface)<br>- (kann auf die GUI-Konsole der VMs zugreifen) | **Collaborators:**<br>Interfaces from:<br>- Orchestration<br>- vSwitches werden automatisiert durch die APIC administriert und konfiguriert |
| **Candidate implementations technologies (and know uses):**<br>- VMWare<br>- oVirt (Red Hat Virtualization)<br>- OpenNebula | |

### User Interface

| Component: User Interface | |
|---|---|
| **Responsibilities:**<br>- alle Lab Templates anzeigen<br>- Lab Template erstellen<br>- Lab Templates importieren / exportieren<br>- Lab Template deployen<br>- alle laufenden Labs anzeigen<br>- Running Lab Info Dashboard | **Collaborators:**<br>Interfaces to:<br>- Orchestration anstossen (deploy, delete, snapshot, import, export etc.)<br>- Backend abfragen |
| **Candidate implementations technologies (and know uses):**<br>- Eigene Implementation mit Vue.js | |

### Reservation

Die Reservation eines Labs im Voraus für einen bestimmten Zeitpunkt ist out of Scope.

### Accounting / Billing

Die Abrechnung der benutzten Ressourcen durch einen bestimmten User ist out of Scope.

### Authentication / Authorization

Aus Zeitgründen ist die Umsetzung der Autorisierung und Authentifizierung out of Scope.

| Component: Authentication / Authorization | |
|---|---|
| **Responsibilities:**<br>- Authentication<br>- Authorization<br>- Identity Management<br>- Role Management | **Collaborators:**<br>Interfaces from:<br>- UI<br>- Backend<br>- Orchestration<br>Interface to:<br>- Active Directory<br>- ADFS |
| **Candidate implementations technologies (and know uses):**<br>- Keycloak | |

## 6.3 Runtime View - Component Interaction

### 6.3.1 Abhängigkeiten

Beim Automatisieren des Deployments, Removements etc. ist es sehr wichtig, dass gewisse Tasks parallel ablaufen um Zeit zu sparen und andere Abläufe sequentiell vonstattengehen, wenn Abhängigkeiten unter den einzelnen Tasks auftreten. Diese Abhängigkeiten werden im Folgenden grafisch dargestellt.
Ein Task darf erst ausgeführt werden, wenn alle eintreffenden Pfeile zu einem abgeschlossenen Task zeigen.
Dabei stellt ein einzelner Kreis einen Task dar und ein roter Kreis einen Task, der parallelisiert auf verschiedenen Ressourcen abläuft.

#### Lab deployment

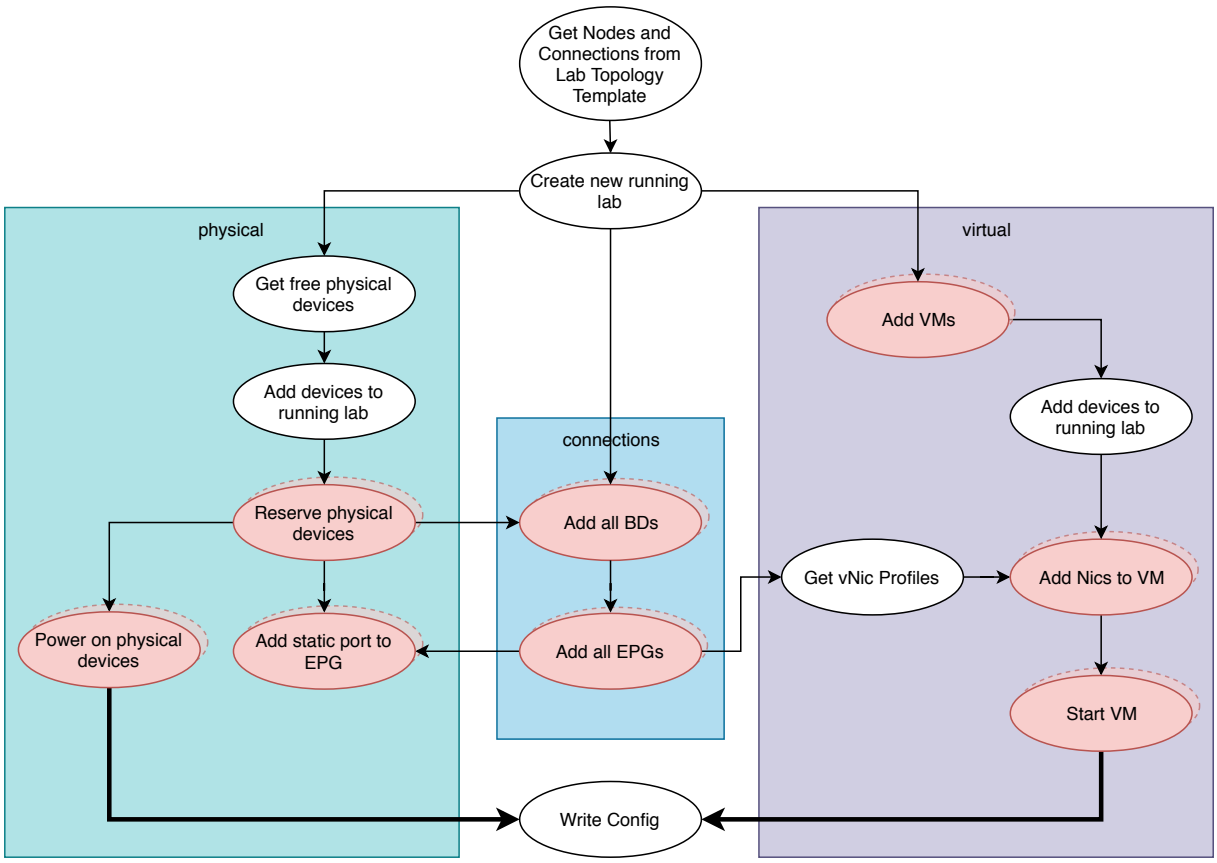



Abbildung 12: Abhängigkeiten beim Lab Deployment

### Lab removement

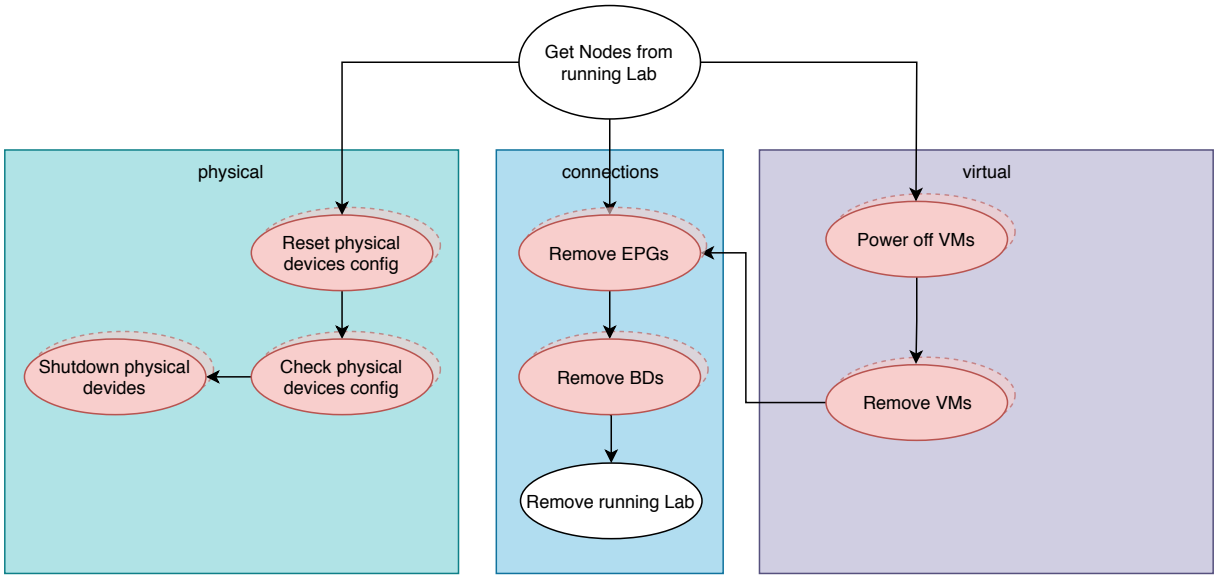



Abbildung 13: Abhängigkeiten beim Lab Removement

### 6.3.2 Sequenzdiagramme

Mit den folgenden Sequenzdiagrammen wird der Prozess der Automatisierung in der Orchestration schematisch dargestellt. Es wird aufgezeigt, wie und in welcher Reihenfolge die Komponenten miteinander interagieren sollen.

#### Lab deployment

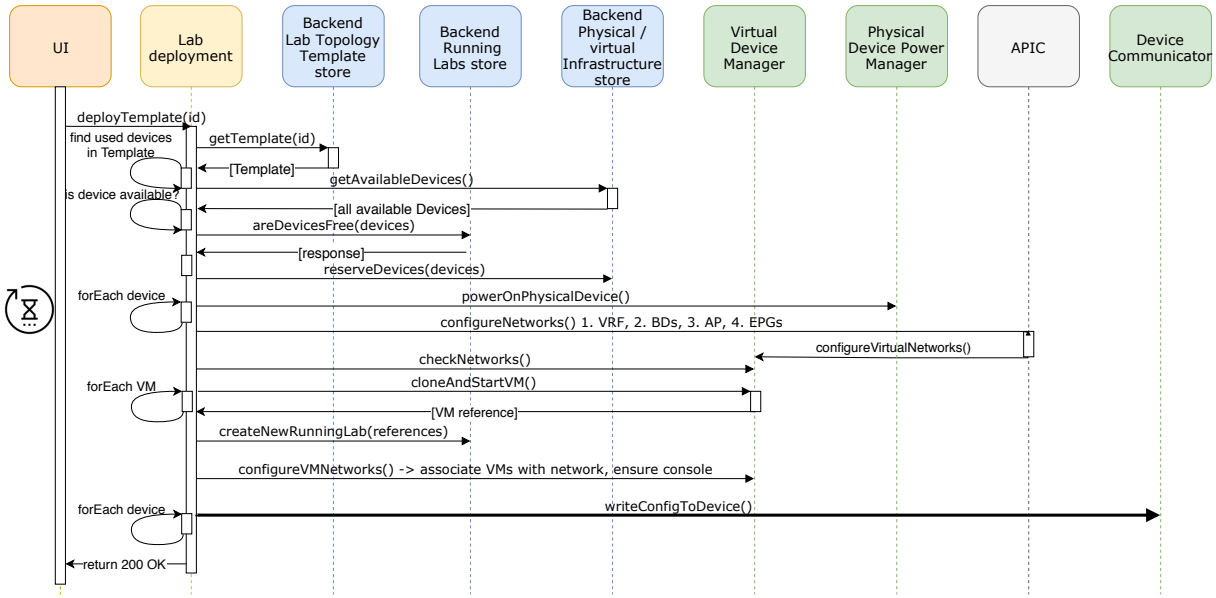



Abbildung 14: Sequenzdiagramm Lab Deployment

#### Lab removement

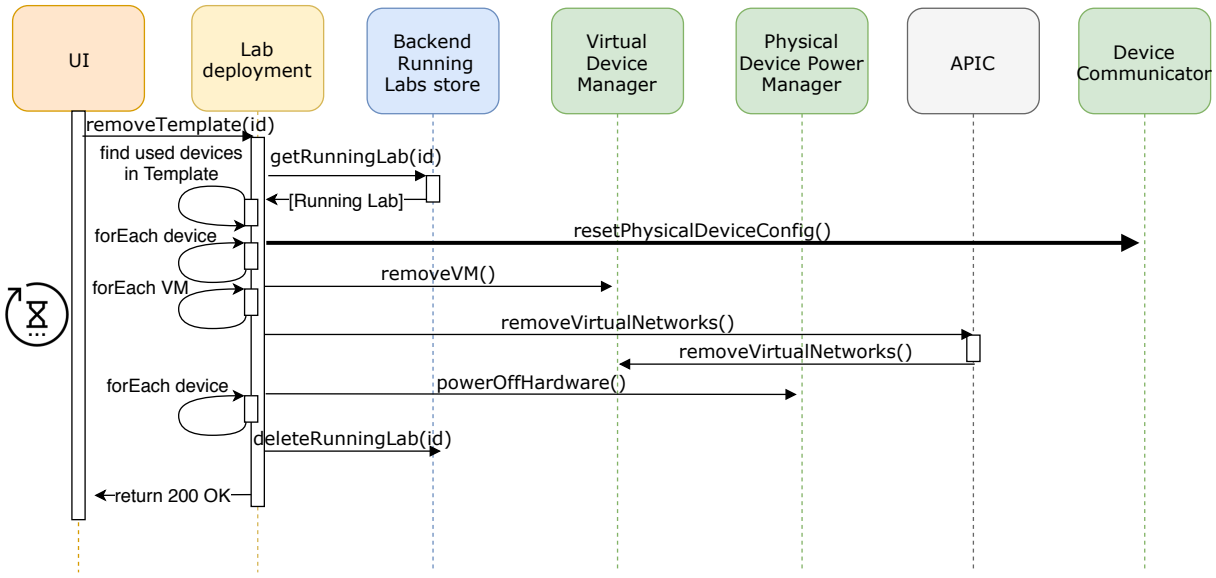



Abbildung 15: Sequenzdiagramm Lab Removement

### Lab snapshoter

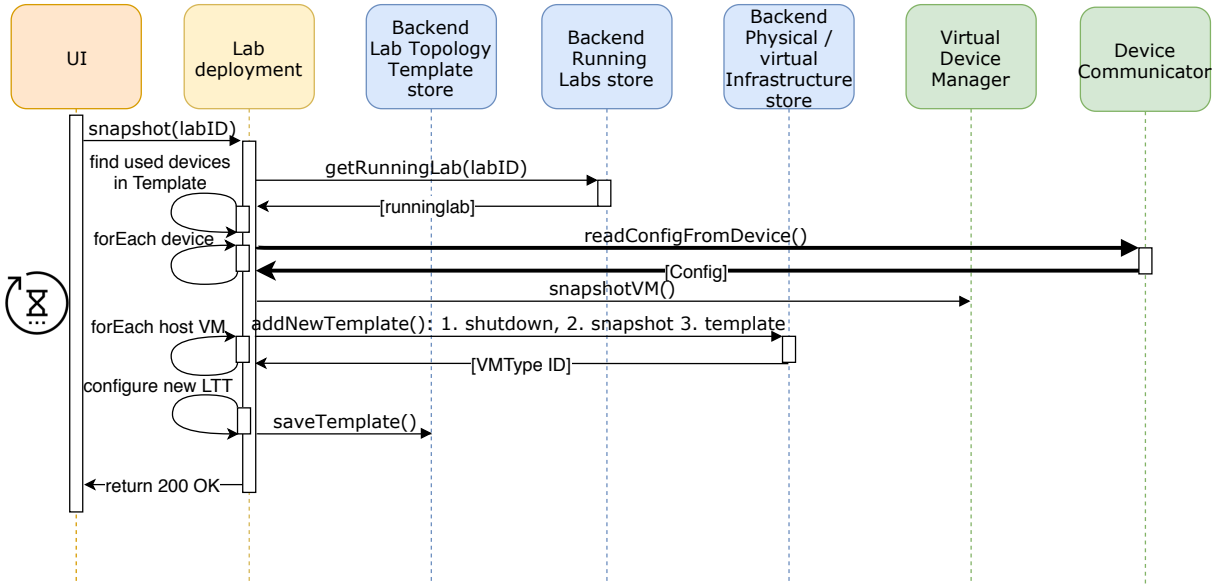



Abbildung 16: Sequenzdiagramm Lab Snapshoter

### 6.3.3 Orchestration

Nornir hat die Möglichkeit, die Tasks zu parallelisieren. Dazu muss bei der Instanziierung von Nornir die Anzahl der Workers (parallele Threads) mitgegeben werden.
In einem Beispiel sieht das dann etwa so aus, dass die Workflows sequentiell abgearbeitet werden. Sobald ein Workflow mit mehreren Subtasks parallel ablaufen soll (hier Connection hinzufügen), wird dies mit der Anzahl an Workers parallel durchgeführt, bis alle Tasks abgeschlossen sind:

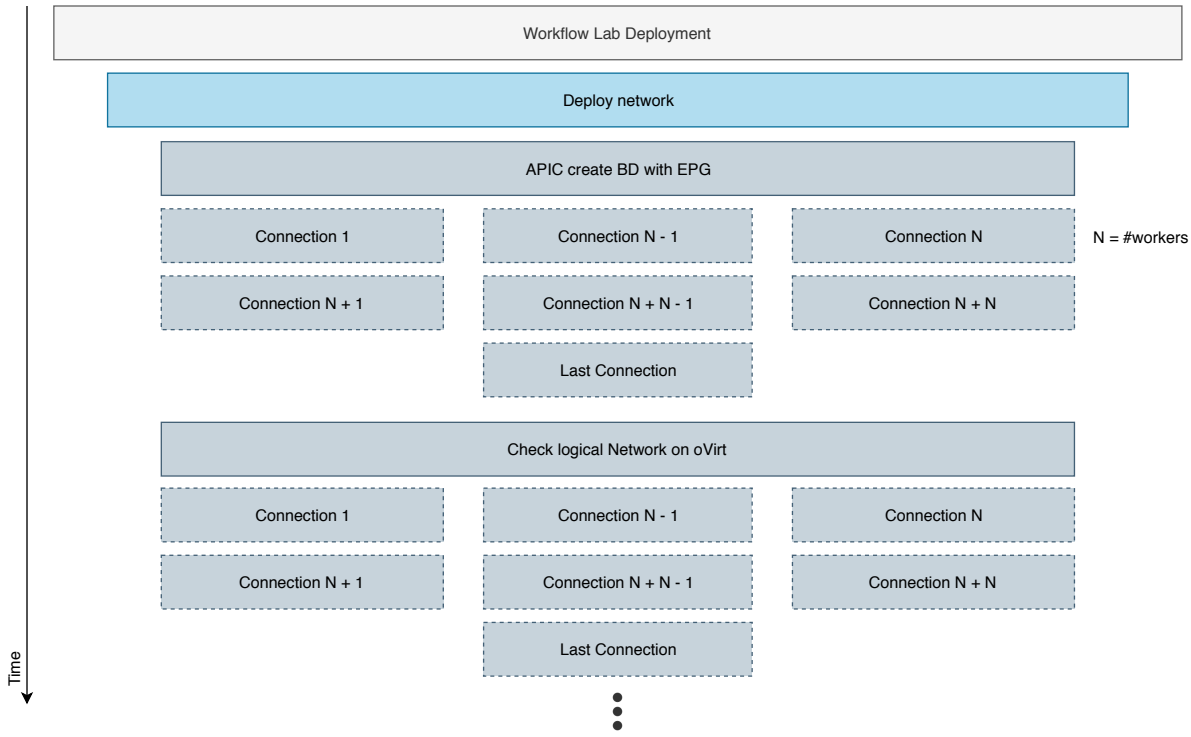



Abbildung 17: Nornir Execution Model [2]

## 6.4 Konzepte

### 6.4.1 Lab Topology Template Aufbau

Das Lab Topology Template beschreibt ein Lab im eigentlichen Sinne.

Als Grundlage für das Template haben wir uns angeschaut, wie so etwas beim Cisco Virtual Internet Routing Lab (VIRL)[16] gehandhabt wird. VIRL ist ein Tool, mit dem ein genaues Modell eines Netzwerks nachgebaut werden kann.
Das Problem am Konzept von VIRL ist, dass XML zur Datenserialisierung verwendet wird, was sehr mühsam zum Schreiben von Hand ist. Auch wurde eine Datenstruktur[15] für die Gerätekonfigurationen gemacht, mit der es sehr aufwendig werden kann, diese sinnvoll vom System hoch- und herunterzuladen. Es müsste jeweils ein ZIP-Ordner des Templates verwaltet werden, in dem alle Dateien in der richtigen Ordnerstruktur vorhanden sind. Dies würde den Aufwand des Import / Exports enorm erhöhen.
Unser Template soll deshalb pro Lab in einem einzigen File verwaltet werden können und von Menschen und Maschine in derselben Form gut lesbar sein, nämlich im *YAML* Format.

Folgendes Diagramm zeigt auf, was alles in einem solchen Template enthalten sein muss und wie die Beziehungen der Entitäten ist.

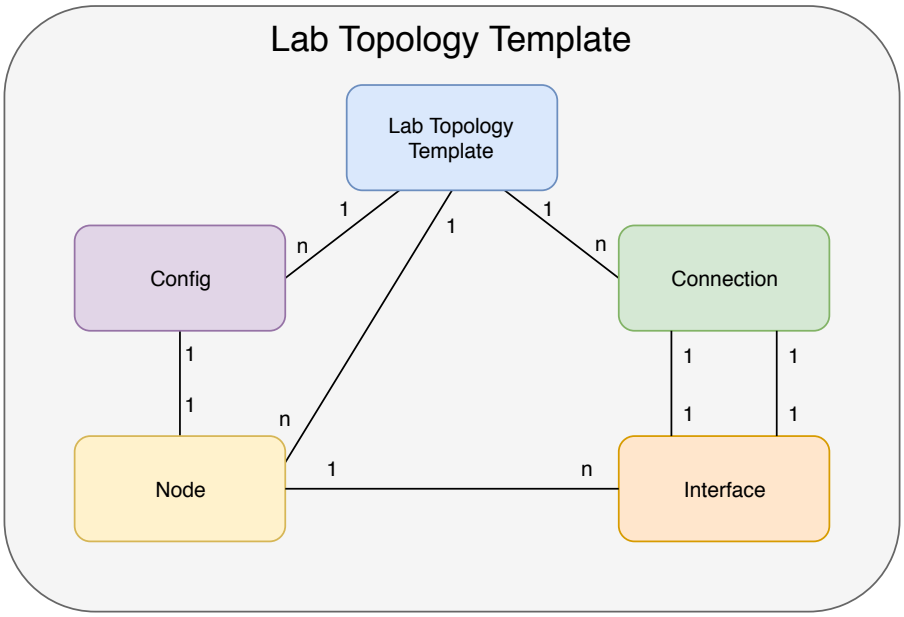



Abbildung 18: Domainmodell des Lab Topology Templates

### Lab Topology Template

Das Template sieht dann in etwa so aus:

```
---
description: Sample Lab SA
virtual_network_devices:
  R1:
    type: csr1000v
    version: 16.0.2
physical_network_devices:
  R2:
    type: catalyst-2821
    version: 16.0.1
virtual_machines:
  Linux1:
    template: ubuntu_18.04
connections:
  -
    R1: 1
    R2: 1
    type:
      name: vcable+
      jitter: 5
      packet-drop: 10%
      delay: 10ms
  -
    Linux1: 1
    R1: 2
running_configs:
  R1: |
    hostname R1
    !
    interface FastEthernet0
    ip address 10.0.1.0 255.255.255.0
    no shutdown
    interface FastEthernet1
    ip address 10.0.2.0 255.255.255.0
    no shutdown
  R2: |
    hostname R2
    !
    interface FastEthernet0
    ip address 10.0.1.1 255.255.255.0
    no shutdown
```

Die wichtigsten Abschnitte sind:

**virtual_network_devices** Hier werden die virtuellen Netzwerkgeräte als YAML-Dictionary verwaltet. Der Key stellt den Gerätenamen dar

**physical_network_devices** Hier werden die physischen Netzwerkgeräte als YAML-Dictionary verwaltet. Der Key stellt den Gerätenamen dar

**virtual_machines** Hier werden die virtuellen Maschinen (Linux, Windows etc.) als YAML-Dictionary verwaltet. Der Key stellt den Gerätenamen dar

**connections** Hier werden in einem YAML-Array die Verbindungen unter den oberen drei Gerätetypen erstellt. Das Array enthält ein Dictionary, in dem der Key den oberen drei Gerätenamen und die Value der Interface-Nr. entsprechen muss. Als optionales Element kann der Key «type» hinzugefügt werden, welcher wiederum ein Dictionary mit Attributen zum vCable+ (siehe vCable+, Seite 46) enthalten (name, jitter, packet-drop, delay), um eine langsame Leitung zu simulieren

**running_configs** Hier werden die Konfigurationen zu den physischen und virtuellen Netzwerkgeräten in einem Dictionary verwaltet. Der Key muss dabei dem Gerätenamen der beiden Gerätetypen entsprechen.

Das Ziel dabei ist es, in einer möglichst generischen und einfach verständlichen Form das Lab in einer einzigen Datei zu erstellen, sodass es später einwandfrei von der Orchestrierung automatisiert deployt werden kann. Diese Datei soll dann einfach von unserem System hoch- und wieder heruntergeladen werden können.

### 6.4.2 Device Label Management

Das Device Label wird in erster Linie im Lab Topology Template gebraucht, um einen Node in einem Lab der entsprechenden Hardware bzw. VM zuzuweisen.
Zum Zeitpunkt des Lab Deployments muss das System wissen, welcher Typ Hardware oder VM zum Einsatz kommen soll.

#### Netzwerkgeräte

So muss zum Beispiel für die Hardware, aber auch für die virtuellen Netzwerkgeräte explizit angegeben werden, welcher Typ und mit welcher Firmware-Version deployt werden soll. Ansonsten kann es vorkommen, dass die Konfiguration nicht mehr richtig funktionieren würde, weil beispielsweise die Netzwerkinterface-Bezeichnung bei verschiedener Hardware anders bezeichnet wird.

#### Virtuelle Maschine

Die Virtuelle Maschine muss über das Label wissen, welches Template verwendet werden soll. Die Templates haben auf der Virtualisierungslösung eine eindeutige Bezeichnung. Diese soll als Label verwendet werden. Falls nicht nur das Template verwendet werden möchte, kann optional ein Snapshot angegeben werden, mit dem noch zusätzliche Konfiguration mitgegeben werden kann.

### 6.4.3 Virtualisierung

Bei der Virtualisierung spielt der Speicherplatz eine grosse Rolle. Es ist entscheidend, wie das Klonen funktioniert, sodass nicht jede VM die volle Festplattengrösse belegt. Es soll eine Methode verwendet werden, bei der die Festplatte des Templates als read-only Master verwendet wird und lediglich die Unterschiede der einzelnen, geklonten VMs auf die Disk geschrieben werden (siehe Copy on write - Speicherplatz Ersparnis, Seite 57).

Die eindeutige Namensgebung der Templates wird für das Labelling im Lab Topology Template verwendet.

Im Optimalfall, und für den Fall, dass verschiedene Snapshots und Templates auf demselben Image aufbauen, macht es durchaus Sinn, die virtuellen Festplatten der VMs auf ein Netzwerkspeicher (SAN) – z.B. iSCSI oder NFS Target – zu speichern, bei dem eine blockbasierte Deduplizierung[33] stattfindet. Dieser Prozess erlaubt es, zwei identische Blöcke auf dem Storage lediglich einmalig abzulegen und dadurch immens Speicherplatz zu sparen. Ein bekannter Hersteller, der diese Technologie sehr gut im Griff hat ist Netapp[34].

### 6.4.4 Namenskonzepte

#### Running Labs

Damit ein Lab Topology Template mehrfach deployt werden kann, muss jedem Running Lab eine eindeutige ID zugeordnet werden, um Namenskonflikte beim Deployment zu vermeiden. Diese ID wird anschliessend für alle Ressourcen, die zu dem Running Lab gehören, verwendet, um ein Namespacing über die gesamte Architektur zu erreichen.
Die ID muss von einer zentralen Instanz vergeben werden, damit sichergestellt ist, dass niemand zur selben Zeit dieselbe ID bekommt und es somit zu Konflikten in der Namensgebung kommen kann.
Eine sinnvolle Möglichkeit ist, dies über ein auto-Increment in einer relationalen Datenbank zu lösen. Die Datenbank kümmert sich im die Atomarität[62] der Daten.

#### Virtualisierung

Jede virtuelle Maschine braucht einen eindeutigen Namen. Es genügt nicht, diesen nur aufgrund von Informationen im Lab Topology Template zusammenzusetzen, da man das Template mehrfach gleichzeitig deployen können muss und sich die VMs nicht in einem eigenen Namespace befinden.
Das Namespacing wird erreicht, indem den Namen der Devices aus dem Lab Topology Template die Runninglab ID vorangestellt wird.

**Konzept:** *{{Runninglab ID}}_{{VM Name}}*

#### Logical Networks

Die ACI erstellt die logical Networks über die VMM Domain[10] auf dem virtual Device Manager wie folgt. Diese Struktur ist durch die CISCO ACI vorgegeben.

**ACI spezifisches Konzept:** *{{Tenant}}_{{Application Profile}}_{{Bridge Domain Name}}*

#### ACI

Die für uns relevanten Komponenten des Domainmodels der ACI sehen wie folgt aus[11].

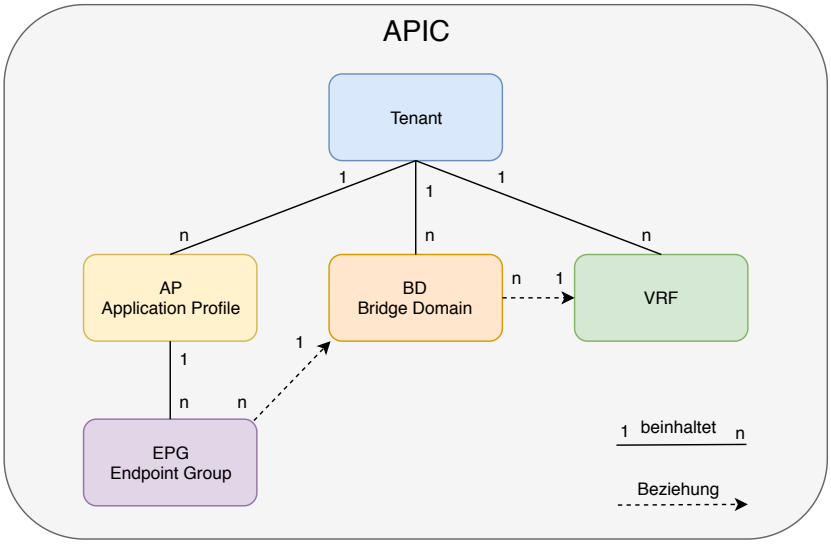



Abbildung 19: ACI-APIC Domainmodel

### Tenant

Der Lab Topology Builder benötigt einen eigenen Tenant. Unter diesem Tenant können die Objekte unabhängig vom Rest der Infrastruktur angelegt werden. Der Tenant besitzt einen eindeutigen Namen.

**Tenant-Name:** *Studienarbeit*

### VRF

Da die BDs kein Subnetz haben, benötigen wir auf Ebene der VRFs keine zusätzliche Virtualisierung. Zudem gibt es keinen eigenen Namespace für die Bridge Domains pro VRF. Die Bridge Domain besitzt lediglich eine Referenz zum VRF.
Es wird ein einziges VRF für die gesamte Lab Topology Builder Applikation verwendet.

**VRF-Name:** *VRF_SA*

### Application Profile

Aufgrund der folgenden Punkte genügt es, wenn lediglich ein Application Profile für alle EPGs Lab übergreifend verwendet wird.

1. Spezielle Funktionen der Application Profiles werden für die reinen Layer 2 Verbindungen nicht benötigt.
2. Die Bridge Domains müssen gemäss Abbildung 19 (Seite 45) pro Tenant einen einzigartigen Namen besitzen. Dies wird ebenfalls mittels der Runninglab ID erreicht (siehe Kapitel 6.4.4, Seite 45). Dadurch sind die Logical Networks bereits einzigartig spezifiziert.

**AP-Name:** *ltb*

### BD

Jede Bridge Domain muss pro Tenant einen eindeutig identifizierbaren Namen besitzen (siehe Abbildung 19, Seite 45). Deshalb ist es auch hier nötig, die Runninglab ID mit in den Namen hineinzunehmen.

**Konzept:**
*{{Runninglab ID}}_{{Device 1}}_{{Interface Number}}_{{Device 2}}_{{Interface Number}}*

### EPG

Pro BD wird ein EPG erstellt. Dies entspricht einer 1:1 Beziehung. Das EPG wird gleich der BD benannt (siehe Kapitel 6.4.4, Seite 45). Dadurch sind EPGs und BDs direkt einander zuordenbar. Jedes EPG muss ebenfalls einen einzigartigen Namen besitzen, da alle EPGs nur einem Application Profile zugeordnet sind.

**Konzept:**
*{{Runninglab ID}}_{{Device 1}}_{{Interface Number}}_{{Device 2}}_{{Interface Number}}*

### Gerätenamenskonzept

Alle Geräte erhalten einen Namen, der ins Namenskonzept des INS hineinpasst:
Konzept: `XX-NNNNNN`
`X`: zwei Buchstaben als Abkürzung des Gerätetyps
`N`: sechs-stellige fortlaufende Nummer

**Beispiel für eine PDU:** `pd-000001`

### vCable+

Das vCable+ kann schlechte WAN Verbindungen simulieren. Wenn die zusätzlichen Eigenschaften gemäss dem Lab Topology Template (siehe Lab Topology Template Aufbau - Connections, Seite 41) definiert sind, wird die Verbindung durch eine VM getrennt, die einen Jitter, Delay und Packet Drop simulieren kann.

Um diese VM muss sich der User nicht kümmern, da diese eine Verbindung, die im Lab Topology Template definiert wurde, im Hintergrund ersetzt wird.

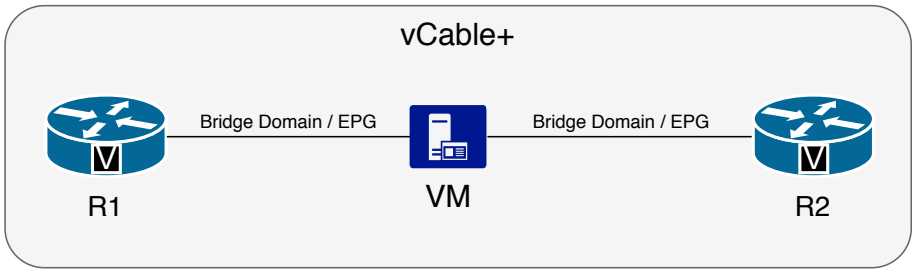



Abbildung 20: vCable+ Umsetzung

## 6.5 Architekturentscheidungen

Die wichtigsten Architekturentscheidungen sind hier festgehalten. Es ist wichtig, dass solche Entscheidungen früh festgehalten werden, um immer wieder Bezug darauf nehmen zu können und damit sich nicht Fehlentscheidungen und Fehler einschleichen, die später nur noch schwer oder gar nicht mehr behoben werden können.

### 6.5.1 Lab deployment - Lab Topology Template handling

In einem ersten Entwurf kann das Lab Topology Template als Datei über ein User Interface (UI) in den Lab Topology Template Store hochgeladen werden. Das Lab Topology Template wird direkt als Datei im Dateisystem gespeichert.
Anschliessend kann die Datei per API ausgelesen werden. Es ist ebenfalls eine Liste aller Templates per API abrufbar.
Dadurch muss in Kauf genommen werden, dass die Templates im Frontend auf ihre Richtigkeit geprüft werden müssen. Der Template Store speichert die Datei dann 1:1 so ab und gibt sie auch auf die gleiche Weise wieder zurück.

### 6.5.2 Lab deployment - Device configuration

Das virtual Device Management stellt einen Proxy zur Verfügung der per SSH angesprochen werden kann. Der Proxy baut über *virsh* die Verbindung zur entsprechenden VM auf.
Die physikalischen Netzwerkgeräte werden ebenfalls über Netmiko oder Napalm via dem Terminalserver konfiguriert. Dieser stellt eine direkte Verbindung auf einen SSH oder Telnet Port zur Verfügung.

### 6.5.3 Backend - Runninglab Store

Für das Speichern der Runninglabs, sowie das Reservationsmanagement der physischen Netzwerkgeräte wird ein separater Container mit einer API und Datenbank verwendet. Damit kann durch die relationale Datenbank eine *unique* Runninglab ID erstellt werden, die in der ganzen Applikation eindeutig ist.
Der Name der physischen Netzwerkgeräte ist ebenfalls *unique* und entspricht ebenfalls einem Primary Key. Damit wird sichergestellt, dass ein Gerät nicht zweimal reserviert werden kann.

# 7 Software Evaluation

Für das Endprodukt ist es wichtig, die richtigen Tools auszuwählen, die später miteinander kommunizieren sollen.
Im Folgenden werden die Anforderungen im Wesentlichen bewertet, um eine gute Abwägung zu erstellen und schlussendlich eine Entscheidung treffen zu können.
Als kleiner Guide dient die «NetworkToCode - Awesome Network Automation»[39] Tools und Library Sammlung, die uns einen Überblick über die möglichen Tools und Libraries verschafft.

## 7.1 Virtualization - Virtual Device Manager

Die Virtualisierungslösung wird verwendet, um virtuelle Netzwerkgeräte, aber auch virtuelle Hosts wie Linux und Windows laufen zu lassen.
Ebenfalls kann eine solche VM für das vCable+ verwendet werden.

| Produkt | OpenSource | VMM Integration | HTTP API | Maintainability |
|---|---|---|---|---|
| RedHat EV | ✗ | ✓ | ✓ | mittel |
| oVirt Engine | ✓ | ✓ | ✓ | mittel |
| VMWare vCenter | ✗ | ✓ | ✓[57] | mittel |
| OpenNebula | ✓ | ✗ | ✓[45] | gut |
| OpenStack | ✓ | ✓ | ✓ | schlecht |

**Kriterien:**
**MUST** OpenSource
**MUST** HTTP API
**SHOULD** Cisco ACI VMM Integration
**SHOULD** Maintainability (siehe Maintainability, Seite 20)

oVirt ist die OpenSource Version von Red Hat Enterprise Virtualization, bei der man nicht offiziell Support von Red Hat erhält, die Funktionalität aber trotzdem fast dieselbe ist.
Wir wollen oVirt aus verschiedenen Gründen als Virtualisierungslösung einsetzen.

- Ist die OpenSource Variante von Red Hat EV
- Unterstützt die direkte VMM Integration in die Cisco ACI
- RESTful HTTP API-Integration
- VM Template Management mit Base-Image

In einem kleinen Demo-Aufbau während der Analyse-Phase wurde das Konzept von oVirt schnell verstanden und es überzeugte uns, dass es das passende Produkt für unser Projekt ist.

## 7.2 Powerstrip PDU

Um die physischen Netzwerkgeräte nach Bedarf automatisiert ein- und wieder auszuschalten werden sogenannte PDUs verwendet, welche über das Netzwerk gesteuert werden können, um einzelne Steckdosen zu steuern. Am INS sind bereits PDUs vorhanden, welche sich aber nur mühsam über Simple Network Management Protocol (SNMP) ansteuern lassen. Hinzu kommt, dass jeweils immer alle acht Ports einer PDU gleichzeitig in einem binären Muster geschaltet werden müssen, sodass man sich den aktuellen Zustand in der Software merken muss.
Wir haben den Auftrag erhalten, eine geeignete PDU zu evaluieren, welche dann für den Einsatz im Lab eingesetzt werden kann. Folgende Hersteller kommen in Frage:

| Hersteller | Bewertung |
|---|---|
| Value | ✓Günstiger Preis<br>✗ Die API ist nicht dokumentiert [6] |
| Intellinet | ✗ Die API ist nicht dokumentiert [28] |
| WTI | ✓Könnte mit Terminal-Server kombiniert werden<br>✗ Sehr teuer<br>✗ Keine direkte Bestellmöglichkeit in der Schweiz [64] |
| GUDE | ✗ Keine direkte Bestellmöglichkeit in der Schweiz [24] |
| Black-Box | ✗ Sehr teuer [5] |
| NetIO | ✓Gute API Referenz [37]<br>✓API Demo zum Testen vor dem Kauf [36]<br>✓Gute Preis-Leistung<br>✗ Haben nur vier steuerbare Stromstecker pro PDU [38] |

Zusammen mit der Betreuung haben wir uns für NetIO entschieden, weil sie das beste Kosten-/Nutzen-Verhältnis besitzen und die HTTP API gut dokumentiert ist.

## 7.3 Backend - Physical Infrastructure Store

In einem Backend sollen alle relevanten Informationen über die physische und die virtuelle Infrastruktur persistiert werden, damit die Labs mit den entsprechenden Referenzen auf eine einfache Art und Weise erstellt, konfiguriert und abgespeichert werden können.
Zudem helfen die hier gespeicherten Informationen der Inventarisierung und Verwaltung der begrenzten Anzahl an physischen Geräten.

| Produkt | Bewertung |
|---|---|
| Netbox | ✓Gute, dokumentierte API<br>✓Opensource |
| NSoT | ✓Opensource<br>✗ Eher mässig gut dokumentiert<br>✗ Nicht alle nötigen Attribute können abgespeichert werden |
| Eigene Implementation | ✓Es kann dieselbe Technologie wie für das Backend verwendet werden<br>✗ Es muss die ganze Architektur selber modelliert und umgesetzt werden |

Netbox scheint für uns eine gute Möglichkeit zu sein, da eine Testinstallation gezeigt hat, dass alle Daten gemäss der Software Architekturanalyse Backend - Physical / Virtual Infrastructure Store darin abgespeichert werden können.

## 7.4 Backend - Eigene Implementation

Die Lab Topology Templates und die Runninglabs müssen in einer strukturierten Form persistiert werden. Dabei gibt es zwei verschiedene Aspekte zu berücksichtigen. Zum einen braucht es eine API, die unsere Schnittstelle zwischen Daten und Consumer darstellt, und zum anderen müssen die Daten in einer einheitlichen Form abgespeichert werden, sodass sie einfach zugreifbar sind.

Das Backend wird aus verschiedenen Gründen in Python umgesetzt. Einerseits soll dies ein guter Lerneffekt für uns haben, da wir beide noch sehr wenig Erfahrungen in dieser Programmiersprache haben. Andererseits ist ein grosses Knowhow in Python bereits am INS vorhanden, sodass die Weiterentwicklung gesichert ist. Es kommen folgende Frameworks in Frage:

| Framework | Bewertung |
|---|---|
| Flask | ✓Wird häufig nur als API verwendet (Pinterest etc.) [55]<br>✓Gut geeignet für kleine Anwendungen<br>✓Datenbank kann selbst gewählt werden |
| Django | ✓Gut für grosse Anwendungen geeignet<br>✗ Bringt per Default eine grosse Umgebung mit sich |

Django ist ein grosses, komplettes Web-Framework mit vielen Funktionen und Flask ist eher ein leichtgewichtiges Framework für APIs. Letzteres ist genau das, was wir brauchen, um unsere Schnittstelle zur Verfügung zu stellen. Es muss simpel und leichtgewichtig sein.

Als Datenbank darunter sehen wir ebenfalls verschiedene Lösungsansätze:

| Technologie | Bewertung |
|---|---|
| Dateibasiert | ✓Gut verständlich, leserlich und geordnet<br>✗ Dateiexport und -import muss dieselbe Struktur besitzen<br>✗ Bei jeder Datenabfrage muss das File gelesen, geparsed und verarbeitet werden |
| SQL (Postgres) | ✓Relational und normalisiert<br>✗ Grosser Overhead |
| NoSQL (NeDB) | ✓Einfach strukturier- und veränderbar<br>✗ Nicht so performant bei sehr vielen Datensätzen |
| In Memory | ✓Sehr schnell<br>✓Gut unterstützt von Python |

Da Flask eine sehr gute Integration von Postgres mittels SQLAlchemy besitzt scheint dies die geeignetste Lösung zu sein.

## 7.5 API Documentation

Als API Dokumentation wird Swagger verwendet. Swagger wird in vielen verschiedenen Programmiersprachen automatisch unterstützt, zum Beispiel mittels Annotationen.
Ein grosser Vorteil liegt darin, dass im Code durch die Annotationen die gesamte API-Dokumentation automatisch erstellt wird.
Die automatisch konfigurierte Swagger-Dokumentation kann im «JSON»-Format exportiert werden und dann dazu genutzt werden, die gesamte API in einer anderen Programmiersprache generieren zu lassen. Der Implementationsaufwand beim Wechsel auf ein anderes Framework ist somit relativ klein.

## 7.6 Load Balancer

Als Load Balancer wird Traefik [18] eingesetzt, weil dieser sehr gut geeignet ist in einer Docker-Umgebung den Traffic zu verwalten. Falls das Projekt skaliert werden muss, ist Traefik ebenfalls eine sehr gut geeignete Lösung, weil er nicht nur Reverseproxy-Funktionalität, sondern auch den Part des Load Balancers übernehmen kann.

Wir haben uns für Traefik entschieden, da dieser bereits am INS für das Loadbalancing des Docker Swarm Clusters eingesetzt wird und unsere Bedürfnisse abdeckt. Zudem kann er die SSL Terminierung für die Docker Container übernehmen und kann selber als Docker Container laufen.

## 7.7 Orchestration Tool

Die Orchestrierung soll den gesamten Ablauf des Lab Deployments, Lab Removals und Lab Snapshoters automatisieren.
Das Tool muss somit wichtige Kernaufgaben lösen, die gewisse spezielle Anforderungen mit sich bringen.
Neben der guten Erweiterbarkeit und Wartbarkeit steht vor allem die Prozess Parallelisierung im Vordergrund. Zum Beispiel müssen die VMs parallel geklont, gestartet und konfiguriert werden können. Würde dieser Prozess sequentiell verlaufen, dann müsste man zu lange auf das Deployment warten.

| Produkt | Forks/Joins | Sub-Workflows | Integrationen | API | Maintainability |
|---|---|---|---|---|---|
| Ansible | ✓ | ✗ | gross | ✗ | viele |
| StackStorm | ✓ | ✓ | gross | ✓ | viele |
| OpenStack Mistral [41] | ✓ | ✓ | keine | ✓ | schlecht |
| Nornir | ✓ | ✓ | keine | ✗ | gut |

**Kriterien:**
**MUST** Forks/Joins bzw. Parallelisierung muss unterstützt sein
**MUST** Es muss möglich sein Workflows und Actions zu verschachteln, damit man genügend Freiheiten hat
**SHOULD** API: Es wäre von Vorteil, wenn es bereits eine API gibt, ansonsten muss eine dazu entwickelt werden, um die Workflows anzustossen
**SHOULD** Maintainability (siehe Maintainability, Seite 20)
**OPTIONAL** Bereits vorhandene Tool Integrationen sind schön, wenn sie vorhanden sind, aber definitiv optional, solange es eine Möglichkeit gibt Python Skripte oder HTTP Requests zu erstellen

Ansible unterstütz leider keine Sub-Workflows und bietet keine API von Haus aus an. Auf der einen Seite gäbe es den Ansible Tower, der das Workflow handling und eine API anbieten würde, jedoch ist dieser Teil nicht open-source [25].

StackStorm bietet alle Funktionen an, jedoch ist die Umgebung nicht sehr «Leichtgewichtig», da sie aus mehreren Services besteht. Zudem müssen die Workflow-Definitionen in YAML geschrieben werden.

Wiederum könnte OpenStack eingesetzt werden, da es ebenfalls eine Workflow Engine besitzt (StackStorm verwendet diese im Hintergrund). OpenStack ist jedoch ein riesiges Tool und dadurch ebenfalls nicht «Leichtgewichtig» gemäss Maintainability auf Seite 20.

Nornir ist im Gegensatz zu den anderen Tools nur ein auf Python basierendes Framework und kein eigentliches Tool. Dafür kann es in einen sehr «leichtgewichtigen» Flask Container mit einer API integriert werden. Zudem kann der ganze Workflow direkt in Python geschrieben werden. Tasks können beliebig tief verschachtelt werden.

Aufgrund dieser Kriterien haben wir uns für Nornir entschieden, da es sehr leichtgewichtig ist und man normalen Python Code schreiben kann, der ebenfalls mittels Python getestet werden kann.

## 7.8 Device Communicator

Der Device Communicator kommuniziert mit den physischen und den virtuellen Netzwerkgeräten. Er verbindet sich automatisiert per SSH / Telnet auf die einzelnen Devices und führt dort die entsprechenden Befehle aus, um die Geräte zu konfigurieren. Die Befehle erhält er von der Konfiguration des jeweiligen Gerätes im Lab Topology Template.

| Produkt | Bewertung |
|---|---|
| Napalm / Netmiko | ✓Wird von SaltStack, Ansible und StackStorm unterstützt [32]<br>✓Nornir baut auf Paramiko, Netmiko und Napalm auf |
| Eigene Implementation (Flask, Django) | ✓Es kann dieselbe Technologie wie für das Backend verwendet werden<br>✗ Es muss die gesamte Logik selber implementiert werden |

# 8 Umsetzung

## 8.1 Infrastruktur Aufbau

Folgende Hardware steht uns im Rahmen des Projekts zur Verfügung:

- 2x Fijitsu PRIMERGY RX2530 M1 Server (oVirt Nodes)
- 3x VMs (oVirt Engine, NFS Server, LTB Server)
- CISCO ACI mit APIC (1 Spine, 2 Leafs)
- 4x Cisco Catalyst 2821 Router
- 6x NetIO PDUs (je 4 Anschlüsse)
- CISCO Catalyst 3725 (Terminal Server)

Diese ist in Abbildung 21 (Seite 54) schematisch erläutert.

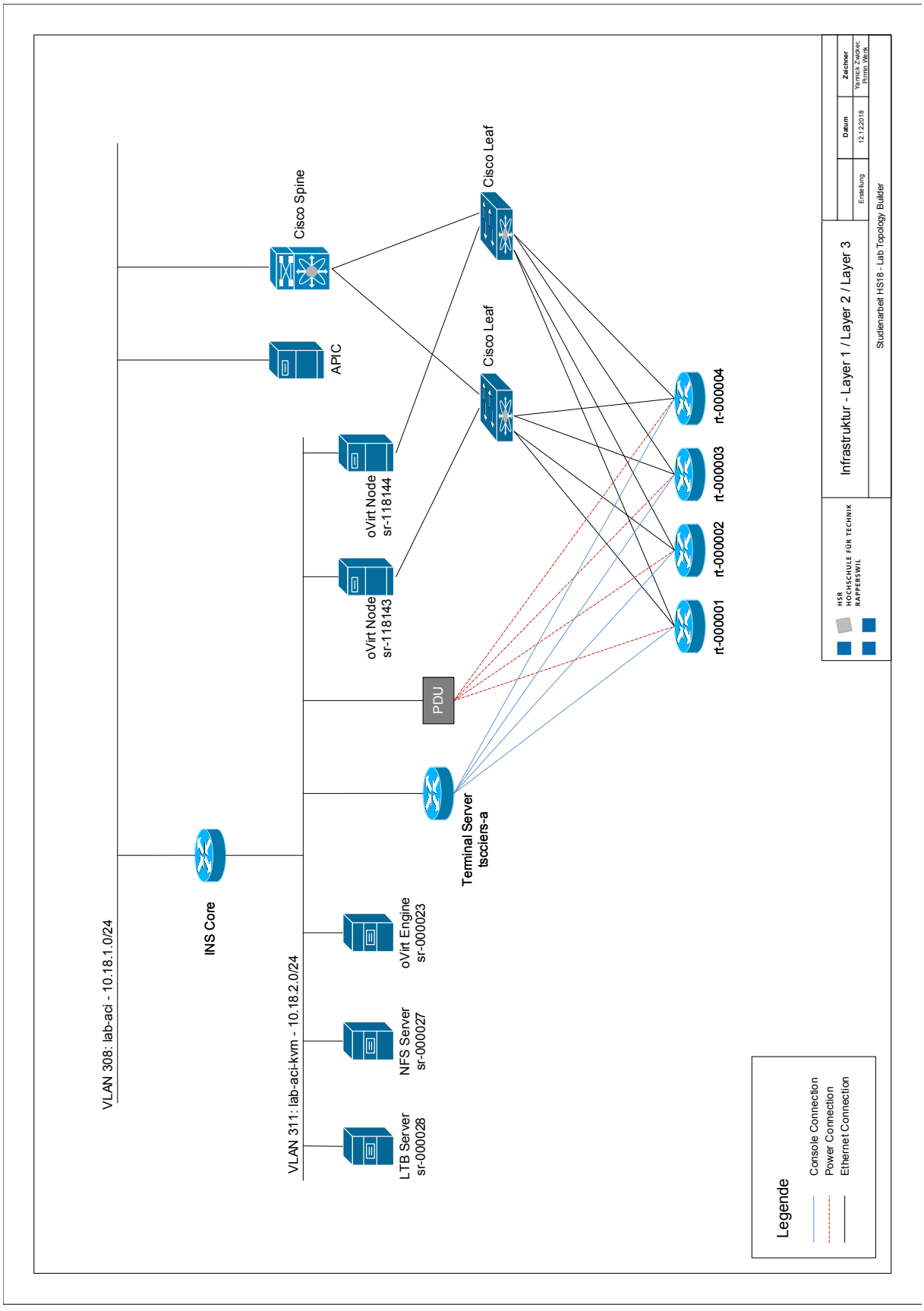



Abbildung 21: Physical Infrastructure

### 8.1.1 Cisco ACI

Die Cisco ACI ist bereits vorhanden und aufgesetzt. Diese wird als Kernelement der Arbeit eingesetzt. Die ACI wird durch den APIC gesteuert, welcher die Konfiguration der Spine- und Leaf-Switches vornimmt.

Eine Einführung in die Konfiguration der Cisco ACI wurde uns von Prof. Laurent Metzger gegeben, da dies sehr umfangreich ist und viele Objekte für eine funktionierende Konfiguration angelegt werden müssen.
Die einzelnen Konfigurationen wurden als Screenshots im Anhang beigelegt (siehe APIC Configuration, Seite 95).
Unsere ACI besteht aus einem Spine- und zwei Leaf-Switches.

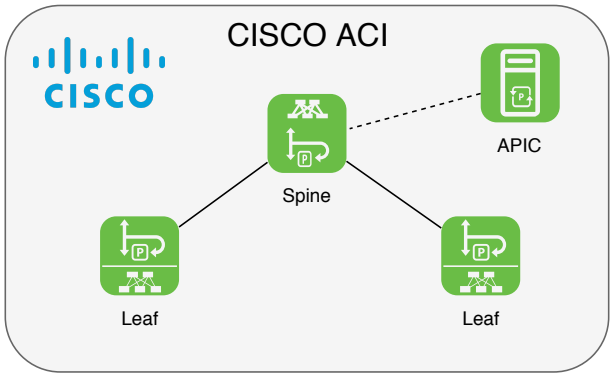



Abbildung 22: Cisco ACI Spine and Leaf Topologie mit APIC

### 8.1.2 Virtual Device Management - oVirt

Eine oVirt-Umgebung besteht aus einem Manager (oVirt Engine) und mehreren Nodes (Server Hosts). Der Manager läuft auf einem virtuellen CentOS-Linux System und ist zuständig für die Verwaltung der Datacenters, die Nodes sind jeweils auf einem Fujitsu Server mit oVirt Node 4.2[47] aufgesetzt.
Die Installation von oVirt ist straight-forward und eine Schritt für Schritt Anleitung befindet sich in der oVirt Dokumentation[48].

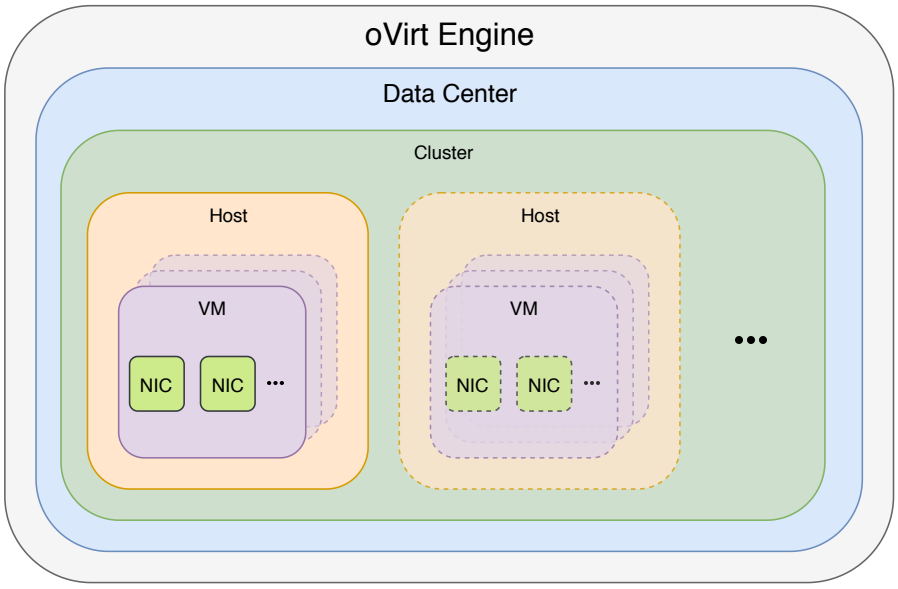



Abbildung 23: oVirt Engine Aufbau

### DataCenter
Ein Datacenter ist eine logische Bündelung von Ressourcen in der Form von Cluster und Nodes.
Ein Datacenter kann mehrere Cluster beinhalten, welche wiederum mehrere Nodes beinhalten können.
Verschiedene Storage-Domains (z.B. NFS, GlusterFS etc.) können einem Datacenter angehängt werden. Es ist dabei zu beachten, dass lokale und shared Storage-Domains nicht gleichzeitig im selben Datacenter verwendet werden kann. Möchte man also von Features wie Virtual Machine Migration vom einen auf den anderen Host profitieren, muss ein shared Storage verwendet werden und der lokale Speicherplatz der Server kann nicht direkt verwendet werden.

### Cluster
Ein Cluster ist eine logische Bündelung von Hosts, welche dieselben Storage-Domains miteinander teilen und dieselbe CPU-Architektur verwenden. Gibt es verschiedene CPU-Modelle, werden nur die Features unterstützt, welche alle Modelle gemeinsam haben [42].
Das Cluster übernimmt die Ressourcenaufteilung auf die Hosts, wobei vollautomatisiert die virtuellen Maschinen auf den Hosts verteilt werden.
Die Netzwerke werden ebenfalls vom Cluster verwaltet. Virtuelle Netzwerkinterfaces werden via oVirt Engine auf die nötigen Netzwerkinterfaces (NICs) der Hosts im Cluster konfiguriert.

### Host
Die Hosts stellen den virtuellen Maschinen ihre Ressourcen zur Verfügung. Sie basieren auf der KVM Virtualisierungslösung [31].

### Storage
Da in einem DataCenter alle Hosts denselben Storage anbinden müssen, ist es ausgeschlossen, dass man den lokalen Storage eines Servers nutzen kann.
Es gibt verschiedene Varianten, wie man einen solchen shared Storage anbinden kann. Dazu gehören die Protokolle NFS, iSCSI, GlusterFS, FCP (Fibre Channel Protocol) [44].

Für uns kommen lediglich NFS oder GlusterFS in Frage, da wir die anderen Protokolle aufgrund der INS Infrastruktur nicht anbinden können.
Wir haben uns für NFS (Network File System) entschieden, der uns über eine VM (NFS Share CentOS-Server 7.0) zur Verfügung gestellt wird.

- ✓Einfach und schnell einzurichten[54]
- ✓Auf den oVirt Nodes wird kein lokaler Speicher benötigt
- ✓Muss nicht zwingend über eine VM zur Verfügung gestellt werden, kann direkt durch ein Storage-System Angeboten werden

Wir haben uns aufgrund der folgenden Gründe gegen GlusterFS entschieden.

- ✗ Für ein gültiges Quorum werden mindestens 3 Nodes benötigt, da man ansonsten ein Split-Brain haben kann [23], was zu Datenverlust führen kann
- ✗ Kompliziertes Setup und vertieftes Wissen fürs Troubleshooting sollte vorhanden sein

### NFS Umsetzung

Bei der Umsetzung des NFS Stores wollten wir zuerst auf den vorhandenen Windows NFS-Store des INS zurückgreifen und dort einen exklusiven Share für unsere Arbeit einrichten. Leider hat dieser Share nie sauber mit oVirt zusammen funktioniert und wir haben den Aufbau wieder abgebrochen.
Mit einer neuen Linux VM (wie oben beschrieben), konnte dann der Storage sauber in die oVirt-Umgebung eingebunden werden.

### VM

Als VM nutzen wir eine Vorlage von virtuellen Routern im `qcow2` Format. Dieses Image kann mit den entsprechenden Rechten von Cisco direkt heruntergeladen werden [12]. Daraus kann dann ein oVirt Template erstellt werden. Dieses besitzt eine Festplatte auf dem NFS Storage und kann sonstige generellen Einstellungen besitzen (siehe oVirt Template erstellen, Seite 93).

Der Vorteil des Templates in oVirt liegt ganz klar darin, dass die Festplatte welche beim Template hinterlegt ist, nicht kopiert werden muss. Es wird pro neuer VM immer nur das Diff zu dem Template abgespeichert. Das Base Image des Templates ist somit read-only und kann von den VMs nicht angepasst werden. Dies hat den Vorteil, dass das Erstellen der VM sehr effizient ist und der Speicherplatz geschont wird. Auch beim Löschen der VM muss nur das Diff gelöscht werden. Das spart viel Zeit und Ressourcen.

### Copy on write - Speicherplatz Ersparnis

Der «cow» Teil vom Dateiformat *qcow2* bedeutet *copy on write* [61]. Dies beschreibt den Mechanismus, dass man ausgehend von einem read-only Base Image jeweils nur immer die Differenz dazu speichern muss.

Im folgenden Vergleich sieht man die Ersparnis von Speicherplatz, wenn man den *copy on write* Mechanismus gegenüber einem kompletten *Klonen* der Festplatte bevorzugt (Berechnung siehe oVirt Speicherplatz Verbrauch, Seite 87).

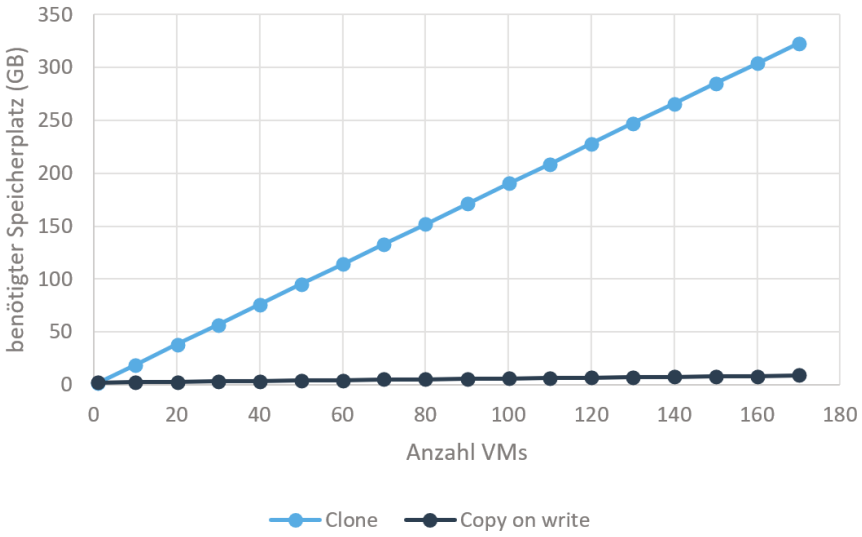



Abbildung 24: Virtual Machine Speicherplatz Vergleich

### 8.1.3 oVirt managen

Wie oVirt gepflegt werden kann, wird im Anhang oVirt Template erstellen (Seite 93) detailliert beschrieben.

### 8.1.4 Docker Setup

Wir sehen den Vorteil in Docker ganz klar darin, dass die Container für sich selbst eigenständig laufen und auch ausgetauscht werden können. Trotzdem kann aber mit einem einzigen Kommando die ganze Umgebung hochgefahren werden.
Im Code-Repository GitLab kann jede Komponente für sich gebuildet werden, wobei dann im Continuous Delivery alle Container gleichzeitig gepullt und gestartet werden können.

#### Network

Über das Docker-Compose File werden vier Docker-Bridges erstellt, damit die Services isoliert voneinander sind. Der Traefik Loadbalancer ist der einzige Docker Container, der direkt von aussen über das *traefik_external* Netzwerk erreichbar ist. Dadurch kann nicht direkt auf die internen Services verbunden werden, was ein zusätzliches Mass an Sicherheit bedeutet.

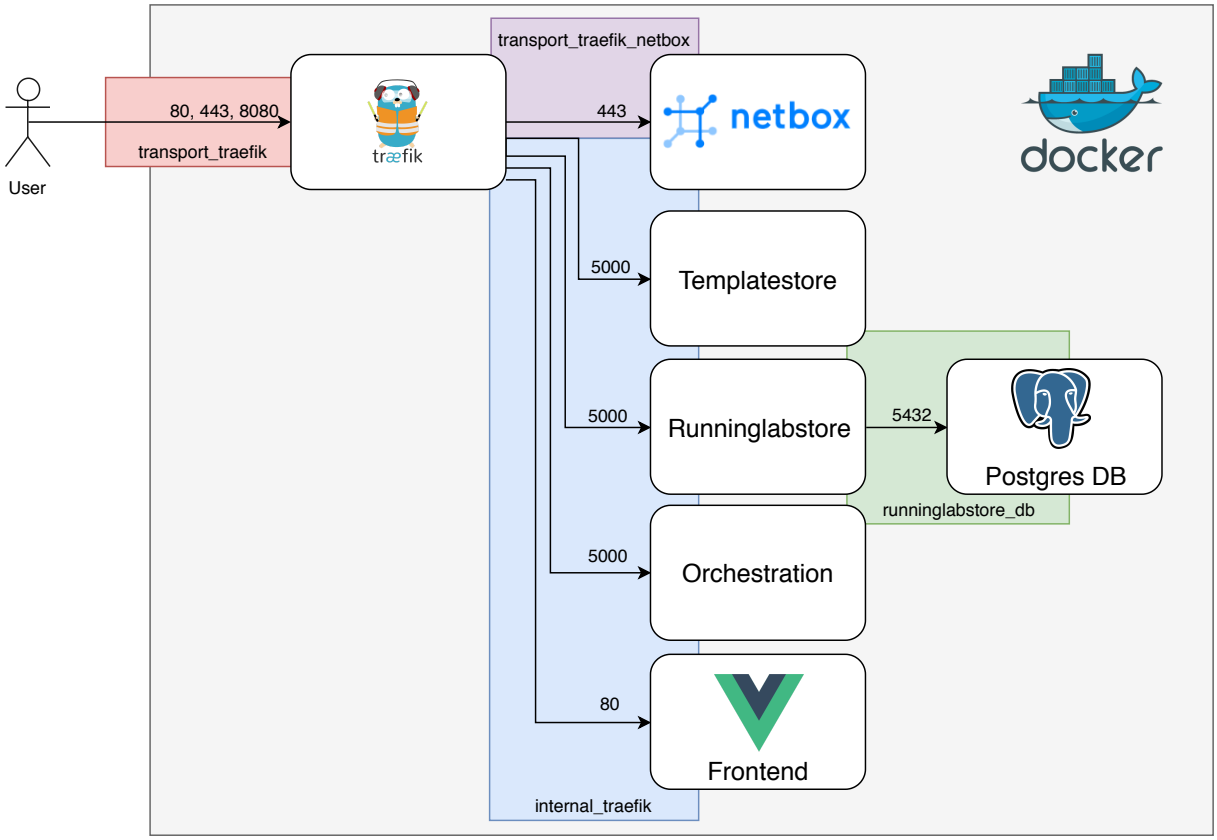



Abbildung 25: Docker Network Setup

#### Traefik Routing

Die Docker Container können sich über Labels dynamisch beim Traefik registrieren. Dadurch können sie einfach horizontal skaliert werden.

Listing 2: Production Frontend Service - docker-compose.prod.yaml

```
frontend:
   image: registry.gitlab.com/mipiya/sa/ltb/frontend:latest
   networks:
     - internal_traefik
   labels:
     traefik.port: 80
     traefik.enable: true
     traefik.backend: frontend
     traefik.frontend.rule: Host:app.ltb.ins
     traefik.docker.network: labbuilder_internal_traefik
```

Die *traefik.frondend.rule: Host* definiert, über welchen Domain-Name die Microservices erreichbar sind.
Wir haben uns für Subdomains entschieden, da dadurch kein komplexes URL-Rewriting gemacht werden muss, wenn nur über den URL-Path zwischen den Microservices unterschieden wird.
Für die Services werden folgende Subdomains verwendet:

- **app**.ltb.ins
- **netbox**.ltb.ins
- **orchestration**.ltb.ins
- **runninglabstore**.ltb.ins
- **templatestore**.ltb.ins

#### Datenpersistenz

Es bestehen zwei Docker Daten Volumes, um die Daten der Postgres Datenbank und die Lab Topology Templates des Templatestores zu persistieren.
Dadurch werden die Daten nicht gelöscht, wenn der Container aktualisiert oder gelöscht wird.

Listing 3: Production Volumes - docker-compose.prod.yaml

```
volumes:
  runninglabstore_postgres_data:
    driver: local
  templatestore_data:
    driver: local
```

### 8.1.5 Deployment

Das Deployment ist durch zwei verschiedene *Docker-Compose* [27] Files in ein *Development* und *Production* Environment getrennt.
Zudem gibt es pro Umgebung eine unterschiedliche *Umgebungsvariablen* Datei.

Für das Setup der Umgebungen siehe Anhang Starten der Dockerumgebung (Seite 90).

### GitLab Registry

Um die gebuildeten Docker Images zu speichern, wurde die direkt in GitLab integrierte Container Registry verwendet. Pro Code Repository kann eine eigene Container Registry verwendet werden.
Unsere Registries sehen wie folgt aus:

- registry.gitlab.com/mipiya/sa/ltb/orchestration
- registry.gitlab.com/mipiya/sa/ltb/templatestore
- registry.gitlab.com/mipiya/sa/ltb/runninglabstore
- registry.gitlab.com/mipiya/sa/ltb/frontend

Dadurch, dass jeder Microservice in einem eigenen Code Repository liegt, existiert pro Service ein eigener Registry-Namespace.

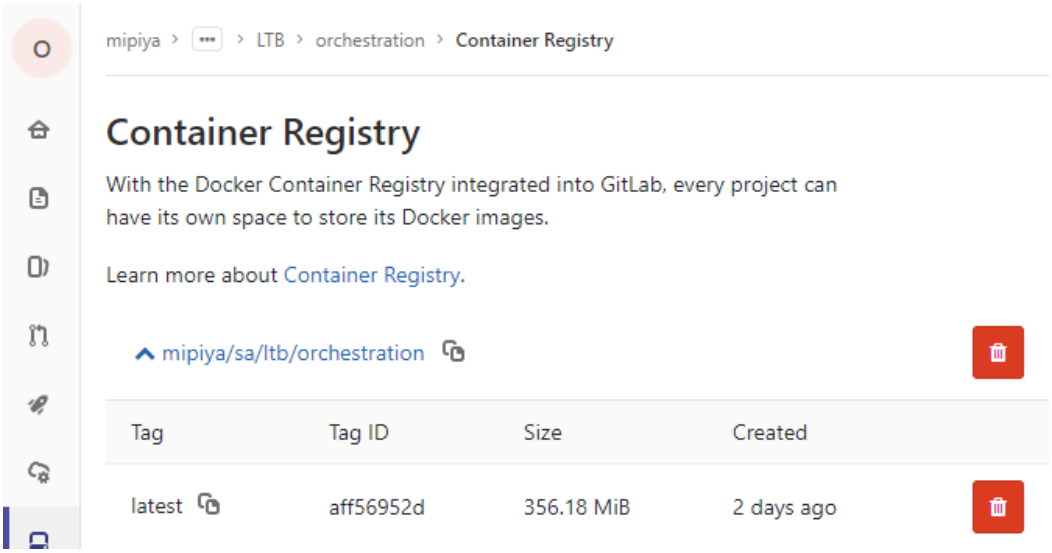



Abbildung 26: GitLab Docker Registry

### GitLab CI/CD

Wird eine Code Änderung auf den Master-Git-Branch gepushed, wird direkt ein neuer GitLab CI/CD Job angestossen [21] (siehe Abbildung 27).

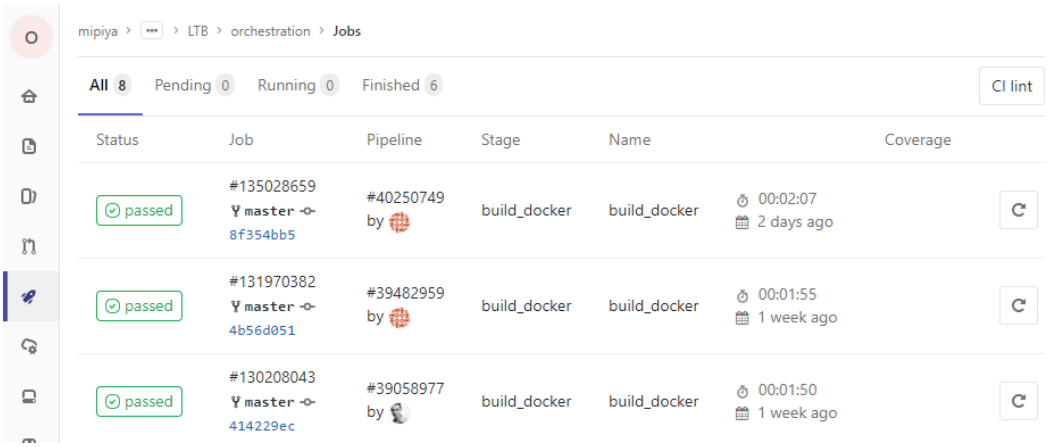



Abbildung 27: GitLab CI/CD Pipeline Jobs

### GitLab CI/CD - Orchestration Repository Configuration Example

Listing 4: Orchestration Repository CI/CD Configuration - .gitlab-ci.yml

```
variables:
  DOCKER_HOST: tcp://docker:2375/
  DOCKER_DRIVER: overlay2

stages:
  - build_docker

build_docker:
  stage: build_docker
  image: docker:stable
  services:
    - docker:dind
  script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker build -f Dockerfile -t registry.gitlab.com/mipiya/sa/ltb/
        ↪ orchestration .
    - docker push registry.gitlab.com/mipiya/sa/ltb/orchestration
```

### GitLab Runner

Damit die GitLab Pipelines gebuildet werden können, haben wir auf einem unserer Server einen GitLab Runner [22] installiert.

Wir verwenden den Docker-in-Docker Build Mechanismus, damit können Docker Container in einem Docker Container gebuildet werden. Dies hat den Vorteil von einer immer sauberen Build-Umgebung, da die Container rückstandslos gelöscht werden können.

Zudem können weiterhin andere Docker Container als Build-Umgebung dienen. Dies wird benötigt um die Build-Artefakte des Frontends zu erstellen, da dort Node.js [17] Libraries für den Build benötigt werden.

Listing 5: GitLab Runner Registration

```
sudo gitlab-runner register -n --url https://gitlab.com/ --registration-
    ↪ token *** --executor docker --description "sr-000028" --docker-image "
    ↪ docker:stable" --docker-privileged
```

Listing 6: GitLab Runner Config - /etc/gitlab-runner/config.toml

```
concurrent = 1
check_interval = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "sr-000028"
  url = "https://gitlab.com/"
  token = "***"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "docker:stable"
    privileged = true
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/cache"]
    shm_size = 0
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]
```

## 8.2 Services

### 8.2.1 Backend

Wir haben zwei Backends, die auf dem Python-basierten Web-Framework «Flask» basieren und ein Backend, das auf dem bewährten Netzwerkverwaltungstool «Netbox» basiert.

Für die Flask Applikationen wird das Zusatzpaket «Flask-RESTPlus»[26] verwendet. Damit kann ganz einfach durch Annotationen im Code eine Swagger-Dokumentation [56] erstellt werden, die komplett selbständig generiert und über eine automatisch generierte Webseite zur Verfügung gestellt wird.

Damit die Ressourcen vom Backend sauber via Frontend geholt werden können, müssen zwingend die CORS-Header auf dem Backend richtig gesetzt werden. CORS schützt die API vor unbefugten indirekten Zugriffen auf die Ressource. Durch den HTTP Response Header «AccessControlAllowOrigin: www.example.ch» kann man spezifischen Webseiten den Zugriff gewähren, auf die Ressource zugreifen zu dürfen (mit der Wildcard: «AccessControlAllowOrigin: *» werden alle Webseiten befugt, auf die Ressourcen zuzugreifen. Folgende Grafik zeigt dies anschaulich auf:

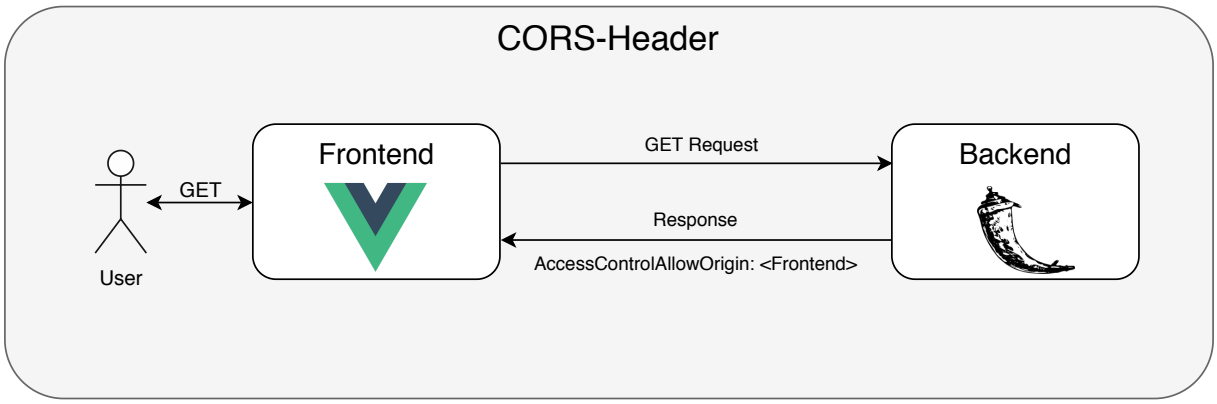



Abbildung 28: Backend CORS Header description

### Templatestore

Der Templatestore läuft auf Flask und nimmt YAML-Files entgegen und gibt diese bei Bedarf wieder zurück. Abgespeichert werden die Files ganz einfach als YAML-File auf dem Filesystem des Docker-Containers in einem Docker Volume.
Folgende Methoden können mit der API verwendet werden. Diese wurden mit Swagger (siehe API Documentation, Seite 51) dokumentiert.

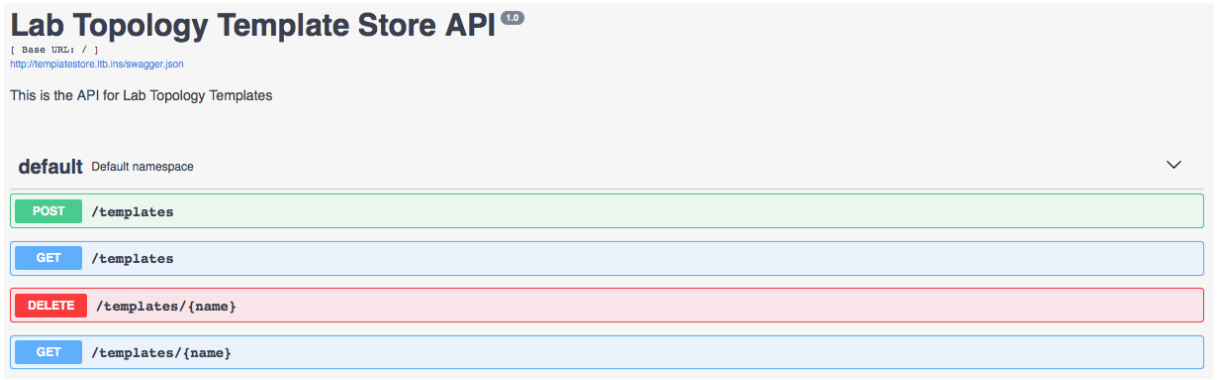



Abbildung 29: Swagger Dokumentation des Templatestores

### Runninglabstore

Der Runninglab store läuft auf Flask und speichert alle nötigen Informationen über ein Runninglab in einer relationalen Postgresql Datenbank ab. Die Relationen und Entitäten werden per Python-Code in einem «Model» niedergeschrieben und automatisch durch das Zusatzpaket «Flask-SQLAlchemy»[53] in die Datenbank programmiert. Flask-SQLAlchemy wiederum basiert auf SQLAlchemy und Psycopg2.
Folgende Methoden können mit der API verwendet werden:

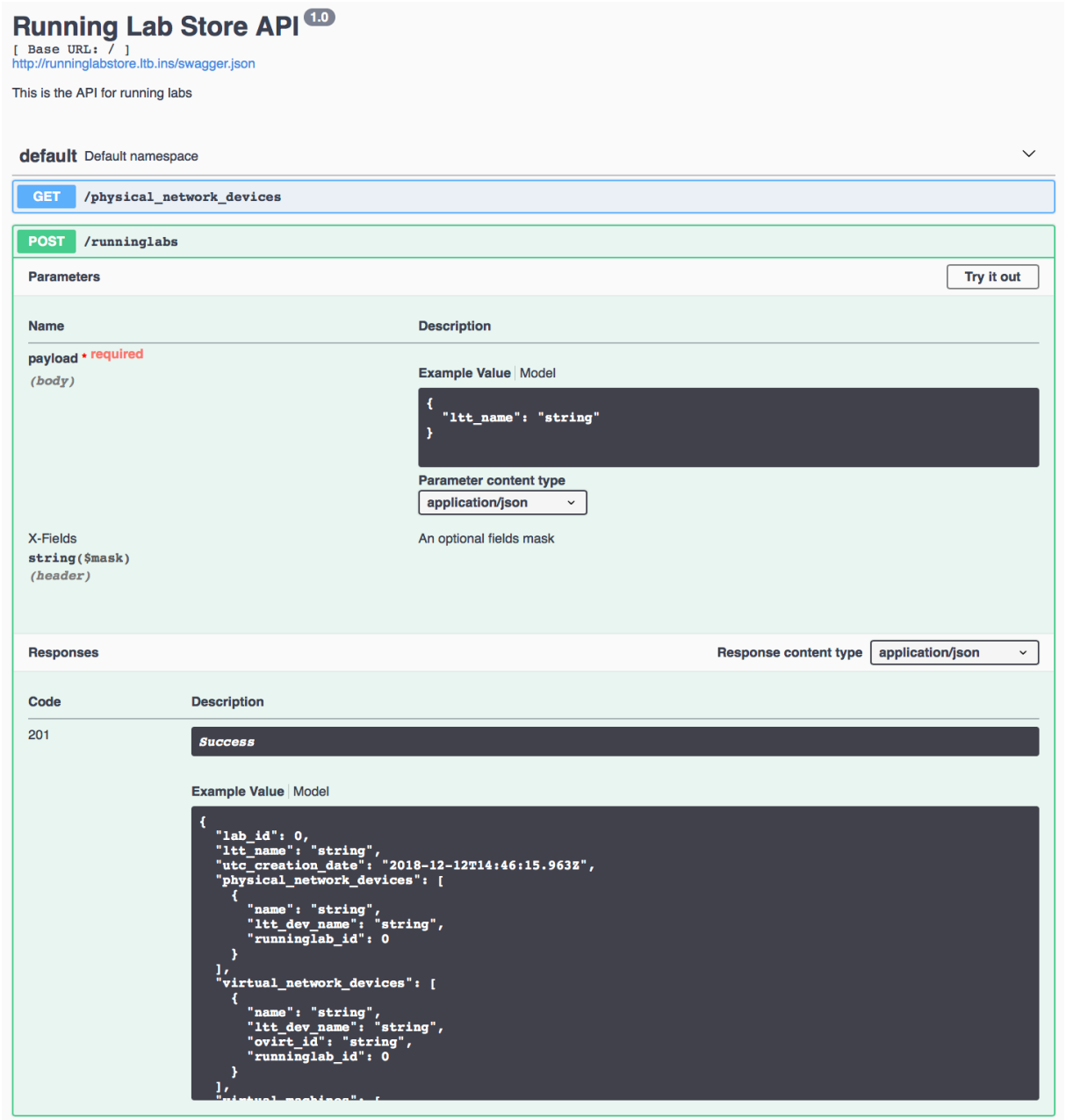



Abbildung 30: Swagger Dokumentation des Runninglab Stores - Teil 1

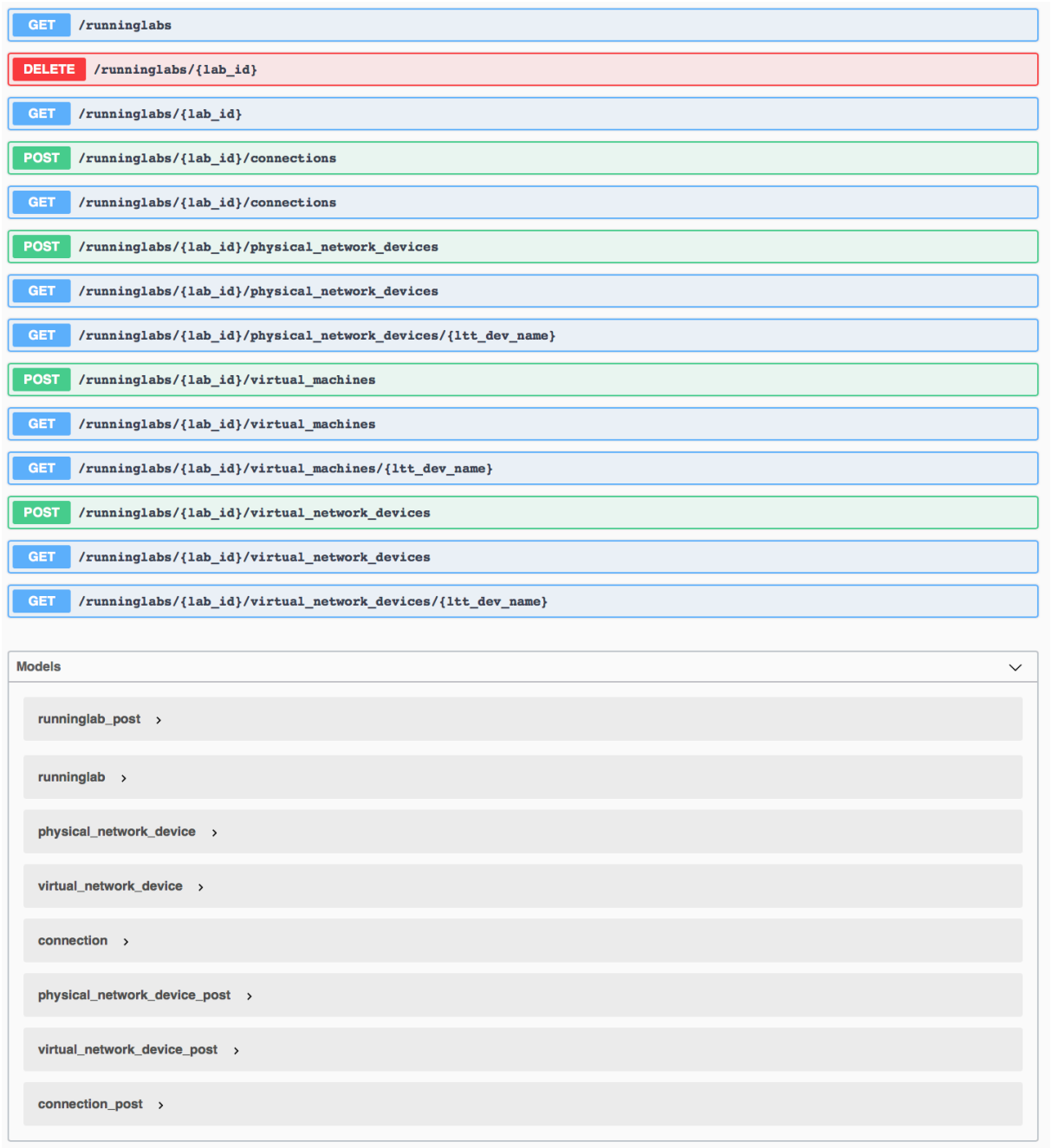



Abbildung 31: Swagger Dokumentation des Runninglab Stores - Teil 2

### Netbox

Netbox [35] dient uns als Single Source of Truth für die physische Infrastruktur und kann per HTTP API abgefragt werden. Alle Komponenten der Topologie Physical Infrastructure (Seite 54) sind darin erfasst und miteinander virtuell verbunden.

## Devices

| Name | Status | Tenant | Site | Rack | Role | Type | IP Address |
|---|---|---|---|---|---|---|---|
| acile01 | Active | LTB | 2.103 | – | Switch Layer 3 | Cisco N9K-C9396PX | |
| acile02 | Active | LTB | 2.103 | – | Switch Layer 3 | Cisco N9K-C9396PX | |
| pd-000001 | Active | LTB | 2.103 | – | PDU | NETIO 4C | 152.96.11.11 |
| pd-000002 | Active | LTB | 2.103 | – | PDU | NETIO 4C | 152.96.11.12 |
| pd-000003 | Active | LTB | 2.103 | – | PDU | NETIO 4C | 152.96.11.13 |
| pd-000004 | Active | LTB | 2.103 | – | PDU | NETIO 4C | 152.96.11.14 |
| pd-000005 | Active | LTB | 2.103 | – | PDU | NETIO 4C | 152.96.11.15 |
| pd-000006 | Active | LTB | 2.103 | – | PDU | NETIO 4C | 152.96.11.16 |
| rt-000001 | Inventory | LTB | 2.103 | – | Router | Cisco Catalyst 2821 | |
| rt-000002 | Inventory | LTB | 2.103 | – | Router | Cisco Catalyst 2821 | |
| rt-000003 | Inventory | LTB | 2.103 | – | Router | Cisco Catalyst 2821 | |
| rt-000004 | Inventory | LTB | 2.103 | – | Router | Cisco Catalyst 2821 | |
| tscciers-a | Active | LTB | 2.103 | – | Terminal-Server | Cisco Catalyst 3725 | 152.96.11.163 |

Abbildung 32: Netbox Devices

### 8.2.2 Frontend

Das Frontend wurde mit dem Progressive JavaScript Framework «VueJS»[29] umgesetzt. Dieses ist ein leichtgewichtiges Framework für Single Page Web Applications und ist schnell zu erlernen. Als Component Framework wurde «VuetifyJS»[30] im Google Material Design eingesetzt.

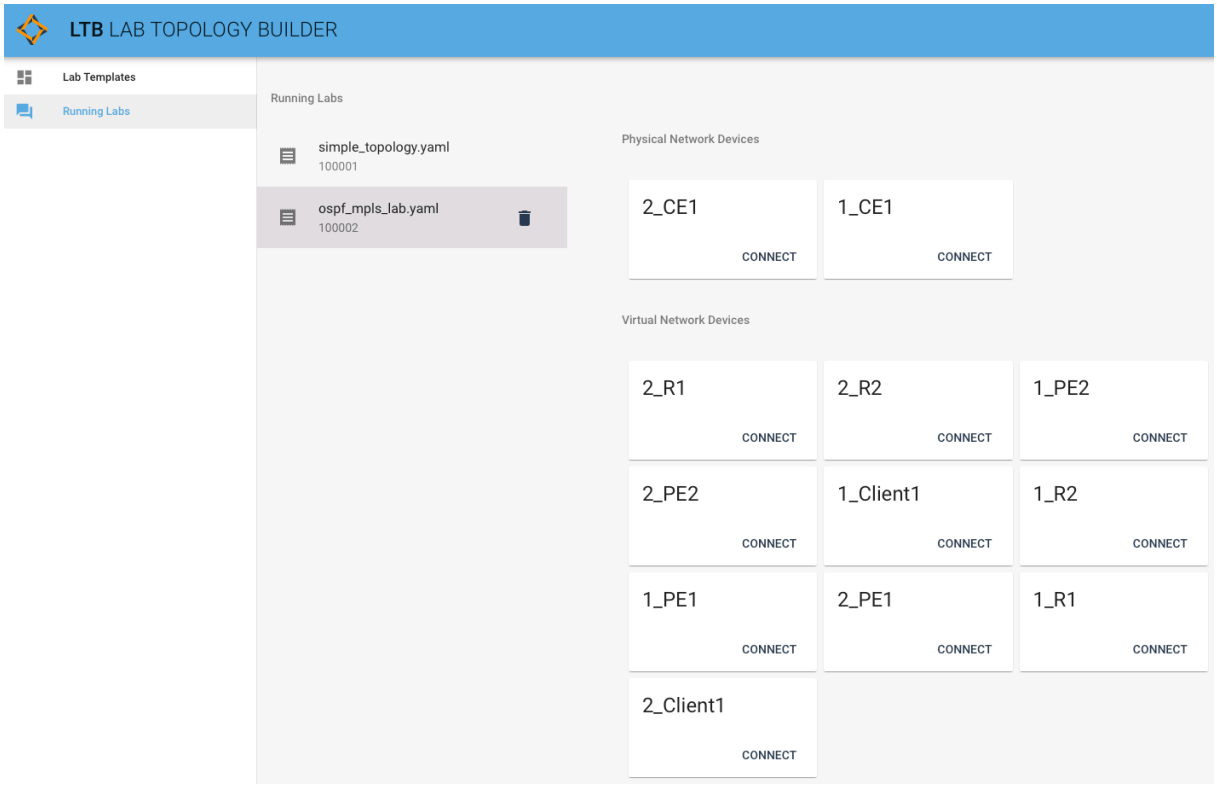



Abbildung 33: User Interface für die Verwaltung der Lab Templates und Runninglabs

### 8.2.3 Orchestration

Die Orchestration mit Nornir bietet eine sehr gute Möglichkeit an, um Tasks in Workflows zu gruppieren und diese parallel auf verschiedene Hosts oder Gruppen anzuwenden.
Diese Eigenschaft wurde dahingehend ausgenutzt, indem das Inventar (Hosts und Groups) direkt zur Laufzeit aufgrund verschiedener Datenquellen dynamisch erstellt werden konnte. Diese Quellen sind z.B. das Lab Topology Template oder das Running Lab aus der Datenbank.

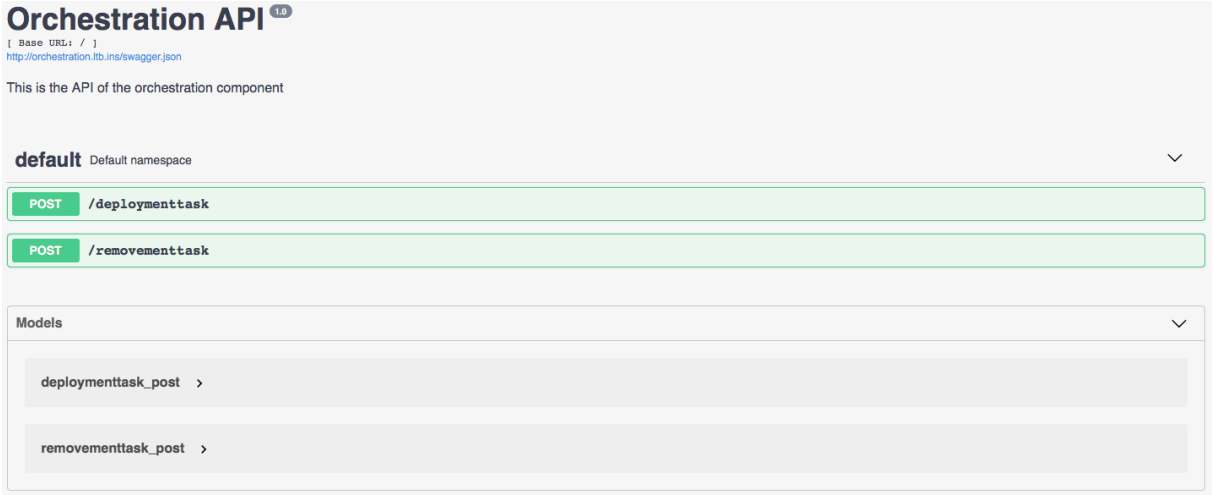



Abbildung 34: Swagger Dokumentation der Orchestration

## Workflows

### Lab Deployment

Aufgrund der Task Abhängigkeiten und den dazugehörigen Sequenzdiagrammen konnte die Orchestration in folgende Tasks und Workflows gegliedert werden. Dabei ist einiges zu beachten:

- dass die Reihenfolge der Pfeile eingehalten wird
- die farbigen Workflows beziehen sich jeweils auf das Inventar der «Hosts». Für jeden Host wird der Workflow simultan abgearbeitet. Erst wenn alle Tasks zu jedem Host des Workflows abgearbeitet sind, kann mit dem nächsten Workflow fortgefahren werden
- die Farben bedeuten jeweils, um welche Art Host es sich handelt (physisches Gerät, virtuelles Gerät, connection)
- tritt ein Fehler auf, wird sofort das Lab Removement angestossen, was die bisher benötigten Ressourcen wieder sauber freigibt

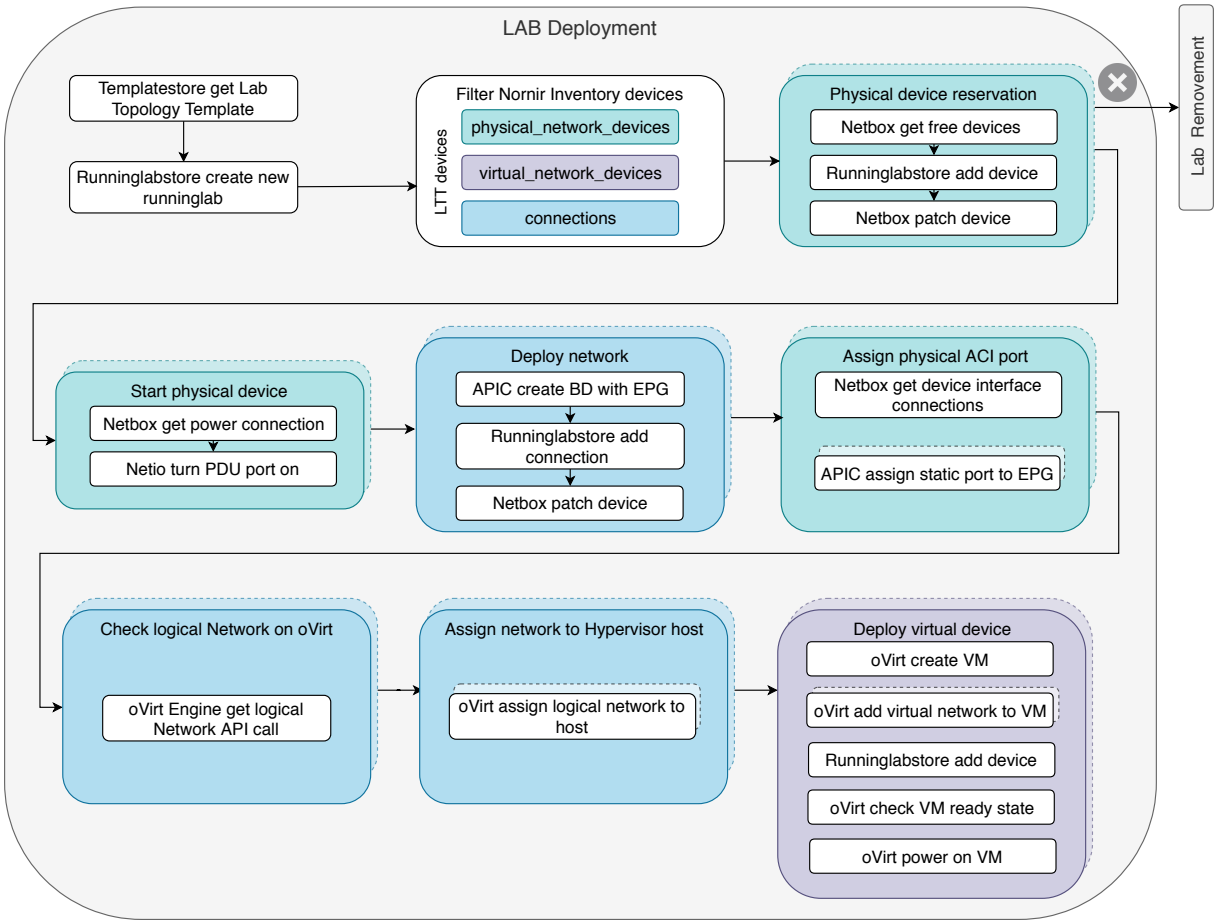



Abbildung 35: Workflow des Lab deployments im Orchestrationtool

### Lab Removement

Auf dieselbe Art und Weise verhält sich das Lab Removement. Dieses basiert auf dem Inventory des Runninglab Stores.

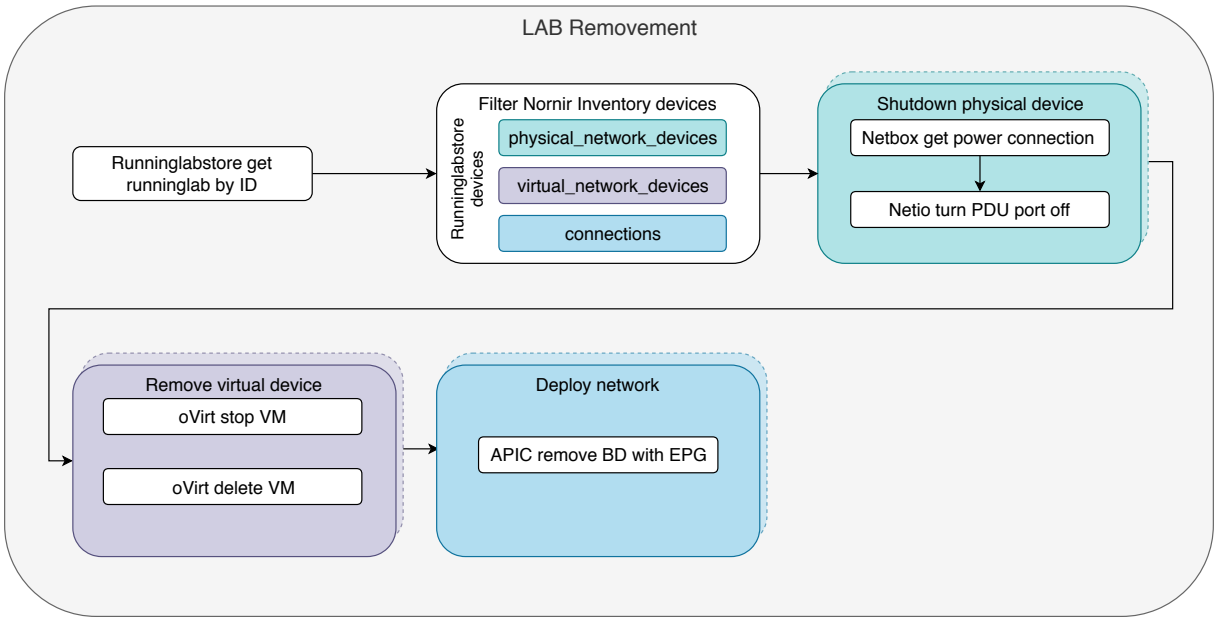



Abbildung 36: Workflow des Lab removements im Orchestrationtool

### 8.2.4 Device Communicator

Der Device Communicator Service wurde lediglich in der Risiko Analyse Phase als Prototyp entwickelt, um zu zeigen, dass es grundsätzlich möglich ist auf die Geräte automatisiert zuzugreifen.
Es ist nämlich möglich, über den folgenden Befehl via SSH auf die virtuellen Netzwerkgeräte des oVirt Clusters zuzugreifen, jedoch funktioniert der *ProxyCommand*[20][60] weder mit der Napalm, Netmiko noch der Paramiko Library, da diese aufeinander aufbauen und keine diesen Implementiert hat.

Der *ProxyCommand* ist der letzte Teil **connect –vm-name csr1000v** des SSH Befehls, welcher direkt beim Login auf dem Zielsystem ausgeführt wird. Dadurch wird dem oVirt Proxy [51] mitgeteilt auf welche VM im Hintergrund eine Konsolen-Verbindung über die KVM Libvirt Library aufgebaut werden soll.

```
ssh -t -p 2222 ovirt-vmconsole@sr-000023.ins -i ~/.ssh/id_ecdsa connect --vm
    ↪ -name csr1000v
```

Eine Erweiterung der Netmiko Library kam aus Zeitgründen nicht in Frage.
Durch die Netmiko Community (siehe Referenzen / Weiterführende Links, Seite 71) sind wir auf zahlreiche ähnliche Issues gestossen.

Es ist möglich über einen sogenannten `redispatch` zuerst auf einen Proxy eine Verbindung zu erstellen mit dem *device_type: linux*, damit der Consolen-Prompt richtig interpretiert wird und danach lokal vom Proxy die weitere SSH Verbindung auf den oVirt Proxy zu öffnen.
Wenn dies erfolgt ist, kann ein `redispatch` durchgeführt werden, um vom *Linux Prompt* auf den *Cisco Prompt* zu wechseln.

Damit die Authentisierung ohne Passwortabfrage funktioniert, konnte ein SSH-Config File angegeben werden.

Listing 7: devicecommunicator/config

```
host *
  IdentityFile ~/.ssh/id_ecdsa
  IdentitiesOnly yes
  user root
  hostname sr-000023.ins
  Port 22
  RequestTTY force
```

Listing 8: devicecommunicator/netmiko_test.py

```
import time
from netmiko import ConnectHandler
from netmiko import redispatch

device = {
    'device_type': 'linux',
    'port': 22,
    'host': 'sr-000023.ins',
    'username': 'root',
    'allow_agent': True,
    'key_file': '~/.ssh/id_ecdsa',
    'ssh_config_file': '~/.ssh/config'
}
net_connect = ConnectHandler(**device)
time.sleep(2)

print(net_connect.send_command('ssh -t -p 2222 ovirt-vmconsole@sr-000023.ins
    ↪  -i ~/.ssh/id_ecdsa connect --vm-name csr1000v', expect_string=''))
time.sleep(2)

print(net_connect.send_command('\n\n\n\n\n\n\n', expect_string=''))
redispatch(net_connect, device_type='cisco_ios')
time.sleep(1)

print(net_connect.send_command('en'))
print(net_connect.send_command('sh run'))
```

### Referenzen / Weiterführende Links

- https://github.com/ktbyers/netmiko/issues/470 [accessed: 31.10.2018]
- https://github.com/ktbyers/netmiko/issues/772 [accessed: 31.10.2018]
- https://github.com/ktbyers/netmiko/issues/633 [accessed: 31.10.2018]
- https://github.com/ktbyers/netmiko/issues/383 [accessed: 31.10.2018]
- https://github.com/ktbyers/netmiko/blob/develop/COMMON_ISSUES.md#does-netmiko-support-connecting-via-a-terminal-server [accessed: 31.10.2018]

## 8.3 Probleme

### APIC

| Severity | Acked | Cause | Creation Time | Affected Object | Description |
|---|---|---|---|---|---|
| ⚠ | ☐ | configuration-failed | 2018-12-12T08:58:5... | topology/pod-1/node-101/local/svc-policyelem-id-0/uni/epp/fv-[uni/tn-Studienarbeit/ap-ltb/epg-100002_1_PE1_1_1_CE1_1]/node-101/stpathatt-[eth1/42]/nwissues | Fault delegate: Configuration failed for uni/tn-Studienarbeit/ap-ltb/epg-100002_1_PE1_1_1_CE1_1 node 101 eth1/42 due to Encap |
| ⚠ | ☐ | configuration-failed | 2018-12-12T08:58:3... | topology/pod-1/node-102/local/svc-policyelem-id-0/uni/epp/fv-[uni/tn-Studienarbeit/ap-ltb/epg-100002_2_CE1_2_2_Client1_1]/node-102/stpathatt-[eth1/41]/nwissues | Fault delegate: Configuration failed for uni/tn-Studienarbeit/ap-ltb/epg-100002_2_CE1_2_2_Client1_1 node 102 eth1/41 due to Encap |

Abbildung 37: APIC Problem mit physical port VLAN attachment in EPG

Auf der APIC gibt es nach wie vor ein Problem, bei dem die physischen Ports auf der ACI nicht sauber in ein VLAN im EPG eingepackt werden können. Das führt dazu, dass die Devices nicht durch das EPG miteinander kommunizieren können. Das Problem ist ACI spezifisch.

### oVirt

Das Konfigurieren der Netzwerke auf oVirt verursachte häufig Schwierigkeiten, da es sehr lange dauern kann, bis die virtuellen Netze auf alle Hosts konfiguriert wurden. Wir haben nach langem Tüfteln und Versuchen schlussendlich herausgefunden, dass es da eine Einschränkung gibt, wenn die Netzwerke via Virtual Desktop and Server Manager (VDSM) Netzwerksetup[52] konfiguriert werden.

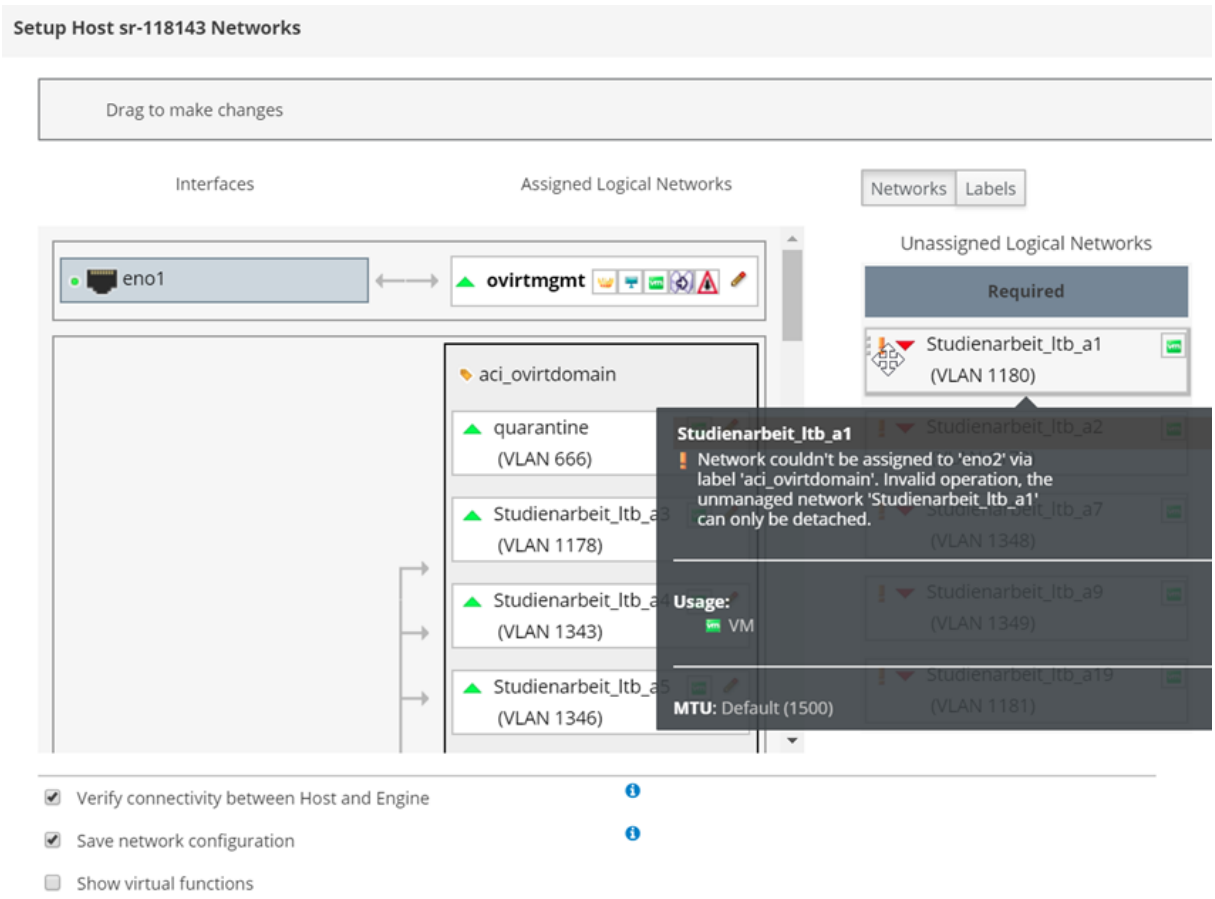



Abbildung 38: oVirt Problem mit dem Netzwerkattachment via Label

Beim automatischen Setzen der virtuellen Netzwerke auf den Hosts via Label kommt es häufig vor, dass diese nicht richtig zugewiesen werden können. Der Grund, weshalb dieser Fehler auftritt ist bis am Schluss unbekannt.

### Abklärungen

Es wurden zahlreiche Versuche gestartet, dem Problem auf den Grund zu gehen:

- Es wurde via Urs Baumann eine Mail an *Rolf Schärer* von Cisco Schweiz gemacht, der sich sehr gut mit dem Thema ACI auskennt (siehe Rolf Schärer - Cisco, Seite 89)
- Hilfe auf Bugzilla von oVirt gesucht [46]
- Hilfe im IRC Chat von oVirt gesucht [43]

Leider waren alle Massnahmen ohne Erfolg und wir haben keine Antworten auf unsere Fragen erhalten.

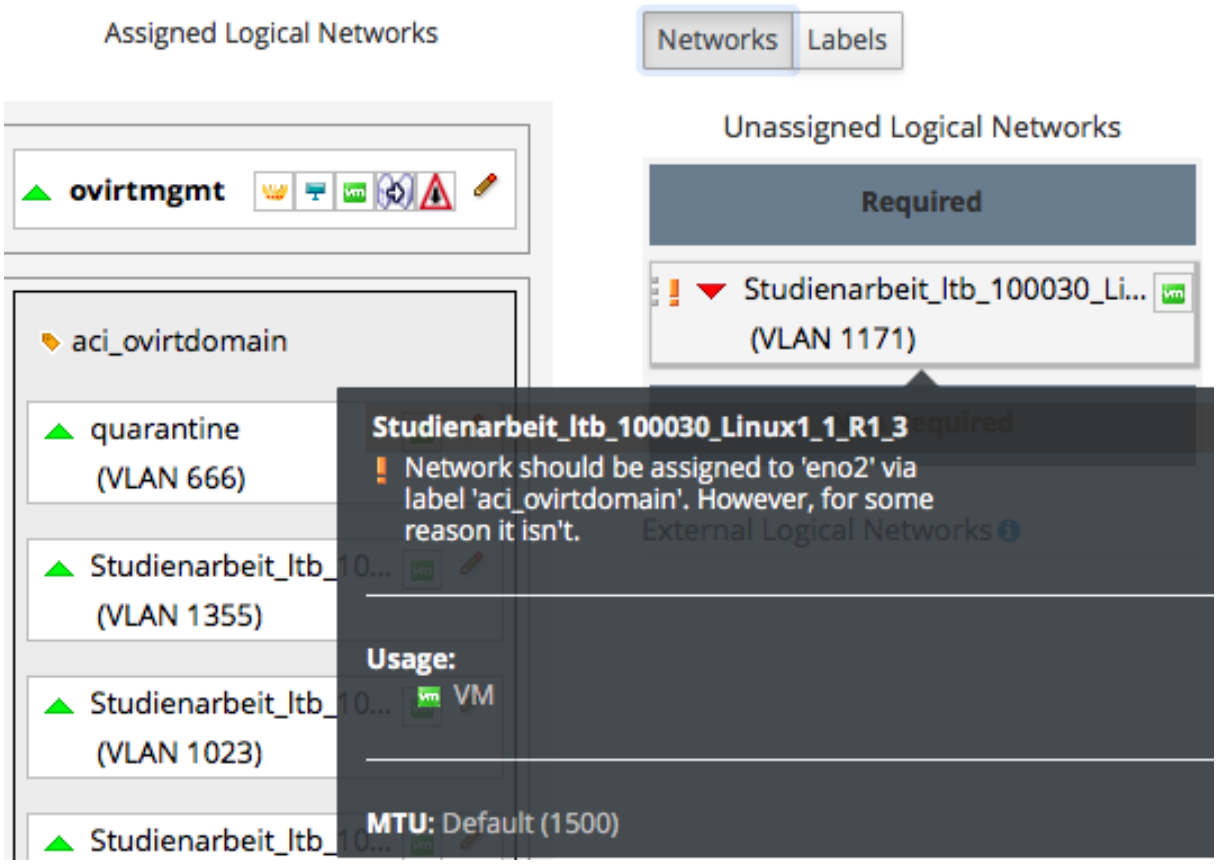



Abbildung 39: oVirt Problem mit dem Netzwerkattachment via Labels

Vermutlich aus denselben Gründen, aber mit einer anderen Fehlermeldung, werden die Netze nicht sauber auf die Hosts konfiguriert.

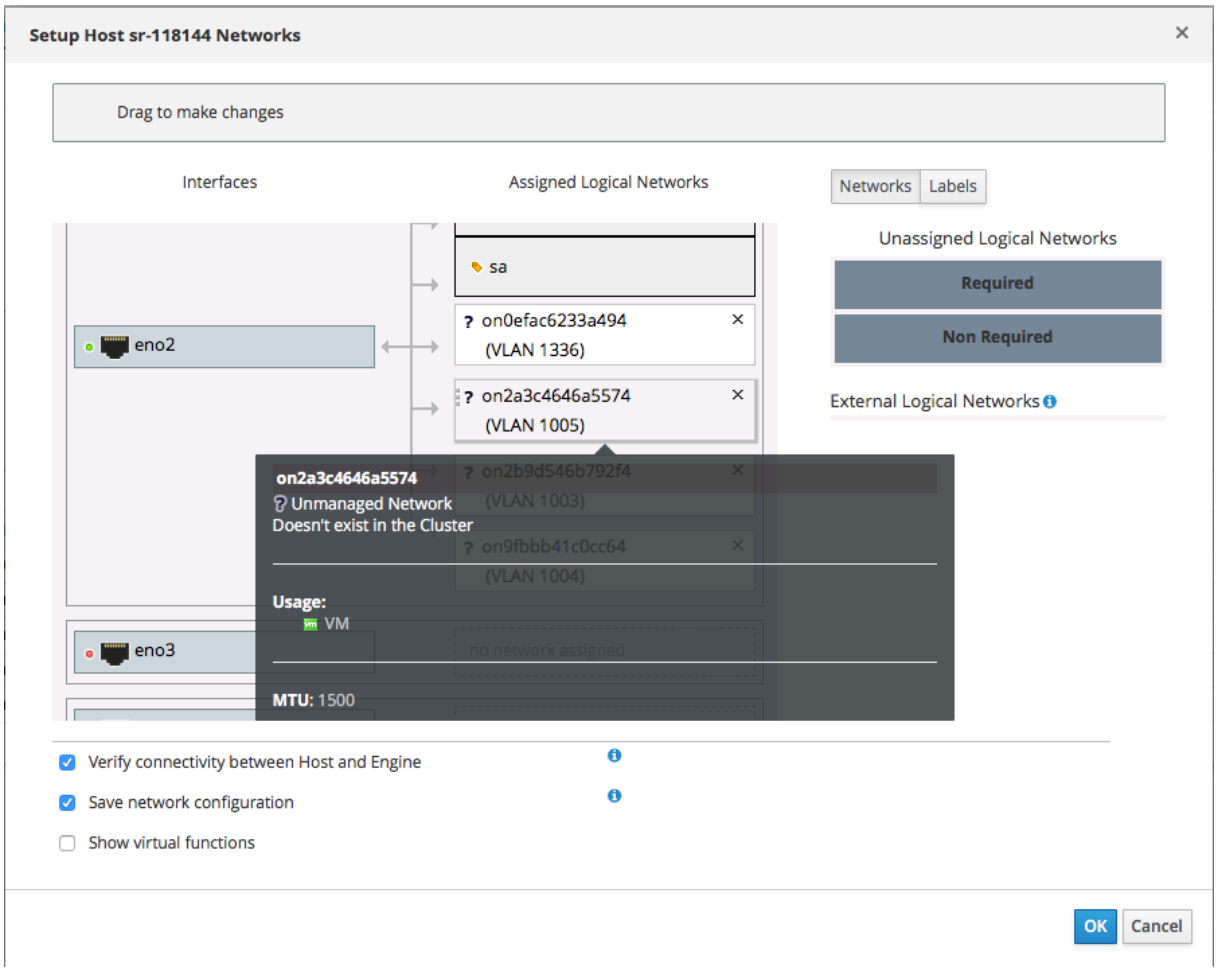



Abbildung 40: oVirt Problem mit dem Netzwerkdetachment

Dasselbe Problem kann manchmal auch auftreten, wenn die Netzwerke wieder von den Hosts automatisch via Label gelöscht werden sollen. Es kommt dann vor, dass das Netzwerk gelöscht wird, dieses aber noch auf den Hosts als unbekanntes Netzwerk angezeigt werden. Diese müssen von Hand (oder automatisiert mit der Orchestration) wieder entfernt werden, damit oVirt weiterhin ordnungsgemäss funktioniert. Werden diese Netzwerke nicht von den Hosts gelöscht, kann es vorkommen, dass keine weiteren Netze mehr auf die Hosts deployt werden können. Es kann dann zu Abbrüchen in dem Lab Deployment führen.

| | Time | Message |
|---|---|---|
| ✖ | Nov 21, 2018, 1:53:24 PM | Failed to run VM 100000_R1 due to a failed validation: [Cannot run VM. There is no host that satisfies cur... |
| ✖ | Nov 21, 2018, 1:51:20 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_Linux1_1_R1_3 on host sr-118144... |
| ✖ | Nov 21, 2018, 1:51:14 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_R1_2_R3_2 on host sr-118144. (Us... |
| ✖ | Nov 21, 2018, 1:51:08 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_R3_1_R4_1 on host sr-118144. (Us... |
| ✖ | Nov 21, 2018, 1:51:02 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_R1_1_R2_1 on host sr-118144. (Us... |
| ✖ | Nov 21, 2018, 1:51:02 PM | VDSM sr-118144 command SetSafeNetworkConfigVDS failed: Resource unavailable |
| ✖ | Nov 21, 2018, 1:51:02 PM | (1/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_Linux1_1_R1_3 on host sr-118143... |
| ✖ | Nov 21, 2018, 1:51:02 PM | (1/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_R2_2_R4_2 on host sr-118143. (Us... |
| ✖ | Nov 21, 2018, 1:51:02 PM | (1/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_R1_2_R3_2 on host sr-118143. (Us... |
| ✖ | Nov 21, 2018, 1:51:02 PM | (1/2): Failed to apply changes for network(s) Studienarbeit_ltb_100000_R3_1_R4_1 on host sr-118143. (Us... |
| ✖ | Nov 21, 2018, 1:51:02 PM | VDSM sr-118143 command HostSetupNetworksVDS failed: Resource unavailable |
| ✖ | Nov 21, 2018, 1:20:06 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_100002_R1_1_R2_1 on host sr-118143. (Us... |
| ✖ | Nov 21, 2018, 1:19:57 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_100002_R2_2_R4_2 on host sr-118143. (Us... |
| ✖ | Nov 21, 2018, 1:19:47 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_100002_R1_2_R3_2 on host sr-118143. (Us... |
| ✖ | Nov 21, 2018, 1:19:47 PM | VDSM sr-118143 command SetSafeNetworkConfigVDS failed: Resource unavailable |
| ✖ | Nov 21, 2018, 1:19:38 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_mybd13 on host sr-118143. (User: admin... |
| ✖ | Nov 21, 2018, 1:19:28 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_mybd19 on host sr-118143. (User: admin... |
| ✖ | Nov 21, 2018, 1:19:19 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_mybd15 on host sr-118143. (User: admin... |
| ✖ | Nov 21, 2018, 1:19:09 PM | (2/2): Failed to apply changes for network(s) Studienarbeit_ltb_mybd16 on host sr-118143. (User: admin... |
| ✖ | Nov 21, 2018, 1:19:09 PM | VDSM sr-118143 command SetSafeNetworkConfigVDS failed: Resource unavailable |

Abbildung 41: oVirt Problem mit dem Netzwerkattachment via Label im Error-log

In der oVirt Engine im Log werden diese Fehler ebenfalls dokumentiert. Es ist aber leider nicht klar ersichtlich, weshalb der Fehler auftritt.

### Docker

In der produktiven Umgebung wurde von Docker der DNS-Server nicht richtig mit in den Container übernommen. Nach einer Recherche im Internet[63] ist herausgekommen, dass es sich um ein bekanntes Problem handelt. In der Datei `/etc/docker/daemon.json` müssen die DNS-Server des Hosts mitgegeben werden:

```
{
  "DNS": ["152.96.120.54", "127.0.0.53"]
}
```

Die Server lassen sich einfach herausfinden mit dem Befehl `less /etc/resolv.conf`.

## 8.4 Nächste Schritte

### 8.4.1 Erweiterungsmöglichkeiten

Folgende Erweiterungsmöglichkeiten sind vorstellbar:

- Das User Interface (Frontend) kann weit ausgebaut werden, sodass zum Beispiel das Lab darüber erstellt werden kann und nicht mehr von Hand geschrieben werden muss. Die möglichen Komponenten können aus dem Backend abgefragt werden.
- Der Lab Topology Template Importer müsste einen Check einbauen, der überprüft, ob das importierte Template gültig ist.
- Damit das UI nicht blockiert ist, soll das Deployment in eine Task-Queue ausgelagert werden. Dies kann z.B. mittels «Python Celery» [9] gemacht werden. Der Fortschritt könnte somit über das Frontend abgefragt und angezeigt werden.
- Ein Reservationssystem ermöglicht das Reservieren eines Labs im Voraus für einen bestimmten Zeitpunkt. Vor allem die physischen Ressourcen müssen dann sicher reserviert sein.
- Ein Abrechnungssystem könnte die genutzten Ressourcen zusammenrechnen und in Rechnung stellen.
- Ein Sicherheitskonzept stellt sicher, dass die Labs nicht durch unbefugte Personen zugreifbar sind. Dazu gehören die virtuellen Maschinen, aber auch die physische Hardware via serieller Kommunikation.
- Der Cisco iOS XRv Router [14] und weitere Devicetypen müssen ebenfalls noch – wie der Cisco CSR1000v – als VM-Template integriert werden.

### 8.4.2 Substitutionsmöglichkeiten

#### oVirt Virtualization

Wir sehen oVirt als eine ungeeignete Virtualisierungslösung, da das Deployment der Netzwerke zu langsam ist und deshalb viele Probleme verursacht werden (siehe Probleme, Seite 72).
Mögliche Alternativen sollen genauer evaluiert werden (siehe Virtualization - Virtual Device Manager, Seite 48).

#### Nornir

Nornir funktioniert ganz gut mit den dynamisch erstellten Inventories und den geschriebenen Workflows. Jedoch ist es relativ formlos und wird schnell komplex, wenn die Workflows zu wachsen beginnen. Allenfalls macht es Sinn, die übergeordneten Workflows mittels einem anderen Tool zu Orchestrieren, aber innerhalb eines Workflows die Tasks noch immer parallelisiert mittels Nornir zu verwalten.

# 9 Anhang

## 9.1 Constraints

Für die Studienarbeit sind pro Person 240 Arbeitsstunden (8 ECTS Credits à 30h) vorgesehen. Diese beinhalten neben der Analyse in diesem Dokument auch die beschriebene Umsetzung. Aus Zeitgründen ist es aber nicht möglich, die ganze analysierte Umgebung zu implementieren. Deshalb wurde ein zentraler Teil ausgewählt, welcher dann auch umgesetzt werden konnte.

## 9.2 Projektplanung

### 9.2.1 Projektorganisation

### 9.2.2 Projektphasenplanung

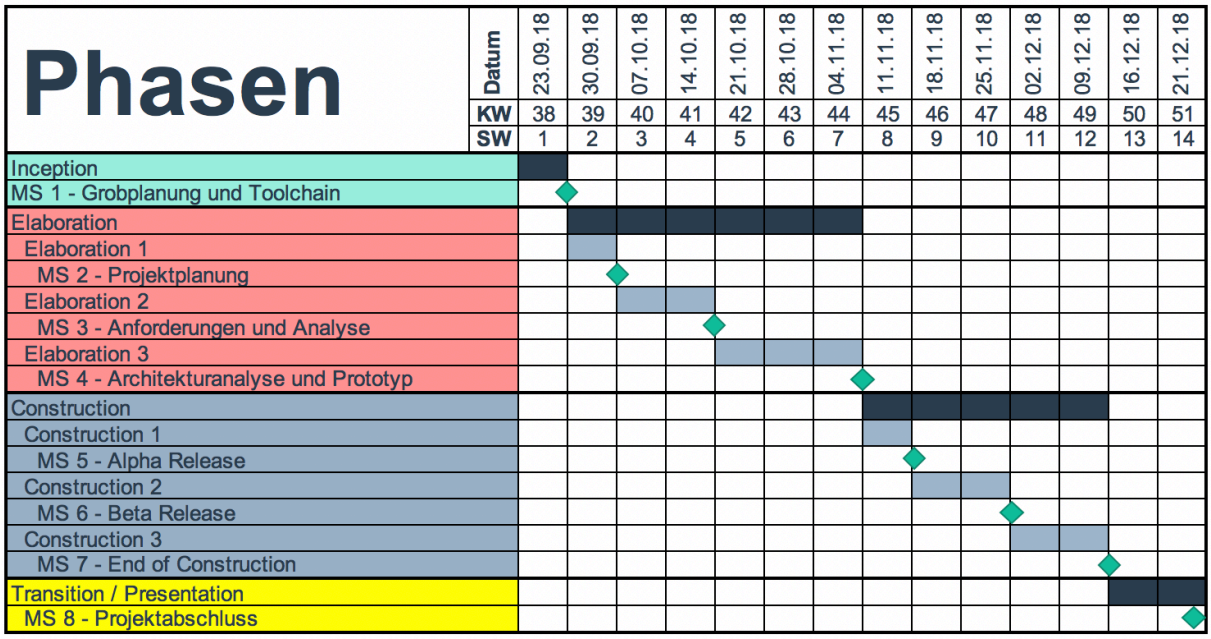



Abbildung 42: Projektphasenplanung Gantt

#### Beschreibung der einzelnen Phasen

#### Inception

Die Inception dauert eine Woche und beinhaltet den ersten Meilenstein.
Darin beinhaltet sind die ersten Abklärungen und Besprechungen, sowie das Aufsetzen der Toolchain und eine Grobplanung der Arbeit. Beim Erreichen des Meilensteins soll alles soweit bereit sein, dass mit der Planung begonnen werden kann.

#### Elaboration

Die Elaboration-Phase dauert sechs Wochen und beinhaltet drei Meilensteine.
Darin enthalten sind die detaillierte Projektplanung, die Analyse an die Software und ein entsprechender Prototyp. Diese Phase ist sehr essentiell, da die darauffolgende Umsetzung stark davon abhängt.
Die End of Elaboration stellt den Übergang von Testing / Prototyping zur Produktivität dar. Zudem steht die Tool-Chain für das produktive Arbeiten zur Verfügung. Die Risiken für die Construction-Phase sollen, wo möglich, alle behoben sein. Eine Grobplanung der Construction-Phase ist hier bereits erstellt.

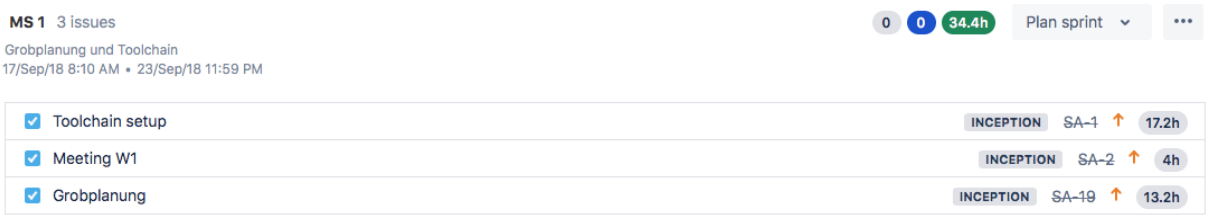
#### Construction
Die Construction-Phase dauert fünf Wochen und beinhaltet drei Meilensteine.
Hier wird das Produkt entwickelt und getestet.
Eine detaillierte Planung wird in einem separaten Kapitel behandelt (9.2.4).

#### Transition
Die Transition dauert zwei Wochen und beinhaltet den letzten Meilenstein.
Diese Zeit wird dazu genutzt, die Präsentation, Dokumentation, Abstract und das Poster fertigzustellen und abzuschliessen und die weiteren Schritte im Vorgehen nach der Arbeit zu besprechen.
Diese Phase endet mit der Abgabe der Arbeit an die Betreuenden.

### 9.2.3 Arbeitspakete
Die Verwaltung der Arbeitspakete wird live auf Jira abgewickelt. Diese Pakete können dort direkt eingesehen werden (https://labbuilder.atlassian.net/).

Die Arbeitspakete werden auf Jira erfasst und die dafür benötigte Zeit wird geschätzt. Zusätzlich werden diese den Meilensteinen und einer Version zugeordnet. In der Construction-Phase werden die priorisierten Tasks in eine Iteration aufgenommen und abgearbeitet. Eine Iteration während der Construction-Phase erhält jeweils einen Sprint, in welchem der Backlog geschätzt, priorisiert und dann dem Sprint zugewiesen wird.
Im Scrum-Board ist dann die Übersicht, welche Pakete welchen Status besitzen, und wem die verschiedenen Arbeitspakete zugewiesen sind. Nebenbei werden die benötigten Zeiten direkt auf die Arbeitspakete verbucht, welche dann auch für die Zeitauswertung verwendet werden.

### 9.2.4 Planung der Sprints
Die Planung der Sprints erfolgte in Jira. Dabei wurden die Arbeitspakete (Tasks / Stories) jeweils zu Beginn des jeweiligen Sprints definiert, geschätzt und nach Priorität sortiert. Dies ist im Folgenden für alle Sprints ersichtlich.

#### Inception


## Elaboration

**MS 2** 7 issues — 0 0 34.4h — Plan sprint

Projektplanung
24/Sep/18 11:58 AM • 01/Oct/18 11:58 AM

| | | | |
|---|---|---|---|
| Meeting W2 | ELABORATION | ~~SA-4~~ | 4h |
| Use Cases erarbeiten | ELABORATION | ~~SA-20~~ | 4h |
| Dokumentation | ELABORATION | ~~SA-21~~ | 2.4h |
| Requirements Analyse - Brainstorming | ELABORATION | ~~SA-22~~ | 4h |
| Projektplanung - brief | ELABORATION | ~~SA-23~~ | 6h |
| Sprint Planung | ELABORATION | ~~SA-24~~ | 4h |
| C4 Component und Container Identification | ELABORATION | ~~SA-25~~ | 10h |

**MS 3** 10 issues — 0 0 68.8h — Plan sprint

01/Oct/18 8:31 AM • 15/Oct/18 8:31 AM

| | | | |
|---|---|---|---|
| Sprint Planung | ELABORATION | ~~SA-34~~ | 2h |
| Virtual Device Manager Prototyping | ELABORATION | ~~SA-38~~ | 8h |
| Tools Requirements | ELABORATION | ~~SA-35~~ | 22h |
| C4 Component Refinement | ELABORATION | ~~SA-36~~ | 6h |
| PDU Evaluation | ELABORATION | ~~SA-37~~ | 4h |
| Meeting W3 | ELABORATION | ~~SA-5~~ | 4h |
| Sequenzdiagramme brief erstellen | ELABORATION | ~~SA-39~~ | 6h |
| Dokumentation | ELABORATION | ~~SA-28~~ | 4.8h |
| Meeting W4 | ELABORATION | ~~SA-6~~ | 4h |
| Projektplanung | ELABORATION | ~~SA-26~~ | 8h |

**MS 4** 9 issues — 0 0 69.3h — Plan sprint

15/Oct/18 8:21 AM • 29/Oct/18 8:21 AM

| | | | |
|---|---|---|---|
| Architektur Analyse | ELABORATION | ~~SA-40~~ | 16h |
| Sprint Planing | ELABORATION | ~~SA-41~~ | 2h |
| Prototyping | ELABORATION | ~~SA-27~~ | 26h |
| Dokumentation | ELABORATION | ~~SA-29~~ | 8.8h |
| Sequenzdiagramme ausarbeiten | ELABORATION | ~~SA-48~~ | 4h |
| Meeting W5 | ELABORATION | ~~SA-7~~ | 4h |
| Meeting W6 | ELABORATION | ~~SA-8~~ | 4h |
| Risiko Abklärung ACI | ELABORATION | ~~SA-49~~ | 0.5h |
| Meeting W7 | ELABORATION | ~~SA-9~~ | 4h |

## Construction

**MS 5** 8 issues — 0 | 0 | 38.8h — Plan sprint

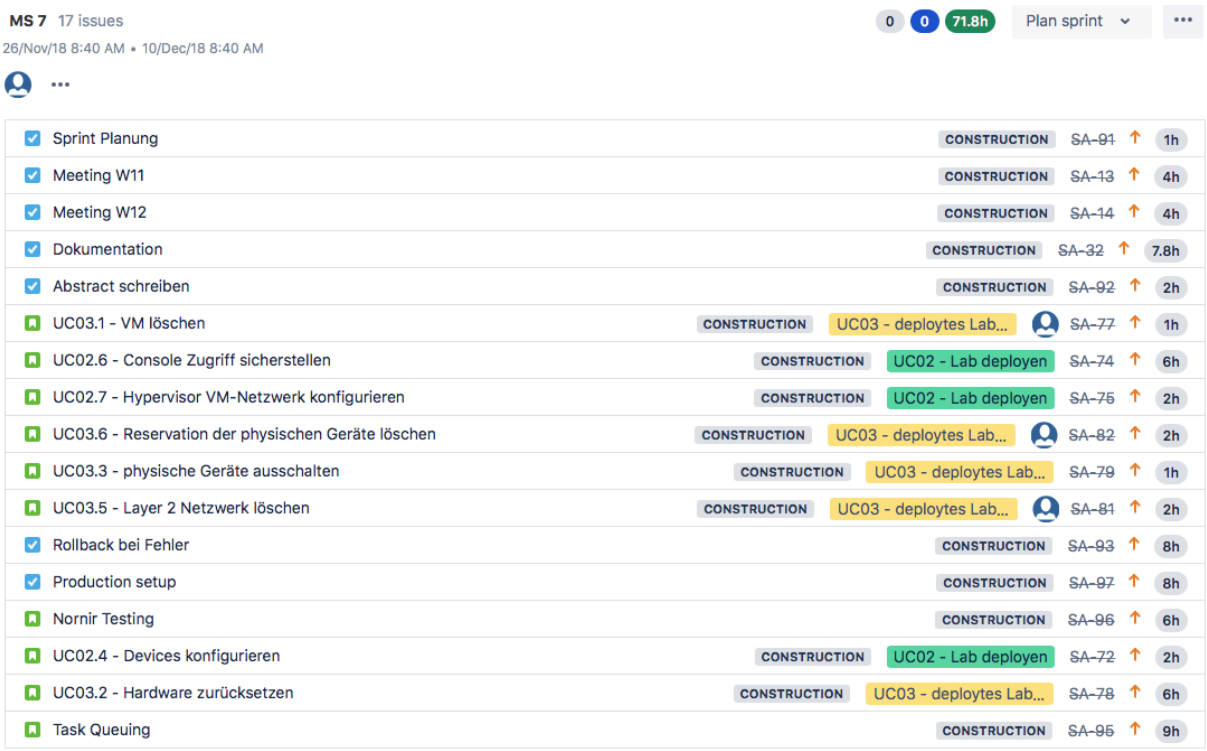
05/Nov/18 10:36 AM • 12/Nov/18 10:36 AM

| | | | | |
|---|---|---|---|---|
| Meeting W8 | CONSTRUCTION | | ~~SA-10~~ | 4h |
| Dokumentation | CONSTRUCTION | | ~~SA-30~~ | 4.8h |
| Sprint Planing | CONSTRUCTION | | ~~SA-50~~ | 4h |
| Running Lab Store | CONSTRUCTION | | ~~SA-55~~ | 4h |
| UC06 - Lab Topology Template exportieren | ELABORATION | UC01 - CRUD Lab Top... | ~~SA-65~~ | 1h |
| UC07 - Lab Topology Template importieren | ELABORATION | UC01 - CRUD Lab Top... | ~~SA-66~~ | 1h |
| UC02.1 - VM erstellen | CONSTRUCTION | UC02 - Lab deployen | ~~SA-69~~ | 8h |
| UC02.5 - Layer 2 Netzwerk konfigurieren | CONSTRUCTION | UC02 - Lab deployen | ~~SA-73~~ | 12h |

**MS 6** 15 issues — 0 | 0 | 69.8h — Plan sprint

12/Nov/18 1:19 PM • 26/Nov/18 1:19 PM

| | | | | |
|---|---|---|---|---|
| Meeting W9 | CONSTRUCTION | | ~~SA-11~~ | 6h |
| Meeting W10 | CONSTRUCTION | | ~~SA-12~~ | 4h |
| oVirt neu aufsetzen | CONSTRUCTION | | ~~SA-90~~ | 4h |
| UC02.2 - Running Lab dokumentieren | CONSTRUCTION | UC02 - Lab deployen | ~~SA-70~~ | 2h |
| Dokumentation | CONSTRUCTION | | ~~SA-31~~ | 4.8h |
| Physical Infrastructure Store aufbauen | CONSTRUCTION | UC011 - CRUD physis... | ~~SA-87~~ | 1h |
| Physical Infrastructure Store mit aktueller Hardware und Topologie befüllen | CONSTRUCTION | UC011 - CRUD physis... | ~~SA-88~~ | 4h |
| Physical Infrastructure Store - Anbindung an Orchestration Tool | CONSTRUCTION | UC011 - CRUD physis... | ~~SA-89~~ | 2h |
| UC13 - Verfügbarkeit von Ressourcen prüfen | CONSTRUCTION | UC02 - Lab deployen | ~~SA-68~~ | 2h |
| UC02.6 - Console Zugriff sicherstellen | CONSTRUCTION | UC02 - Lab deployen | ~~SA-74~~ | 20h |
| UC02.7 - Hypervisor VM-Netzwerk konfigurieren | CONSTRUCTION | UC02 - Lab deployen | ~~SA-75~~ | 11h |
| UC03.6 - Reservation der physischen Geräte löschen | CONSTRUCTION | UC03 - deployes Lab... | ~~SA-82~~ | 2h |
| UC02.8 - physische Geräte als in Gebrauch reservieren | CONSTRUCTION | UC02 - Lab deployen | ~~SA-76~~ | 2h |
| UC02.3 - physische Geräte einschalten | CONSTRUCTION | UC02 - Lab deployen | ~~SA-71~~ | 4h |
| UC03.4 - Running Lab löschen | CONSTRUCTION | UC03 - deployes Lab... | ~~SA-80~~ | 1h |



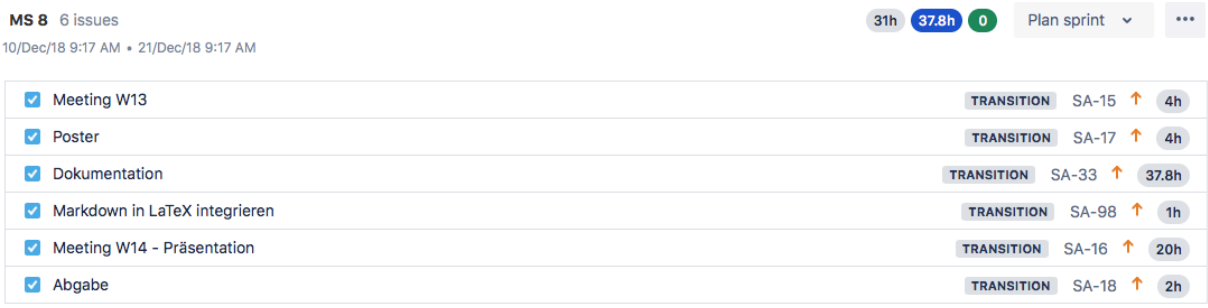
**MS 7** 17 issues

26/Nov/18 8:40 AM • 10/Dec/18 8:40 AM

0 0 71.8h Plan sprint

| | | | | |
|---|---|---|---|---|
| Sprint Planung | CONSTRUCTION | | ~~SA-91~~ | 1h |
| Meeting W11 | CONSTRUCTION | | ~~SA-13~~ | 4h |
| Meeting W12 | CONSTRUCTION | | ~~SA-14~~ | 4h |
| Dokumentation | CONSTRUCTION | | ~~SA-32~~ | 7.8h |
| Abstract schreiben | CONSTRUCTION | | ~~SA-92~~ | 2h |
| UC03.1 - VM löschen | CONSTRUCTION | UC03 - deployes Lab... | ~~SA-77~~ | 1h |
| UC02.6 - Console Zugriff sicherstellen | CONSTRUCTION | UC02 - Lab deployen | ~~SA-74~~ | 6h |
| UC02.7 - Hypervisor VM-Netzwerk konfigurieren | CONSTRUCTION | UC02 - Lab deployen | ~~SA-75~~ | 2h |
| UC03.6 - Reservation der physischen Geräte löschen | CONSTRUCTION | UC03 - deployes Lab... | ~~SA-82~~ | 2h |
| UC03.3 - physische Geräte ausschalten | CONSTRUCTION | UC03 - deployes Lab... | ~~SA-79~~ | 1h |
| UC03.5 - Layer 2 Netzwerk löschen | CONSTRUCTION | UC03 - deployes Lab... | ~~SA-81~~ | 2h |
| Rollback bei Fehler | CONSTRUCTION | | ~~SA-93~~ | 8h |
| Production setup | CONSTRUCTION | | ~~SA-97~~ | 8h |
| Nornir Testing | CONSTRUCTION | | ~~SA-96~~ | 6h |
| UC02.4 - Devices konfigurieren | CONSTRUCTION | UC02 - Lab deployen | ~~SA-72~~ | 2h |
| UC03.2 - Hardware zurücksetzen | CONSTRUCTION | UC03 - deployes Lab... | ~~SA-78~~ | 6h |
| Task Queuing | CONSTRUCTION | | ~~SA-95~~ | 9h |

## Transition



**MS 8** 6 issues

10/Dec/18 9:17 AM • 21/Dec/18 9:17 AM

31h 37.8h 0 Plan sprint

| | | | |
|---|---|---|---|
| Meeting W13 | TRANSITION | SA-15 | 4h |
| Poster | TRANSITION | SA-17 | 4h |
| Dokumentation | TRANSITION | SA-33 | 37.8h |
| Markdown in LaTeX integrieren | TRANSITION | SA-98 | 1h |
| Meeting W14 - Präsentation | TRANSITION | SA-16 | 20h |
| Abgabe | TRANSITION | SA-18 | 2h |

## Backlog

Die optionalen Anforderungen wurden teilweise mit eingeplant aber nicht mehr vollständig umgesetzt.

**Backlog** 7 issues

Create sprint

| | | | | |
|---|---|---|---|---|
| UC02.4 - Devices konfigurieren | CONSTRUCTION | UC02 - Lab deployen | SA-72 | 8h |
| UC03.2 - Hardware zurücksetzen | CONSTRUCTION | UC03 - deployes Lab... | SA-78 | 6h |
| Task Queuing | CONSTRUCTION | | SA-95 | 10h |
| Lab Topology Template generieren und erstellen | CONSTRUCTION | UC04 - Snapshot eine... | SA-83 | - |
| Running Configs der virtuellen Devices exportieren | CONSTRUCTION | UC04 - Snapshot eine... | SA-84 | - |
| Running Configs der physischen Network Devices exportieren | CONSTRUCTION | UC04 - Snapshot eine... | SA-85 | - |
| Snapshot der VMs erstellen und als VM-Type-Template speichern | CONSTRUCTION | UC04 - Snapshot eine... | SA-86 | - |

## 9.3 Projecttracing

### 9.3.1 Zeitauswertung

Aufgrund der geloggten Zeiten in Jira konnten folgende Graphen erstellt werden, welche die Arbeitseinteilung und Aufteilung aufzeigen.

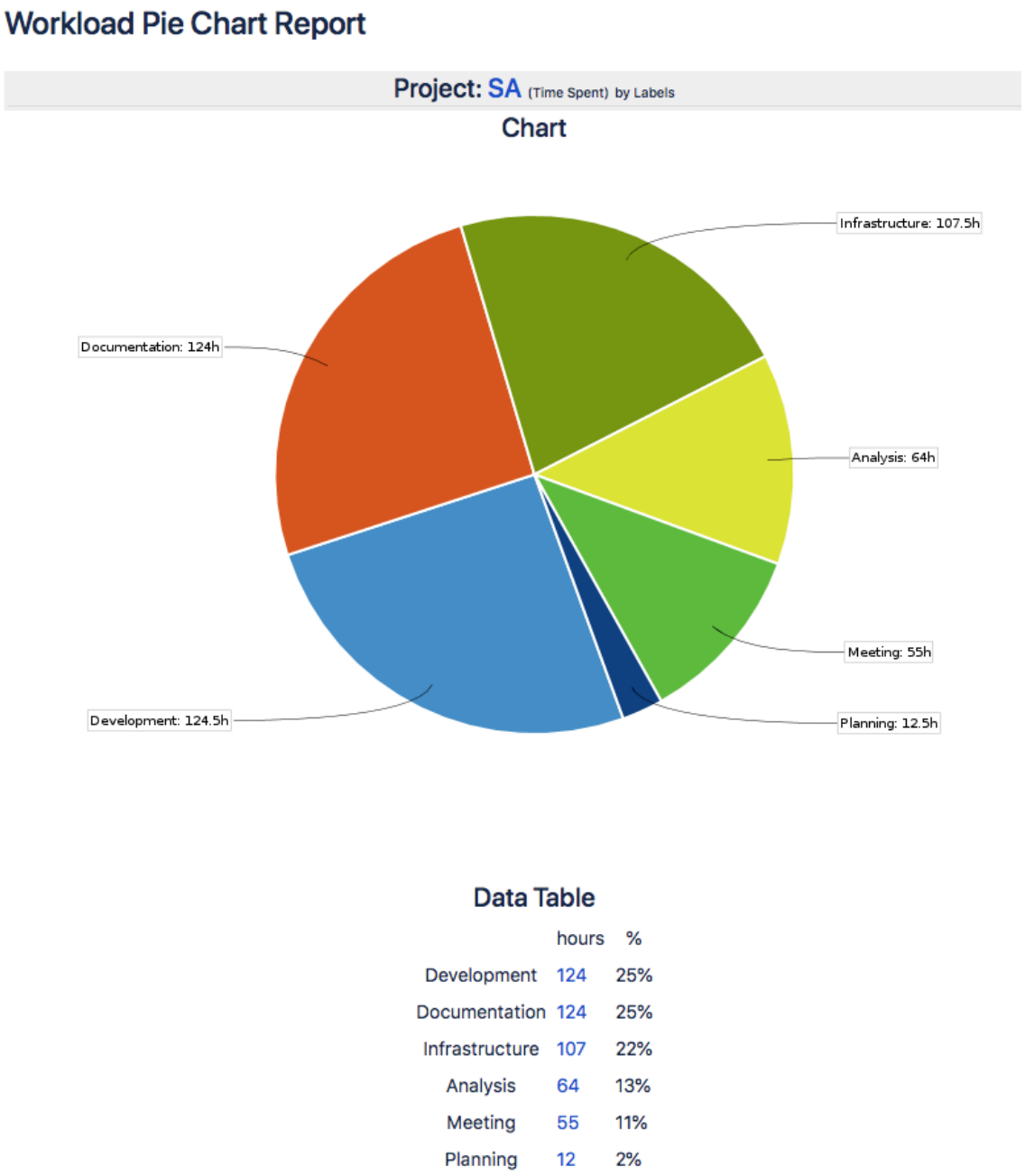



Abbildung 43: Aufteilung der Arbeitsstunden auf die Jira Labels

Hier sind die Stunden und die prozentuale Aufteilung der Labels ersichtlich. Die Labels wurden auf die Arbeitspakete im Jira gesetzt. Die wichtigsten Labels sind Entwicklung (Development), Dokumentation, Infrastrukturarbeiten und die Architekturanalyse.
Auffällig ist hierbei der grosse Infrastruktur-Teil. Dort haben wir einen grossen Teil der Zeit in die Fehleranalyse der Netzwerkproblematik (siehe Probleme, Seite 72ff) in oVirt hineingesteckt

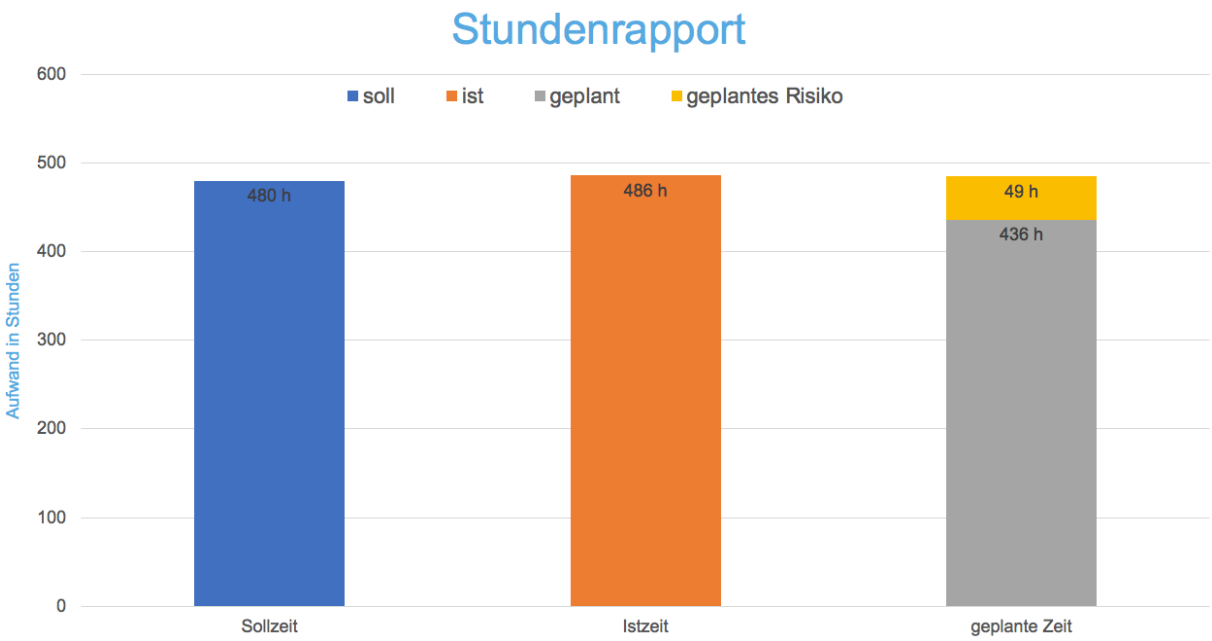



Abbildung 44: Gesamtarbeitsstunden der Studienarbeit

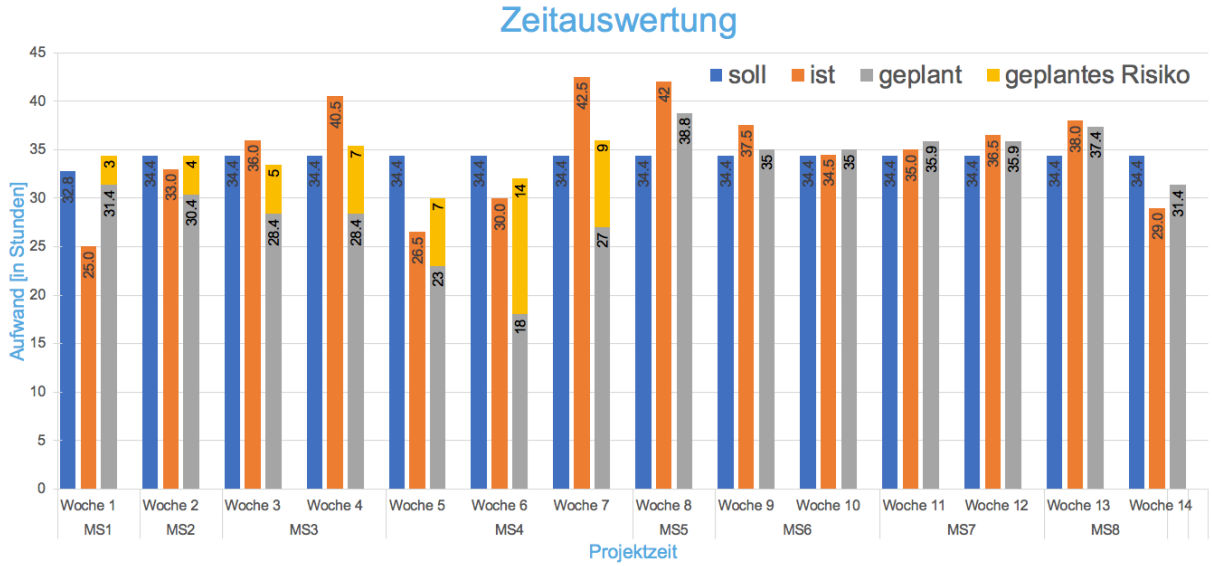



Abbildung 45: Einteilung der Arbeitsstunden auf die Semesterwochen

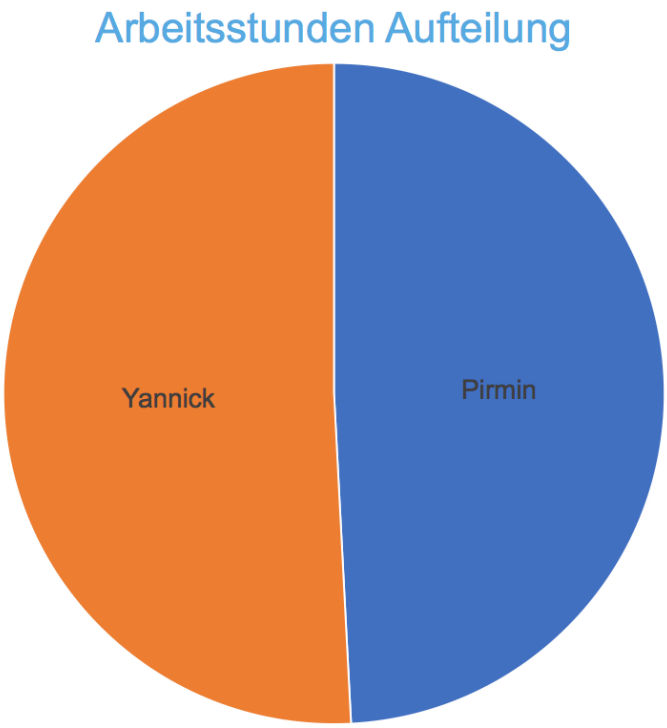



Abbildung 46: Prozentuelle Arbeitsstundenaufteilung auf die Gruppenmitglieder

### 9.3.2 Metriken

```
github.com/AlDanial/cloc v 1.78  T=0.30 s (401.6 files/s, 65543.7 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JSON                            13              0              0          14475
Python                          40            451             86           1752
Vuejs Component                 10             33              1            627
YAML                            25             34            279            592
Markdown                         8            153              0            424
Bourne Shell                     7             64             42            315
JavaScript                      11              8             10            129
Dockerfile                       5             44              7             83
HTML                             1              0              0             18
TOML                             1             21             86             13
-------------------------------------------------------------------------------
SUM:                           121            808            511          18428
-------------------------------------------------------------------------------
```

Abbildung 47: Lines of Code im Projekt (mittels cloc [1] ermittelt)

### 9.3.3 Zielerreichung

Die Analyse der Architektur wurde wie geplant abgeschlossen.
Bei der Umsetzung wurden die erforderlichen Use Cases (grün) gemäss Use Case Diagramm umgesetzt. Eine grosse Hilfe dabei war die Zeitplanung der Arbeitspakete im Jira, welche uns eine ständige Übersicht über die Stunden und die Arbeiten gab.

#### Optionale Features

Die optionalen Features vom Use Case Diagramm (rot, blau) wurden aus Zeitgründen grösstenteils nur theoretisch angeschaut, nicht aber praktisch umgesetzt.

#### Projekt in Zahlen

**7** unterschiedliche Docker Container
**14** Wochen
**114** Arbeitspakete in Jira
**132** Commits bei der praktischen Umsetzung
**486** Stunden Arbeit
**3529** Lines of Code (siehe Metriken, Seite 85)

### 9.3.4 Fazit

Die Studienarbeit war ein interessantes Aufgabengebiet, welches uns beiden Freude und Vergnügen bei der Arbeit bereitete. Wir konnten viel von dem Erlernten im Studium wieder hervorholen und gebrauchen, aber trotzdem auch noch sehr viel Neues dazulernen.
Aufgrund der Probleme, die wir hatten, kam es oft zu Verzögerungen bei dem Arbeiten. Vor das grösste Problem stellte uns die oVirt Virtualization, da sich dieses Produkt nicht so verhalten hat, wie wir uns das vorgestellt hatten und wie es sich in unseren Tests während der Evaluierung gezeigt hat. Das hat uns viele Stunden gekostet, den Fehler einzugrenzen und zu umgehen.
Andere Software, wie Flask mit Flask-RESTPlus und Nornir hingegen haben uns positiv überrascht.

Mit Nornir haben wir trotz der Beta Version sehr gute Erfahrungen gemacht. Im Nachhinein muss man sich überlegen, wie man die Tasks und Workflows noch etwas generischer und strukturierter verwalten bzw. ablegen könnte, damit man den Überblick nicht verliert. Ansonsten hat sich dieses Framework sehr gut bewährt.

#### Arbeitsaufteilung

Das Arbeiten im Team war sehr ausgeglichen und beide Parteien konnten ihren Teil zur Arbeit beitragen.

## 9.4 Berechnungen

### 9.4.1 oVirt Speicherplatz Verbrauch

Die Daten basieren auf folgenden Auszügen aus dem oVirt und dem NFS Share Server.

Das Base Image (Cisco CSR1000V IOS XE UNIVERSAL - CRYPTO Serial QCOW2 [12]) mit der ID `fad47f02-44aa-4d32-ace8-9f574ce81144` hat eine Grösse von 1.9 GB auf dem NFS Share.

Das Image mit der ID `f1060327-21dd-4e66-ab1e-ab93c5d63b3d` benötigt lediglich 41 MB Speicherplatz. Jedoch wird dazu einmalig das Base Image benötigt. Alle weiteren Images basieren auf dem Base Image und sind deshalb nur ca. 41 MB gross. Wie dem NFS Share Output zu entnehmen ist, gibt es noch eine geclonte VM, deren Image ebenfalls 1.9 GB gross ist (`6794c253-0f4d-4990-a63c-b16f91b849ae`).

| Alias | ID | | | Attached To | Virtual Size | Status | Type |
|---|---|---|---|---|---|---|---|
| csr1000v | 3f75d1f9-da70-42cb-9eac-ed350b8df3a5 | | | 100001_R2 | 8 GiB | OK | Image |
| csr1000v | c648eea7-1040-4f74-aaa4-1810abdc63a0 | | | 100002_1_R1 | 8 GiB | OK | Image |
| csr1000v | 6794c253-0f4d-4990-a63c-b16f91b849ae | | | csr1000v | 8 GiB | OK | Image |
| csr1000v | 664623e5-8e87-426b-a7fd-1b29c60459d9 | | | 100002_2_PE2 | 8 GiB | OK | Image |
| csr1000v | 2fd2be22-ddfd-4a8b-8a02-d84d01ef1624 | | | 100002_2_R1 | 8 GiB | OK | Image |
| csr1000v | 5b441d89-ccd3-4661-80ed-50bb12601d87 | | | 100002_2_Client1 | 8 GiB | OK | Image |
| csr1000v | c8753d61-9041-453b-9d9d-c0a7a6332736 | | | 100002_2_R2 | 8 GiB | OK | Image |
| csr1000v | a0d5d4bd-e4b4-444d-b339-6a81071f44cb | | | 100002_1_PE1 | 8 GiB | OK | Image |
| csr1000v | 0e811f00-e04f-4245-b9f2-13929d12867a | | | 100001_R1 | 8 GiB | OK | Image |
| csr1000v | 00630fa2-c495-4573-80e8-f1829c095d23 | | | 100002_2_PE1 | 8 GiB | OK | Image |
| csr1000v | 9cec5634-7d2f-4704-af84-eed93c831b80 | | | 100002_1_Client1 | 8 GiB | OK | Image |
| csr1000v | f1060327-21dd-4e66-ab1e-ab93c5d63b3d | | | 100002_1_PE2 | 8 GiB | OK | Image |
| csr1000v | fad47f02-44aa-4d32-ace8-9f574ce81144 | | | csr1000v (base version) | 8 GiB | OK | Image |

Abbildung 48: oVirt Storage Domain Disks

```
[root@sr-000027 images]# du . -h
1.9G ./6794c253-0f4d-4990-a63c-b16f91b849ae
1.2M ./78099215-6de3-4d89-9247-7d39f6b04031
1.2M ./54a2e56e-7885-447f-bac7-96b44cbbe925
1.9G ./fad47f02-44aa-4d32-ace8-9f574ce81144
41M ./0e811f00-e04f-4245-b9f2-13929d12867a
42M ./3f75d1f9-da70-42cb-9eac-ed350b8df3a5
41M ./c648eea7-1040-4f74-aaa4-1810abdc63a0
41M ./f1060327-21dd-4e66-ab1e-ab93c5d63b3d
41M ./00630fa2-c495-4573-80e8-f1829c095d23
42M ./5b441d89-ccd3-4661-80ed-50bb12601d87
41M ./a0d5d4bd-e4b4-444d-b339-6a81071f44cb
41M ./b1a9a2a4-cc2b-4223-b3a8-973f54b01978
41M ./664623e5-8e87-426b-a7fd-1b29c60459d9
41M ./9cec5634-7d2f-4704-af84-eed93c831b80
41M ./c8753d61-9041-453b-9d9d-c0a7a6332736
41M ./2fd2be22-ddfd-4a8b-8a02-d84d01ef1624
4.2G .
```

## 9.5 API Documentation

### 9.5.1 PDU

Als PDU haben wir sechs NetIO 4C erhalten. Jede dieser PDU besitzt vier Power Outlets, die separat angesteuert werden können.
Nach der Inbetriebnahme und der IP-Adresszuweisung muss zuerst die JSON Machine to Machine (M2M) API aktiviert und mit einem Passwort versehen werden.

Die PDUs unterstützen als Autorisierung das «BasicAuth» Protokoll, welches durch TLS im HTTPS-Protokoll gesichert ist.

Mit folgendem HTTP POST an die URL https://<IP-Address>/netio.json kann ein Port geschaltet werden:

```
{
  "Outputs": [
    {
      "ID": 1,
      "Action": 0
    }
  ]
}
```

Hierbei sind folgende Parameter nennenswert: [37]

| Key | Value |
|---|---|
| ID | [1-4] Definiert den Power-output Port |
| Action | [0-4] Definiert die Action<br>• 0: Turn output Off<br>• 1: Turn output On<br>• 2: Short Off (Cold Boot)<br>• 3: Short On (An für bestimmte Zeitdauer)<br>• 4: Zustand Toggle (Zustand invertieren) |
| Delay [ms] | Definiert die On oder Off Zeit bei Action 2 und 3 |
| State | [0-1] (Nur im GET) Zeigt den aktuellen Schaltzustand an<br>• 0: Der State ist Off<br>• 1: Der State ist On |

## 9.6 Weitere Abklärungen

### 9.6.1 E-Mail

**Rolf Schärer - Cisco**

Mail an roschaer@cisco.com

```
Hallo Rolf

Wie geht es dir? Ich habe nun doch noch eine «kleine» ACI Frage.
Hast du Erfahrung mit ACI und Red Hat Virtualization? Kann es sein das die
    ↪ Integration nicht sehr gut ist?
Cool wäre wenn Ihr ein Lab hättet und schauen könnt ob Ihr die gleichen
    ↪ Probleme habt oder wenn du eventuell eine Ansprechperson hättest.

Die ganze Problematik findest du unten im Mail.

Gruss und besten Dank für deine Hilfe.
Urs
```

```
Hoi Urs

Ausgangslage:
Cisco APIC Version: 3.2(2o)
oVirt Version: 4.2.7.5-1.el7

VMM Integration ist gemacht.
1 VRF mit einem Application Profile ist erstellt.
2 oVirt Hosts in einem Cluster, 1 oVirt-Engine

Anbei unsere Beschreibung des Problems:
Wenn mehrere EPGs (mit je einer Bridge Domain) über die API des APICs
    ↪ erstellt werden, werden diese im oVirt als Logical Networks erstellt
    ↪ und mit dem Label versehen.
Das Label ist bereits zuvor auf den beiden Hosts dem entsprechenden
    ↪ Interface zugewiesen worden.
Eigentlich sollten alle Logical Networks auf dem Interface mit diesem Label
    ↪ erstellt werden. Jedoch werden meistens nicht immer alle erstellt und
    ↪ bleiben bei den «Unassigned Logical Networks».
(Siehe Screenshot)

Getestet haben wir es wie folgt:
5 BDs/EPGs –5 Requests direkt nacheinander
5 BDs/EPGs –1 Request
20 BDs/EPGs –1 Request

Kann es sein, dass das oVirt zu langsam ist und nicht alles sauber erstellen
    ↪  kann?
Danke fürs weitere Abklären!

Gruess
Yannick
```

## 9.7 Anleitungen

### 9.7.1 Environment

Wie unter III. Config (Seite 26) beschrieben, werden die Environmentvariablen beim Start des Services mit in die Software hineingegeben und sind nicht hard-coded. Es handelt sich dabei um Einstellungen der Infrastruktur, Passwörter etc.
Folgende Environmentfiles sind von Bedeutung:

1. Das Code Repo «Labbuilder» enthält im Ordner `services` verschiedene «Environment Files». Diese müssen alle an die entsprechende Infrastruktur angepasst werden.
2. Das Code Repo «Frontend» enthält die Dateien `.env` und `.env.development`. Diese müssen ebenfalls entsprechend angepasst werden.
3. Das Code Repo «Netbox» enthält im Ordner `env` wiederum einige Dateien, die angepasst werden müssen.

### 9.7.2 Starten der Dockerumgebung

Die Umgebungsvariabeln und die *Docker-Compose* [27] Files sind pro Microservice in dem vom Code unabhängigen *labbuilder* Repository gespeichert.

#### Netbox Setup

Für das Netbox Setup verwenden wir die Docker-Compose Umgebung von ninech [40] als Basis und erweitern es mit unserem eigenen Docker-Compose File.

1. Clone https://github.com/ninech/netbox-docker.git
2. Take netbox repo
3. add the files of the netbox repo into the netbox-docker folder
4. edit the «env» files with the appropriate information
5. start the docker-containers

```
docker-compose -f docker-compose.yml -f docker-compose.ltb.yml up -d
```

#### Production Environment

First setup Netbox.

Add external network for netbox:

```
docker network create transport_traefik_netbox

# If traefik is already running:
docker service update --network-add transport_traefik_netbox traefik
```

#### Start Containers

```
docker-compose -f docker-compose.prod.yml up -d
```

#### Update Containers

```
docker-compose -f docker-compose.prod.yml pull
docker-compose -f docker-compose.prod.yml up -d
```

### Development Environment

First setup Netbox.

```
docker-compose -f docker-compose.yml pull
docker-compose -f docker-compose.yml up -d
```

## 9.7.3 Netbox

Im Netbox müssen die Daten in der richtigen Form gepflegt werden, damit die Orchestrierung damit klarkommt.

### Device Tags

Und zwar müssen die «Device» Tags auf die Lab Devices (Devices, die in einem Lab verwendet werden können) so gesetzt werden, dass sie mit der Environment Variable (siehe Environment, Seite 90) übereinstimmen (z.B. `lab_device`). Nur jene mit dem richtigen Device Tag werden für das Lab berücksichtigt.

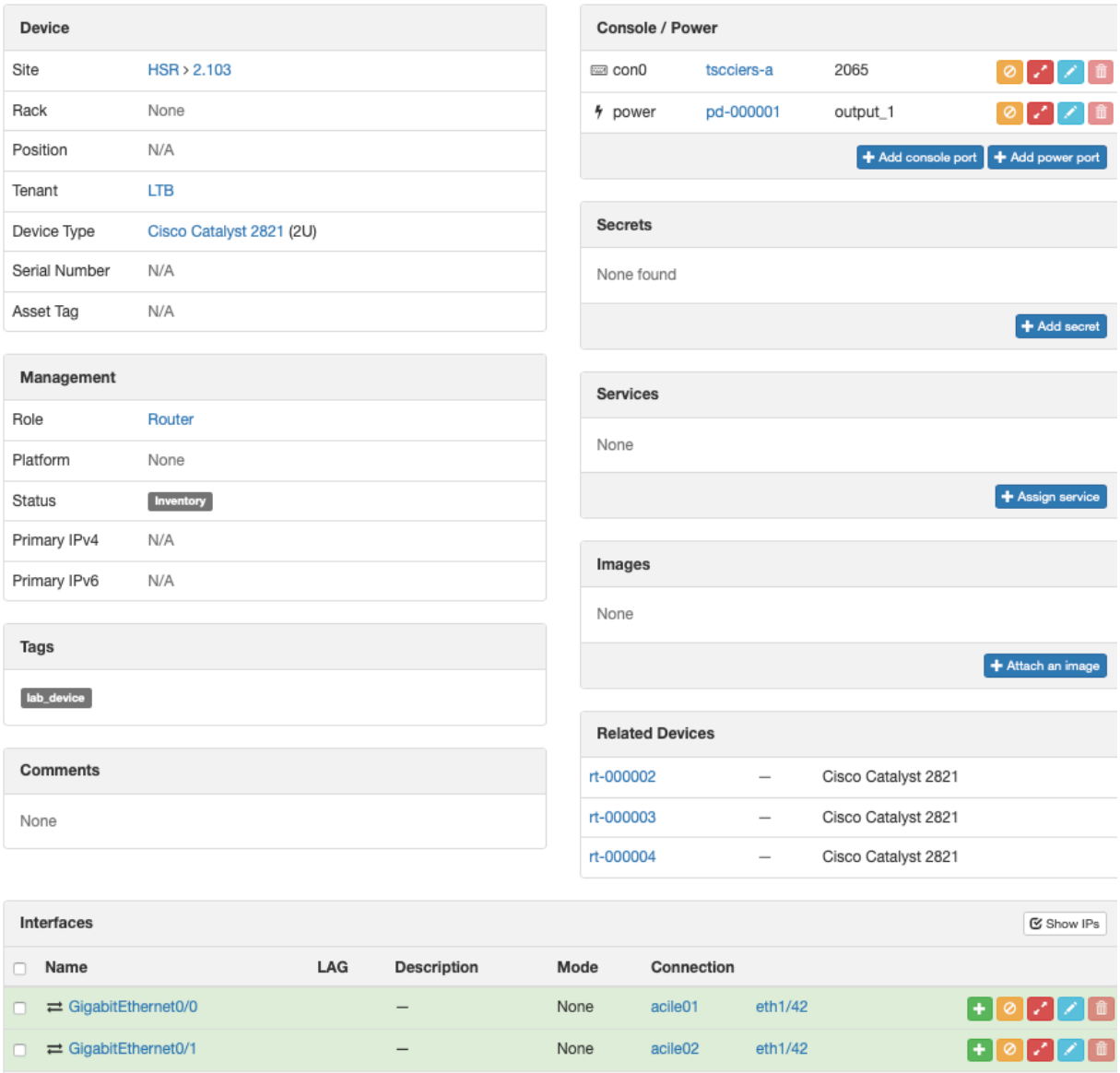



Abbildung 49: Netbox Device Details

### Config Context

Der Config Context für die ACI Leafs ist notwendig, damit die Orchestration die physischen Devices richtig miteinander verbinden kann. Folgendes Objekt muss im Kontext jedes Leafs konfiguriert werden:

```
{
   "aci": {
      "id": <aci_id>,
      "pod": <aci_pod>,
      "type": "leaf"
   }
}
```

Die benötigten Informationen sind im APIC auffindbar:

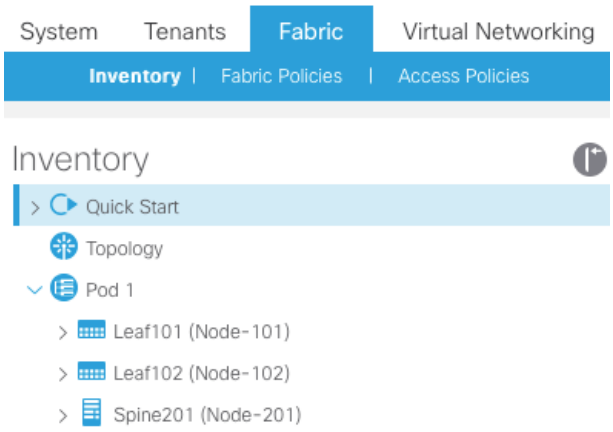



Abbildung 50: APIC Pod Info

### Netbox ACI Leaf Config Context


Abbildung 51: Netbox ACI Config Context

### Status
Es gibt verschiedene Status eines Gerätes. Die wichtigsten sind:

**Active** Das Gerät ist in Verwendung
**Inventory** Das Gerät ist im Inventar und wird zurzeit nicht verwendet
**Planned** Das Gerät ist in Planung und wurde noch nicht installiert

Unsere Orchestration verwendet den Status «Inventory», um abzufragen, welche Geräte für ein Lab verfügbar sind (siehe Netbox Devices, Seite 66).
In Kombination mit den Device Tags (siehe Device Tags, Seite 91) lassen sich so nur die freien Netzwerkgeräte von Netbox abfragen, die für die Lab Provisionierung von Bedeutung sind.

## 9.7.4 oVirt Template erstellen
Ein Template kann gemäss Dokumentation erstellt werden [49]. In unserem Fall mussten wir vorher noch ein Snapshot der VM erstellen [50], damit davon ein Template gemacht werden kann.

### Beispiel CSR 1000v Template
### System Settings
Die VMs müssen genügend RAM und CPU besitzen, damit sie überhaupt starten. Siehe Cisco Configuration Guide für weitere Informationen [13].

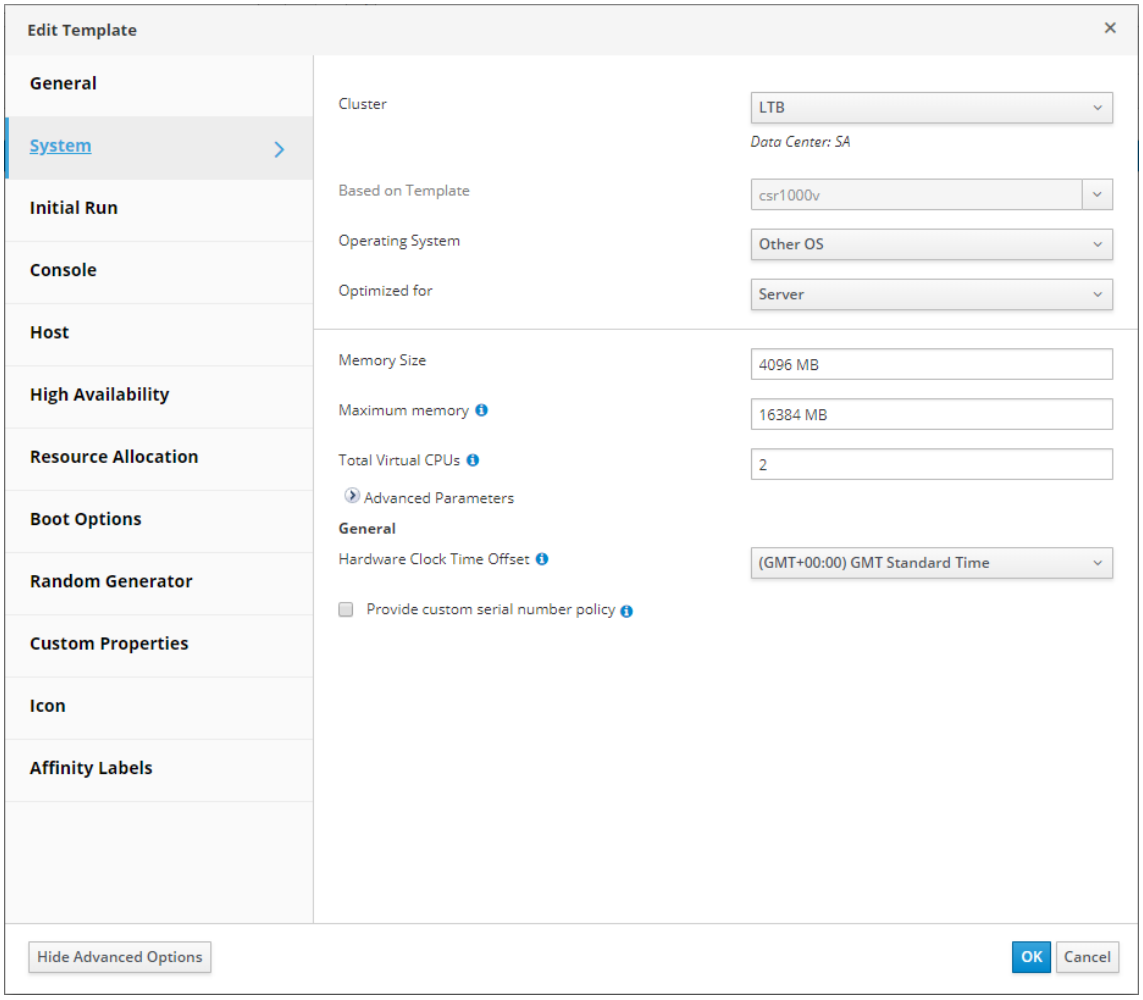



Abbildung 52: oVirt Template System Settings

## Console Configuration

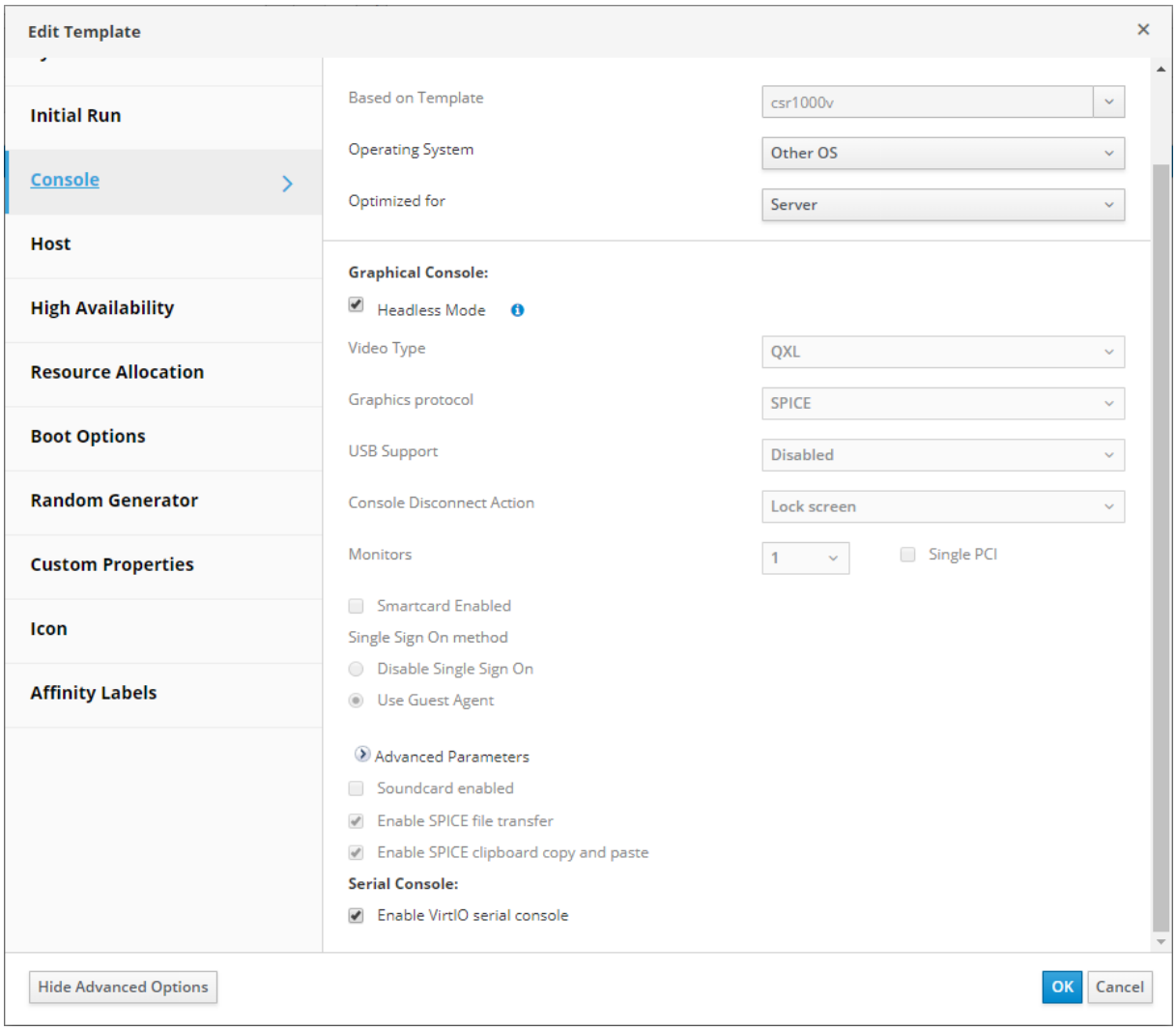



Abbildung 53: oVirt Template Console Settings

### 9.7.5 APIC Configuration

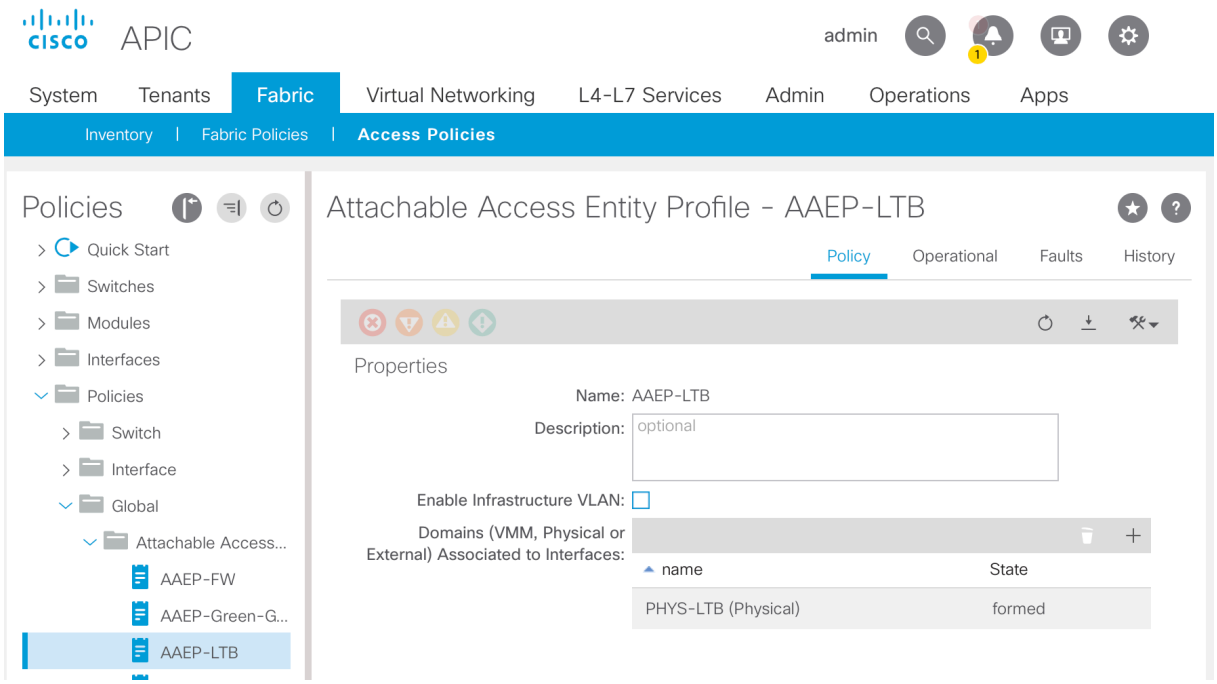



Abbildung 54: APIC global policy Attachable Access Entity Profile

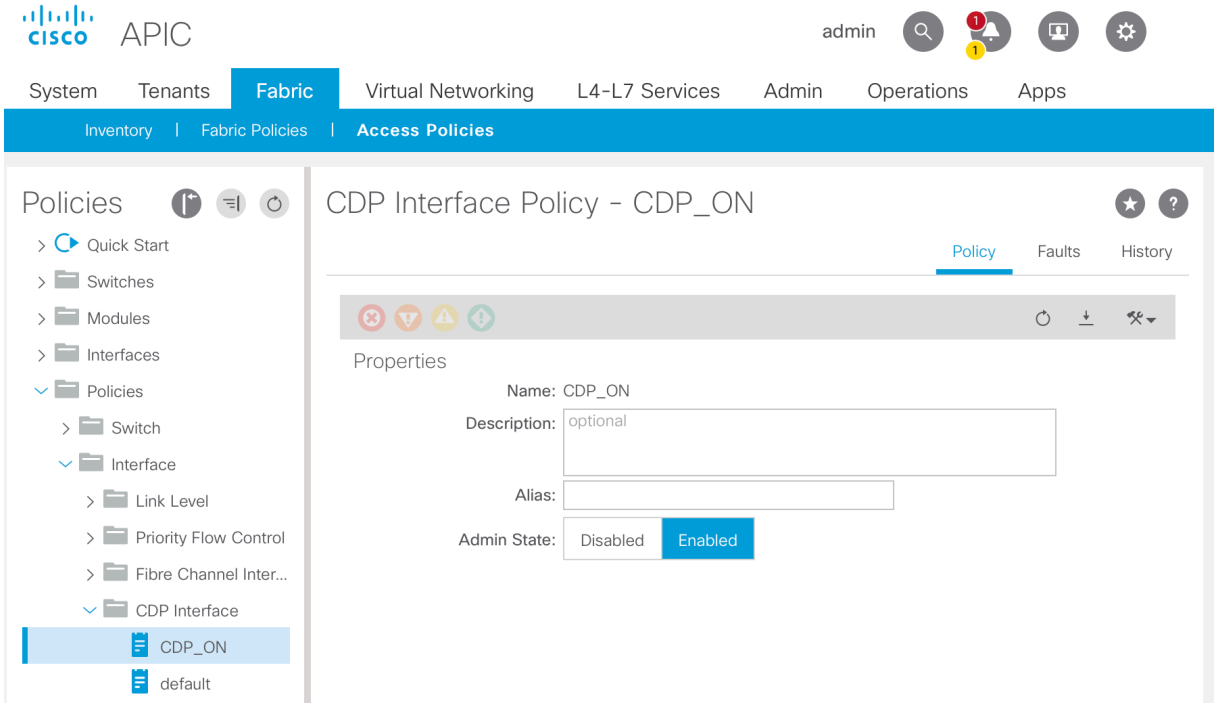



Abbildung 55: APIC Interface Policy CDP

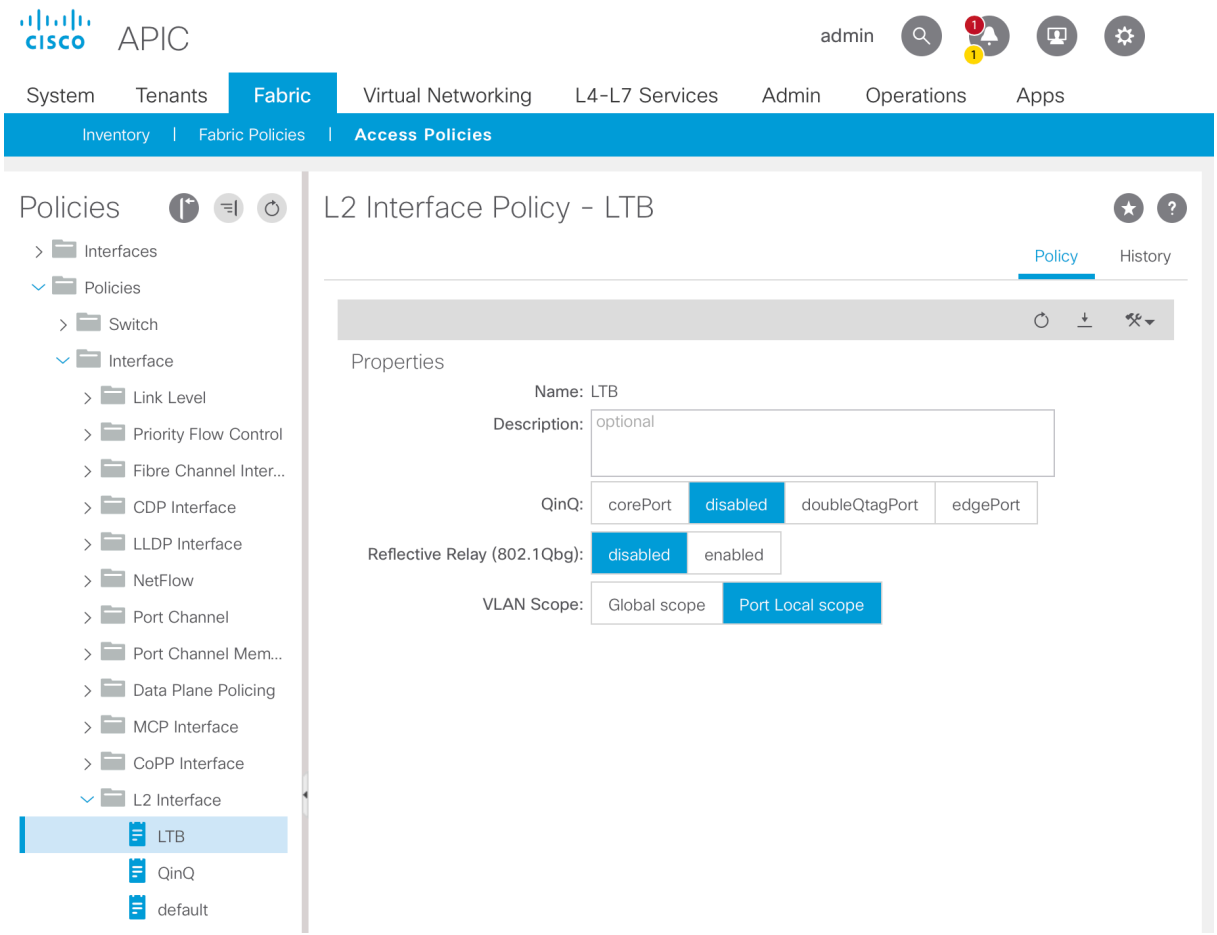



Abbildung 56: APIC Interface Policy Layer2

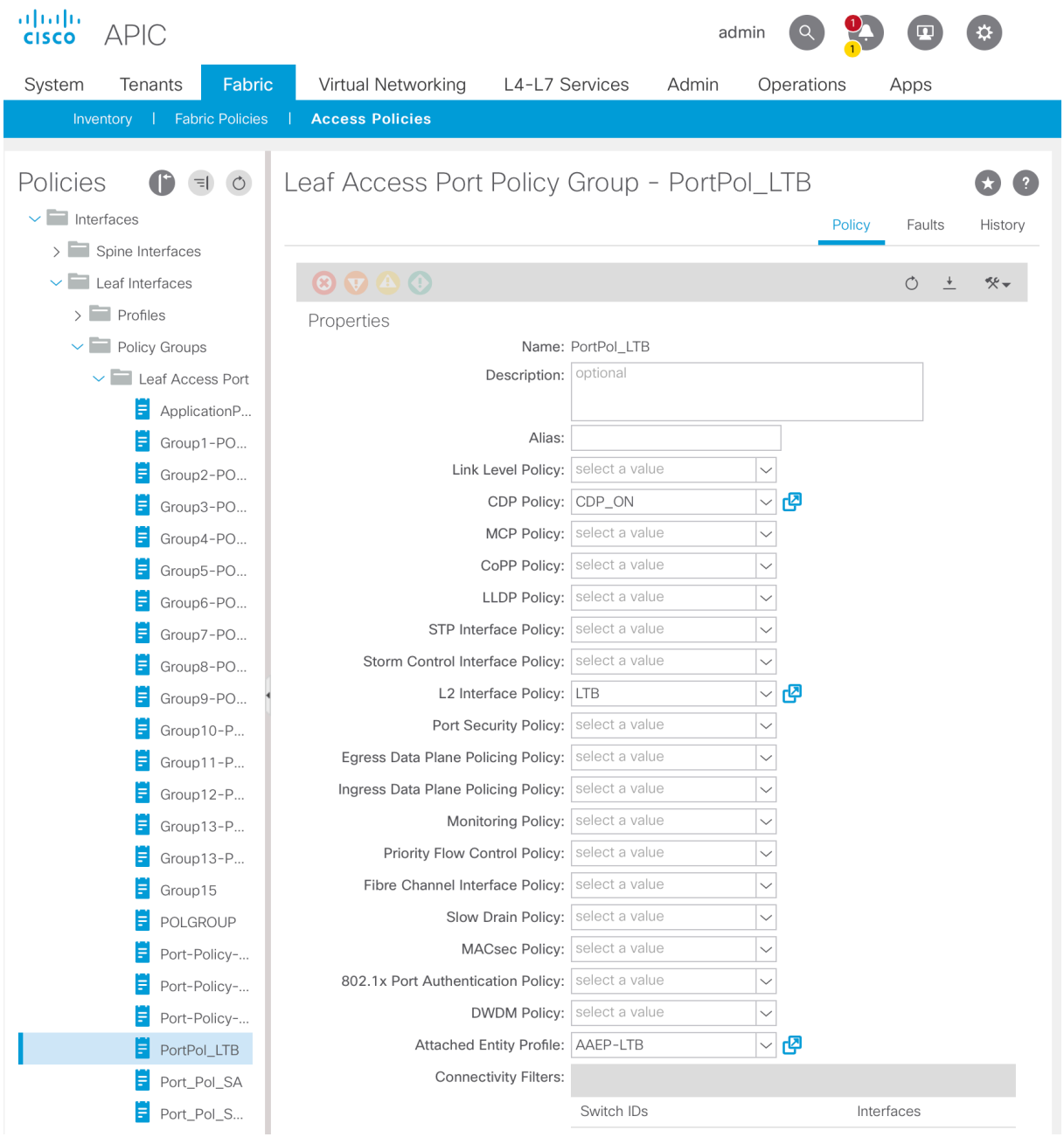



Abbildung 57: APIC Leaf Interface Policy Group

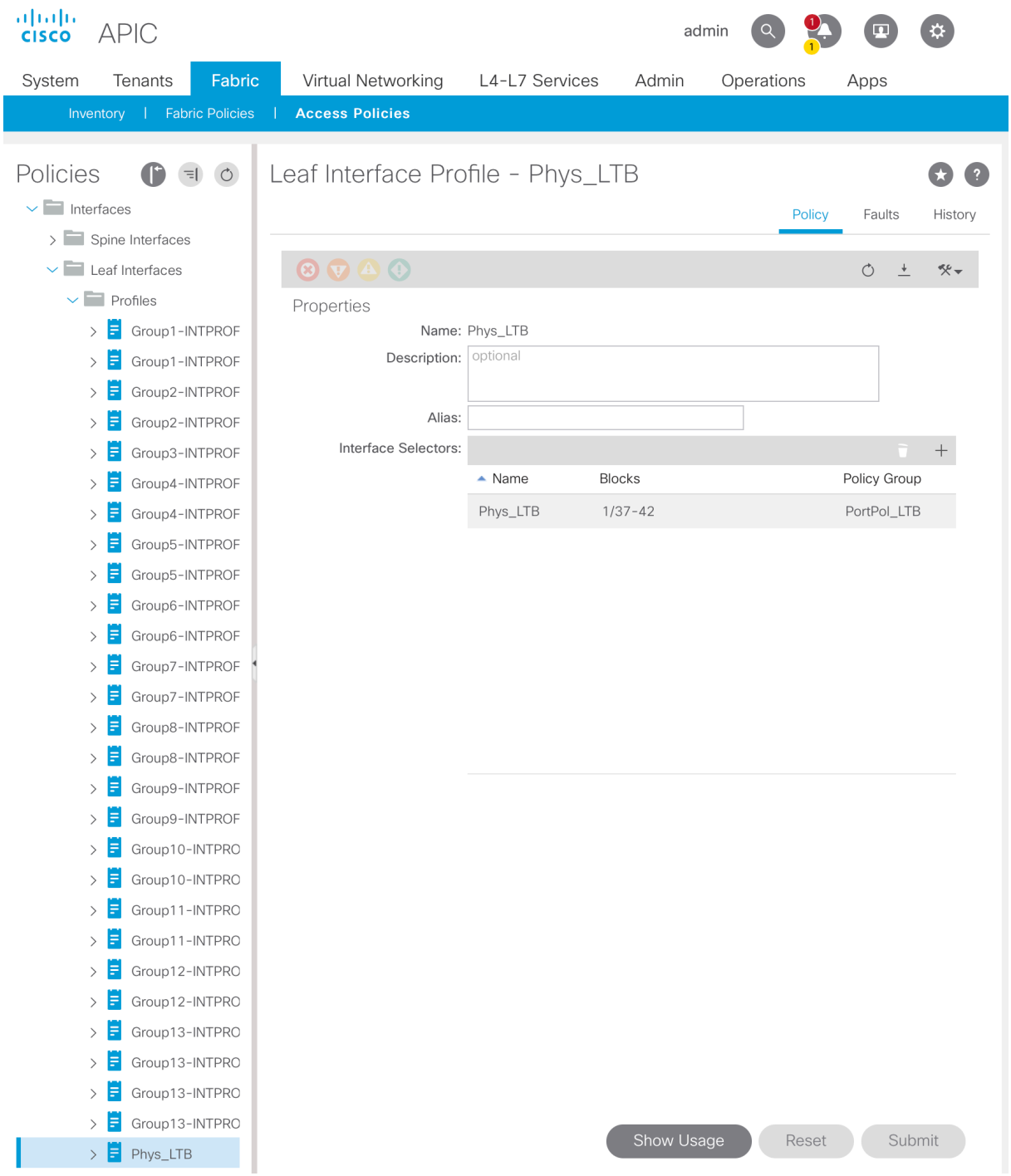



Abbildung 58: APIC Leaf Interface Profile

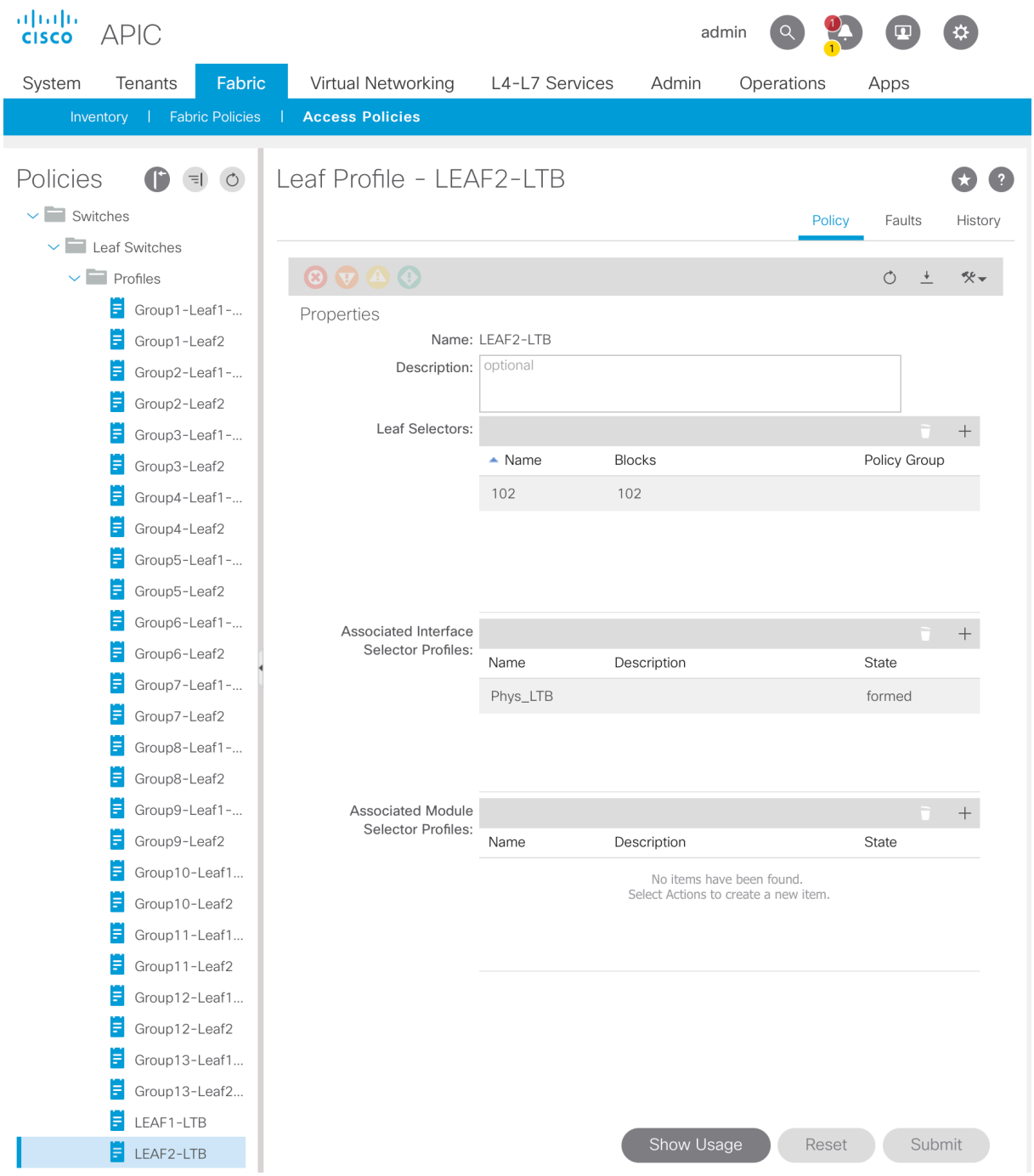



Abbildung 59: APIC Leaf Switch Profiles

## 9.8 Glossar

**ACI** Cisco Application Centric Infrastructure. 8, 11, 13–15, 21, 23, 55, 72, 73, 92

**API** Das Application Programming Interface ist eine Programmierschnittstelle, über die andere Programme an das System angebunden werden können. 14, 21, 23, 47, 49–51, 63, 88

**APIC** Cisco Application Policy Infrastructure Controller. 11, 15, 55, 72, 92

**BD** Bridge Domain. 23, 45, 46

**CCIE** Cisco Certified Internetwork Expert ist das höchste Level der technischen Cisco-Zertifizierungen und eine der angesehensten Zertifizierungen im Netzwerkbereich überhaupt. 11, 21

**CI/CD** Continuous Integration / Continuous Delivery. 26, 60

**CORS-Header** «Cross-Origin Resource Sharing» dient dazu, die HTTP REST API vor unbefugtem Zugriff durch Browserapplikationen zu schützen. 62

**EPG** Endpoint Group. 21, 46, 72

**INS** Institute for Networked Solutions. 11, 46, 49, 51, 57

**M2M** Machine to Machine. 88

**OSI Modell** Open Systems Interconnection Modell. 14

**PDU** Power Distribution Unit. 14, 15, 17, 21, 49, 88

**Snapshot** Ein Snapshot (Schnappschuss) bezeichnet man in der Virtualisierungstechnologie ein Abbild eines spezifischen Standpunktes einer virtuellen Maschine. 16

**SNMP** Simple Network Management Protocol. 49

**UI** User Interface. 47

**vCable+** Das vCable+ wird im Rahmen dieses Projekts als ein virtuelles Kabel beschrieben, dessen Eigenschaften (wie z.B. Jitter, Delay und Packed drop rate) sich variabel einstellen lassen, um z.B. eine langsamere WAN-Leitung zu simulieren. 21, 43, 46, 48

**VDSM** Virtual Desktop and Server Manager. 72

**VIRL** Cisco Virtual Internet Routing Lab. 41

**VM** virtual machine. 13–18, 23, 43, 44, 46–48, 51, 56, 57, 70

**VMM** Virtual Machine Manager. 23, 44

**VRF** Virtual Routing and Forwarding ist eine Routing-Technik in IP-Netzen die virtuelle Router auf einem physischen Router ermöglicht. 21, 45

**YAML** YAML Ain't Markup Language. 9

## 9.9 Abbildungsverzeichnis

## 9.10 Literaturverzeichnis


[1] AlDanial. Github Projekt Count Lines of Code. URL: https://github.com/AlDanial/cloc. (accessed: 2018-12-18).

[2] David Barroso. Execution Model. URL: https://nornir.readthedocs.io/en/2.0.0-beta/ref/internals/execution_model.html. (accessed: 2018-12-17).

[3] David Barroso. Nornir Netbox Plugin. URL: https://github.com/nornir-automation/nornir/blob/2.0.0-beta/nornir/plugins/inventory/netbox.py. (accessed: 2018-12-17).

[4] David Barroso. Writing a custom inventory plugin. URL: https://nornir.readthedocs.io/en/latest/howto/writing_a_custom_inventory_plugin.html. (accessed: 2018-12-17).

[5] Black-Box. Power Switch Twin 32 Satellite. URL: https://www.black-box.ch/ch-ch/fi/1574/12716/Power-Switch-Twin32-maitre/. (accessed: 2018-10-01).

[6] Brack. PDU 8x c13 ip. URL: https://www.brack.ch/value-19-pdu-8x-c13-ip-232375. (accessed: 2018-10-01).

[7] Elasticsearch B.V. Elasticsearch. URL: https://www.elastic.co/de/products/elasticsearch. (accessed: 2018-11-18).

[8] Kirk Byers. Netmiko Documentation. URL: https://netmiko.readthedocs.io/en/latest/. (accessed: 2018-12-17).

[9] Celery. Distributed Task Queue. URL: http://www.celeryproject.org. (accessed: 2018-12-17).

[10] Cisco. Cisco ACI Virtualization Guide, Release 2.1(1) - VMM Domain. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/1-x/virtualization/b_ACI_Virtualization_Guide_2_1_1/b_ACI_Virtualization_Guide_2_1_1_chapter_010.html#concept_74EFC437C0AA44A391676F70ACE59DF3. (accessed: 2018-12-11).

[11] Cisco. CISCO APIC Policy Model - Tenant. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/L2_config/b_Cisco_APIC_Layer_2_Configuration_Guide/b_Cisco_APIC_Layer_2_Configuration_Guide_chapter_011.html. (accessed: 2018-11-07).

[12] Cisco. Cisco CSR 1000v Download Source. URL: https://software.cisco.com/download/home/284364978/type/282046477/release/Everest-16.6.1. (accessed: 2018-10-14).

[13] Cisco. Cisco CSR 1000v Software Configuration Guide. URL: https://www.cisco.com/c/en/us/td/docs/routers/csr1000/software/configuration/b_CSR1000v_Configuration_Guide/b_CSR1000v_Configuration_Guide_chapter_00.html. (accessed: 2018-10-14).

[14] Cisco. Cisco IOS XRv Router Installation and Configuration Guide. URL: https://www.cisco.com/en/US/docs/ios_xr_sw/ios_xrv/install_config/b_xrvr_432_chapter_01.html. (accessed: 2018-10-14).

[15] Cisco. Cisco VIRL Labs. URL: https://github.com/Internetworkexpert/INE-VIRL. (accessed: 2018-12-17).

[16] Cisco. Virtual Internet Routing Lab. URL: http://virl.cisco.com. (accessed: 2018-12-17).

[17] Node.js Docker Working Group Collaborators. Node.js Docker Image. URL: https://github.com/nodejs/docker-node. (accessed: 2018-10-25).

[18] Containous. traefik - The Cloud Native Edge Router. URL: https://traefik.io/. (accessed: 2018-10-18).

[19] Martin Fowler. Mocks Aren't Stubs. URL: https://www.martinfowler.com/articles/mocksArentStubs.html. (accessed: 2018-12-16).

[20] Vivek Gite. SSH ProxyCommand example. URL: https://www.cyberciti.biz/faq/linux-unix-ssh-proxycommand-passing-through-one-host-gateway-server/. (accessed: 2018-10-30).

[21] GitLab. GitLab CI/CD - Introduction to pipelines and jobs. URL: https://docs.gitlab.com/ee/ci/pipelines.html. (accessed: 2018-12-17).

[22] GitLab. GitLab Runner. URL: https://docs.gitlab.com/runner/. (accessed: 2018-12-17).

[23] Gluster. GlusterFS - Server Quorum. URL: https://staged-gluster-docs.readthedocs.io/en/release3.7.0beta1/Features/server-quorum/. (accessed: 2018-12-14).

[24] GUDE. GUDE. URL: http://www.distribution.ch/de/products/1068738-8210-gude-gude-epc-net-8xrev3-rear. (accessed: 2018-10-01).

[25] Red Hat. Ansible Tower - Workflows. URL: https://docs.ansible.com/ansible-tower/latest/html/userguide/workflows.html. (accessed: 2018-10-18).

[26] Axel Haustant. Flask-RESTPlus's documentation. URL: https://flask-restplus.readthedocs.io/en/stable/. (accessed: 2018-10-20).

[27] Docker Inc. Overview of Docker Compose. URL: https://docs.docker.com/compose/overview/. (accessed: 2018-12-10).

[28] Intellinet. Smart PDU 8x C13. URL: https://intellinetnetwork.de/intellinet-de-19-8-fach-ip-steckdosenleiste-smart-pdu-mit-c13-kaltgeratesteckdosen-163682.html. (accessed: 2018-10-01).

[29] Vue JS. Vue JS documentation. URL: https://vuejs.org. (accessed: 2018-10-25).

[30] Vuetify JS. Vuetify JS documentation. URL: https://vuetifyjs.com/en/. (accessed: 2018-10-25).

[31] Linux-KVM. KVM - Kernel Virtual Machine. URL: https://www.linux-kvm.org/page/Main_Page. (accessed: 2018-12-14).

[32] Napalm. Napalm Automation. URL: https://napalm-automation.net. (accessed: 2018-10-18).

[33] Netapp. How deduplication works. URL: https://library.netapp.com/ecmdocs/ECMP1368859/html/GUID-7D804119-9EB1-4CE8-B02D-6AF215D17201.html. (accessed: 2018-12-14).

[34] Netapp. Netapp storage. URL: https://www.netapp.com/us/index.aspx. (accessed: 2018-12-14).

[35] Netbox. Github source. URL: https://github.com/digitalocean/netbox. (accessed: 2018-12-17).

[36] NetIO. M2M API Interface demo. URL: http://netio-4c.netio-products.com/#/interfaces/protocol-json. (accessed: 2018-10-06).

[37] NetIO. M2M API Interface documentation. URL: https://www.netio-products.com/files/download/sw/version/JSON---description-of-NETIO-M2M-API-interface_1-2-0.pdf. (accessed: 2018-10-15).

[38] NetIO. NetIO 4C. URL: https://www.netio-products.com/en/device/netio-4c. (accessed: 2018-10-01).

[39] NetworkToCode. Awesome Network Automation. URL: https://github.com/networktocode/awesome-network-automation/blob/master/README.md. (accessed: 2018-10-14).

[40] nine. ninech Docker Compose Environment. URL: https://github.com/ninech/netbox-docker/. (accessed: 2018-11-10).

[41] OpenStack. Mistral - OpenStack Workflow Engine. URL: https://docs.openstack.org/mistral/latest/index.html. (accessed: 2018-10-18).

[42] oVirt. Clusters. URL: https://ovirt.org/documentation/admin-guide/chap-Clusters.html. (accessed: 2018-12-17).

[43] oVirt. IRC Chat. URL: https://ovirt.org/community/about/contact.html#irc. (accessed: 2018-12-18).

[44] oVirt. oVirt Storage. URL: https://www.ovirt.org/documentation/admin-guide/chap-Storage/. (accessed: 2018-10-12).

[45] OpenNebula Project. OneFlow Specification. URL: https://docs.opennebula.org/5.6/integration/system_interfaces/appflow_api.html#appflow-api. (accessed: 2018-12-18).

[46] RedHat. Bugzilla. URL: https://bugzilla.redhat.com/enter_bug.cgi?classification=oVirt. (accessed: 2018-12-18).

[47] RedHat. Installing oVirt Node. URL: https://www.ovirt.org/documentation/install-guide/chap-oVirt_Nodes/. (accessed: 2018-12-02).

[48] RedHat. oVirt Documentation. URL: https://www.ovirt.org/documentation/. (accessed: 2018-10-12).

[49] RedHat. oVirt Documentation - Creating a Template. URL: https://ovirt.org/documentation/vmm-guide/chap-Templates/#creating-a-template. (accessed: 2018-10-12).

[50] RedHat. oVirt Documentation - Creating Snapshots. URL: https://www.ovirt.org/documentation/user-guide/user-guide/#creating-snapshots. (accessed: 2018-10-12).

[51] RedHat. oVirt Documentation - Serial Console Access. URL: https://ovirt.org/develop/release-management/features/virt/serial-console.html. (accessed: 2018-10-30).

[52] RedHat. oVirt Netzwerk setup. URL: https://ovirt.org/blog/2017/11/setting-up-multiple-networks-is-going-to-be-much-faster-in-ovirt-4-2/. (accessed: 2018-12-04).

[53] Armin Ronacher. Flask-SQLAlchemy documentation. URL: http://flask-sqlalchemy.pocoo.org/2.3/. (accessed: 2018-10-20).

[54] Philip Schmid. How To Install A NFS Server. URL: https://gist.github.com/PhilipSchmid/f75cb9d22efec9d7e24ade25abe8ecb2. (accessed: 2018-12-17).

[55] Scotch.io. Flask Or Django? An In-Depth Comparison | Part One. URL: https://scotch.io/bar-talk/flask-or-django-an-in-depth-comparison-part-one. (accessed: 2018-10-18).

[56] SmartBear Software. Bob Lee Swagger UI. URL: https://swagger.io/tools/swagger-ui/. (accessed: 2018-12-14).

[57] VMWare. vSphere Automation API 6.7. URL: https://code.vmware.com/apis/366/vsphere-automation. (accessed: 2018-12-18).

[58] Vue JS / Vuex. Vuex Documentation. URL: https://vuex.vuejs.org/. (accessed: 2018-10-25).

[59] Adam Wiggins. The Twelve-Factor App. URL: https://12factor.net/. (accessed: 2018-10-10).

[60] Wikibooks. Jump Hosts – Passing Through a Gateway or Two. URL: https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts#Jump_Hosts_--_Passing_Through_a_Gateway_or_Two. (accessed: 2018-10-30).

[61] Wikibooks. QEMU Images - Copy on write. URL: https://en.wikibooks.org/wiki/QEMU/Images#Copy_on_write. (accessed: 2018-12-12).

[62] Wikipedia. Atomare Operationen in Datenbanken. URL: https://de.wikipedia.org/wiki/Atomare_Operation. (accessed: 2018-12-14).

[63] Robin Winslow. Fix Docker's networking DNS config. URL: https://development.robinwinslow.uk/2016/06/23/fix-docker-networking-dns/. (accessed: 2018-12-10).

[64] WTI. Console server PDU 8x C13. URL: https://www.wti.com/p-324-cpm-800-1-e-console-server-pdu-8-port-8-outlet-dual-gige.aspx. (accessed: 2018-10-01).

## 9.11 Danksagungen

Wir danken folgenden Personen ganz herzlich für Ihre Unterstützung während der Studienarbeit:

- Prof. Laurent Metzger für die Betreuung unserer Arbeit als Experte
- Urs Baumann für die Betreuung und Unterstützung bei unserer Arbeit
- Christian Spielmann für die Einrichtung der Studienräume und die prompte Bereitstellung der VMs und DNS-Einträge
- Philip Schmid für die ermutigenden Worte und die Hilfestellung