Bachelorthesis

Entwicklung einer architekturübergreifenden PXE-Infrastruktur mit webbasierter Schnittstelle

Vorgelegt am:	04.09.2017
Von:	André Röder Hammerbrücker Str. 7f 08262 Muldenhammer
Studiengang:	Technische Informatik
Studienrichtung:	Daten- und Kommunikationstechnik
Seminargruppe:	TI2014
Matrikelnummer:	4001610
Praxispartner:	Hetzner Online GmbH Am Datacenter-Park 1 08223 Falkenstein
Gutachter:	Dipl.-Inf. Markus Schade (Hetzner Online GmbH) Dipl.-Ing. Ruben Mühlhans (Staatliche Studienakademie Glauchau)

STUDIEREN IM MARKT

Themenblatt Bachelorthesis

Studiengang Technische Informatik

Student:	**Andrè Röder**
Matrikelnummer:	**4001610**
Seminargruppe:	**4TI14-1**

Thema der Bachelorthesis

Entwicklung einer architekturübergreifenden PXE-Infrastruktur mit webbasierter Schnittstelle

Gutachter/ Betreuer:	Dipl.-Inf. Markus Schade
Gutachter (Studienakademie):	Dipl.-Ing. Ruben Mühlhans

Ausgabe des Themas:	**29.05.2017**
Abgabe der Arbeit an den SG am:	**21.08.2017, 14:00:00**

Menzel

Prof. Dagmar Menzel
Vorsitzende des Prüfungsausschusses
Technik

BA BERUFSAKADEMIE SACHSEN STAATLICHE STUDIENAKADEMIE GLAUCHAU UNIVERSITY OF COOPERATIVE EDUCATION

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AoE	ATA over Ethernet
API	Application Programming Interface
ARMv8	64-Bit ARM-Architektur (ARM64)
AS	autonomes System
B-API	Bootserver-API
BBS	BIOS Boot Specification
BDS	Boot Device Selection
BS	Betriebssystem
BSD	Berkeley Software Distribution
CIFS	Common Internet File System
CQL	Cassandra Query Language
CRUD	Create, Read, Update, Delete
CSM	Compatibility Support Module
DHCP	Dynamic Host Configuration Protocol
DRBD	Distributed Replicating Block Device
DSL	Domain Specific Language
DXE	Driver Execution Environment
EFI	Extensible Firmware Interface
ERB	Embedded RuBy
FCoE	Fibre Channel over Ethernet
FIN	Finnland
FSN	Falkenstein
GPL	General Public License
GPT	GUID Partition Table
GRUB	GRand Unified Boot Loader
HTTP	Hypertext Transfer Protocol
INT	Interrupt
ISC	Internet Systems Consortium
iSCSI	Internet Small Computer System Interface
JSON	Java Script Object Notation
LDAP	Lightweight Directory Access Protocol

LDIF	LDAP Data Interchange Format
MAC	Media Access Control
MMR	Multi Master Replikation
MNP	Managed Network Protocol
MVC	Model View Controller
NBG	Nürnberg
NBP	Network Bootstrap Programm
NFS	Network File System
NIC	Network Interface Card
NII	Network Interface Identifier Protocol
NVRAM	Non-Volatile Random Access Memory
OID	Object IDentifier
OpROM	Option ROM
OSI	Open Systems Interconnection
OU	Organizational Unit
PAP	Programmablaufplan
PCIe	Peripheral Component Interface Express
PI	Platform Initialization
PNP	Plug and Play
POST	Power On Self Test
PXE	Preboot Execution Environment
REST	Representational State Transfer
RFC	Request For Comments
RPC	Remote Procedure Call
RZ	Rechenzentrum
SMBIOS	System Management Basic Input/Output System
SNP	Simple Network Protocol
SOAP	Simple Object Access Protocol
SoC	System on Chip
SPOF	Single Point Of Failure
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol

TLS	Transport Layer Security
UDP	User Datagram Protocol
UEFI	Unified Extensible Firmware Interface
UNDI	Universal Network Driver Interface
URI	Uniform Resource Identifier
VPN	Virtual Private Network
WOL	Wake-On LAN

1. Einleitung

1.1 Umfeld und Ausgangssituation

Die Hetzner Online GmbH ist ein professioneller Webhosting-Dienstleister und erfahrener Rechenzentrenbetreiber. Die Angebotspalette erstreckt sich neben dem klassischen Webhosting, auf die Vermietung von Root- bzw. virtualisierten Servern, Colocation und die Bereitstellung von Diensten zur Domainregistrierung und Beantragung von SSL-Zertifikaten. Das Unternehmen betreibt in Deutschland zwei Datacenterparks an den Standorten Falkenstein und Nürnberg. Ein weiterer geplanter Standort in Finnland befindet sich zur Zeit im Bau. Zur Provisionierung der Serversysteme für den Endkunden sowie für Wartungs- und Diagnosezwecke existiert für die bestehenden Standorte eine einheitliche PXE-Infrastruktur, die aus verschiedenen Diensten und Anwendungen besteht.

1.2 Zielsetzung

Diese Bachelorthesis befasst sich mit der Anpassung bzw. Erweiterung der bestehenden PXE-Infrastruktur und Betrachtung der dazu notwendigen theoretischen Grundlagen. Ziel ist es eine Lösung zu entwickeln, mit der bestehende und neue Serversysteme, wahlweise mit oder ohne aktiviertem UEFI-Netzwerkstack, über PXE-Boot automatisch gestartet und vorkonfiguriert werden können. Da das Unternehmen auch Serversysteme auf Basis der ARMv8-Architektur in den Rechenzentren einsetzen möchte, sollen alle Systeme den gleichen Bootloader beim Startvorgang über PXE verwenden. Der Transfer von benötigten Startdateien im Netzwerk soll zukünftig primär über HTTP erfolgen, um auf den bisher eingesetzten TFTP-Server weitestgehend verzichten zu können und gleichzeitig die Übertragungsgeschwindigkeit zu steigern. Dazu sind Anpassungen an der bestehenden „Bootserver-API“ erforderlich, die die Konfigurationsdateien für den TFTP-Server generiert. Zudem wird geprüft, ob die derzeit getroffenen Maßnahmen bzgl. Ausfallsicherheit und Skalierbarkeit der PXE-Infrastruktur, im Hinblick auf die bevorstehende Anbindung des Standorts in Finnland, noch wirksam und ausreichend sind. Nach Abschluss der Planung werden die erarbeiteten Ergebnisse in einer Testumgebung praktisch umgesetzt.

2. Theoretische Grundlagen

2.1 Das Preboot Execution Environment

Das Preboot Execution Environment (PXE) ist ein von der Firma Intel im Jahr 1998 definierter Standard, der es ermöglicht, Computersysteme automatisiert über das lokale Netzwerk zu starten[1]. Dabei muss weder ein Betriebssystem (BS) auf dem System vorinstalliert sein, noch muss dieses über lokale Datenträger verfügen. Damit das BIOS mit der Netzwerkkarte interagieren kann, sind im PXE-Standard verschiedene Protokolle und Schnittstellen vorgesehen, wie in Abbildung 1 ersichtlich.

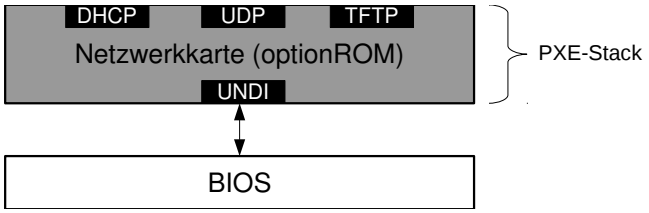


Abbildung 1 - PXE-Protokolle und -schnittstellen

Ein klassisches BIOS besitzt keinen eigenen Netzwerkstack, sondern muss dazu auf die implementierten Dienste in der Netzwerkkarte zurückgreifen, die in einem ROM, auch „Option ROM“ (OpROM) genannt, untergebracht sind. Beim Start eines Computers erkennt das BIOS über „Plug and Play“ (PNP) alle im System vorhandenen Erweiterungskarten, lädt den im ROM erhaltenen Code in den Hauptspeicher und führt diesen aus.

Das „Universal Network Driver Interface“ (UNDI) umfasst den eigentlichen Netzwerktreiber, der die Funktionalitäten der Schichten 1–4 im OSI-Modell abbildet. Über das „Dynamic Host Configuration Protocol“ (DHCP) sendet der Client mittels Broadcast eine Anfrage mit erweiterten Protokolloptionen an einen DHCP-Server. Dieser beantwortet die Anfrage mit einer IP-Adresse für den Client und übermittelt zusätzlich die Adressen der zur Verfügung stehenden Boot-Server. Danach kontaktiert der Client erneut über DHCP-Broadcast einen dieser Boot-Server, um die Adresse des TFTP-Servers und den Dateipfad für das Startprogramm (NBP) zu erhalten. Anschließend lädt der Client per TFTP, welches auf dem Transportprotokoll UDP aufsetzt, das NBP und führt dieses aus. Das NBP ist ein der Regel ein Bootloader, der nachfolgend den Start des Betriebssystems initiiert.

1 online: INTEL, 1999 (11.07.2017)

2.2 Das Unified Extensible Firmware Interface

2.2.1 Entwicklungsgeschichte

Das „Unified Extensible Firmware Interface“ (UEFI) ist der Nachfolger des klassischen PC-BIOS und definiert gemäß Spezifikation eine zusätzliche Abstraktionsschicht zwischen der Firmware des Systems und dem BS[2]. Die Entwicklung von UEFI geht auf das Jahr 1999 zurück, als Intel eine neue Bootumgebung für seine Itanium-Architektur benötigte[3]. Ursprünglich sollte dafür das bisherige BIOS in seiner damaligen Form verwendet werden, doch man erkannte sehr schnell, dass dessen Struktur für eine Portierung auf andere Architekturen eher ungeeignet und zeitaufwendig war. Deshalb entschied sich Intel für einen völlig neuen Ansatz und bezeichnete diesen abschließend als „Extensible Firmware Interface“ (EFI). Die Weiterentwicklung der Spezifikation setzte sich im Laufe der Jahre fort und wird seit 2005 durch das UEFI-Forum betreut. Aus EFI wurde UEFI. Im Forum sind mittlerweile neben Intel weitere namhafte Hersteller aus der Hard- und Softwarebranche, wie z. B. AMD, AMI, Apple, Dell, HP, IBM oder Microsoft vertreten.

2.2.2 Architektur

Die Architektur von UEFI gliedert sich in die zwei Hauptbestandteile „Platform Initialization“ (PI) und der eigentlichen Abstraktionsschicht (UEFI) zum BS, für die jeweils eigene Spezifikationen existieren. Der Bootprozess wird in sieben Phasen untergliedert, siehe Abbildung 2.

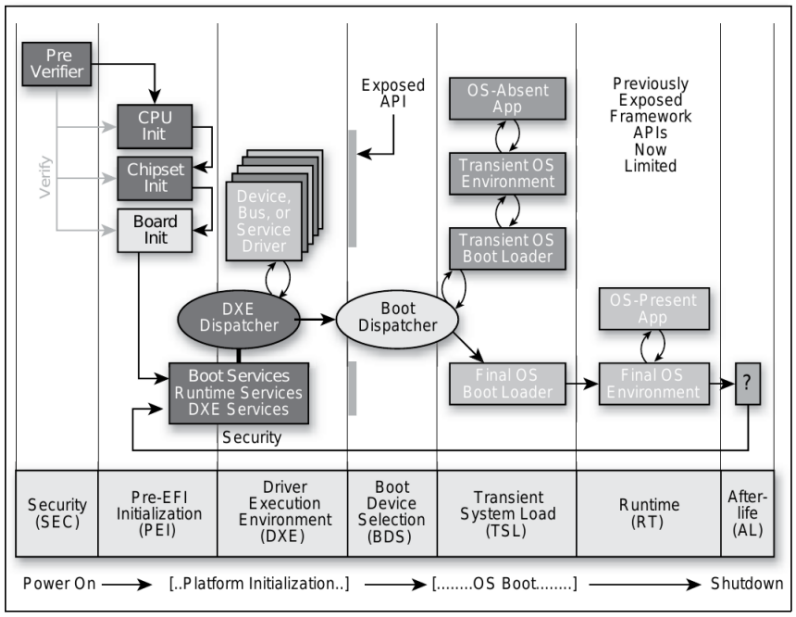


Abbildung 2 - UEFI-Bootphasen
(ZIMMER; u.a., 2010, S. 141)

2 online: UEFI Forum, 2016 (11.07.2017)
3 vgl. ZIMMER; u.a., 2010, S.6

Der PI-Teil wird durch die ersten vier Phasen abgedeckt, wo u. a. die erstmalige Initialisierung von wichtigen Systemkomponenten, wie CPU, Hauptspeicher, Interruptsystem und Zeitgebern erfolgt. Die DXE-Phase ist für das Laden der sogenannten „Boot Services“ und „Runtime Services“ verantwortlich, die verschiedene Dienste und Methoden für nachfolgenden Anwendungen, wie z. B. Bootloader, Diagnosesoftware und letztendlich das BS bereitstellen.

2.2.3 PXE unter UEFI

Der PXE-Standard ist unter UEFI weiterhin verfügbar, wurde hier jedoch erweitert und angepasst. Ein wesentlicher Unterschied zwischen UEFI und einem klassischen PC-BIOS besteht darin, dass bereits vor dem Start eines BS ein vollständiger TCP/IP-Stack zur Verfügung steht, der in der BDS-Phase initialisiert wird, siehe Abbildung 3. Die Kommunikation zwischen Netzwerkkarte (NIC) und Netzwerkstack erfolgt in UEFI über mehrere Abstraktionsschichten und definierten APIs, auch UEFI-Protokolle genannt, und erfordert auf der untersten Ebene nach wie vor einen eigenen Treiber, den der Hersteller der NIC bereitstellen muss. Der Netzwerktreiber kann dabei entweder Bestandteil der Firmware sein oder sich in einem OpROM befinden, wenn eine PCIe-Steckkarte verwendet wird.

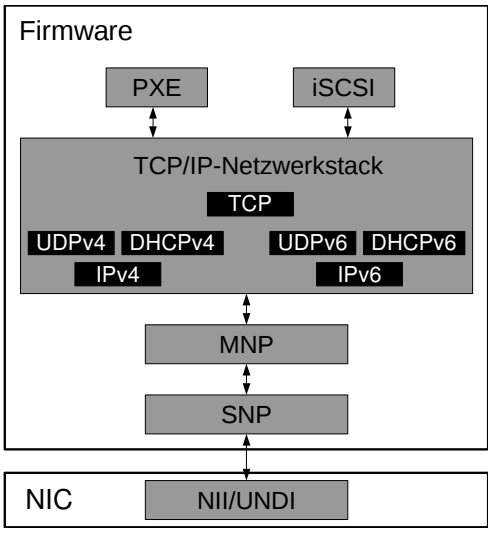


Abbildung 3 - Netzwerkstack und Treibermodell in UEFI
(eigene Darstellung in Anlehnung an DELL Inc., 2014, S. 7)

Damit ein Rechner mit UEFI-Firmware von einer Netzwerkkarte booten kann, muss der Netzwerktreiber gemäß dem Leitfaden von Intel zur Entwicklung von UEFI-Treibern nach wie vor als UNDI implementiert werden, um danach auf das in der Firmware bereits enthaltene „Simple Network Protocol“ (SNP) und „Managed Network Protocol“ (MNP) aufsetzen zu können[4]. SNP definiert einfache Methoden zum Senden und Empfangen von Netzwerkpaketen, sowie zum Initialisieren und Beenden der Netz-

4 online: INTEL, 2012, S. 447 (11.07.2017)

werkschnittstelle. MNP erlaubt asynchronen Netzwerkverkehr zwischen verschiedenen Treibern und Anwendungen zur gleichen Zeit und für mehrere Netzwerkkarten. Da UNDI im Sinne von UEFI kein Protokoll darstellt, wird über das „Network Interface Identifier Protocol“ (NII) noch eine Zwischenstufe geschaffen, die einen Einstiegspunkt zum Gerätetreiber definiert. Bei Verwendung von PXE-Boot über UEFI muss das NBP immer eine EFI-Anwendung sein.

Der UEFI-Netzwerkstack bietet zudem die Möglichkeit das NBP mittels HTTP von einem Webserver zu beziehen, was entscheidende Geschwindigkeitsvorteile gegenüber TFTP bietet.

2.3 Serversysteme mit 64-Bit ARM-Architektur

Mit Einführung der 64-Bit-Architektur ARMv8-A im Jahr 2011, hat ARM die Möglichkeit geschaffen, auch in den Markt der Server- und Desktopsysteme vorzudringen, anstatt sich nur auf den Bereich der eingebetteten Systeme zu konzentrieren. Damit verbunden, sollte auch die Vormachtstellung von Intel in diesem Segment gebrochen und dem Wunsch der Kunden nach alternativen und effizienteren Systemen Rechnung getragen werden. Da ARM selbst jedoch keine eigene Hardware herstellt, sondern nur Lizenzen für die einzelnen CPU-Kerne bzw. die Architektur verkauft, mussten entsprechende Lizenznehmer diese Aufgabe übernehmen. Erste Versuche, geeignete Prozessoren bzw. SoCs für den Serverbereich zu entwickeln, unternahmen die Firmen Applied Micro, Cavium und AMD. Unterstützt wurden sie dabei von Herstellern im Bereich Computerhardware und Auftragsfertigern der Halbleitertechnologie, wie z. B. Gigabyte, Globalfoundries oder TSMC. Aber nicht nur die Hardware musste entwickelt werden, sondern auch die dazugehörige Software, um letztendlich eine sinnvolle und gewinnbringende Nutzung der neuen Technologie durch potentiellen Kunden zu gewährleisten. Dies umfasst die Teilbereiche Firmware, Betriebssysteme und Anwendungssoftware. Jedoch kam es gerade im Firmwarebereich aufgrund von fehlenden Standards zu einer Art „Wildwuchs“ bzw. zu Insellösungen, die anfangs eine Integration der neuen ARM64-Systeme in eine aus vorwiegend mit x86-Servern bestehende Umgebung erschwerten bzw. unmöglich machten. Die Integration eines Testsystems bei der Hetzner Online GmbH wurde im Rahmen einer Studienarbeit[5] des Autors ausführlich untersucht. Erst als ARM und die beteiligten Partner sich auf gewisse Standards einigen konnten, verbesserte sich die Situation. Eine wichtige Entscheidung war als Firmware für diese Serversysteme grundsätzlich UEFI zu verwenden. Dadurch kann PXE-Boot in genau derselben Art und Weise erfolgen wie bei x86-Systemen. Anpassungen an den nachfolgenden Softwarekomponenten, die die Provisionierung neuer Serversysteme steuern, sind weiterhin notwendig, aber der Aufwand für die Wartung und Pflege ist wesentlich geringer, als wenn

5 siehe beiliegende CD

für beide Architekturen komplett unterschiedliche Lösungen existieren. Eine gute Ausgangsbasis bietet ein gemeinsamer Bootloader für PXE. Mögliche Lösungen sollen im Verlauf der nachfolgenden Arbeit untersucht werden.

Insgesamt betrachtet hat der Versuch, ARM64-Server im vorhandenen Markt wirksam zu platzieren, bisher noch nicht zu dem erhofften Erfolg geführt. Dies liegt nicht nur an den aufgetretenen Problemen im Softwarebereich und der geringen Auswahl an verfügbarer Hardware, sondern auch an der Leistungsfähigkeit der Systeme, die in vielen Anwendungsbereichen geringer ist, als vergleichbare Konkurrenzprodukte von Intel oder AMD. Weiterführende Informationen zu dieser Thematik finden sich in einem interessanten Artikel von Nermin HAJDARBEGOVIC[6].

Dennoch ist das Interesse von potentiellen Kunden an ARM64-Systemen weiterhin ungebrochen, wie die Bemühungen von Microsoft in Zusammenarbeit mit Qualcomm und Cavium zeigen, die Technologie für die Microsoft „Azure Cloud“ einzusetzen[7]. Auch bei Hosting-Providern besteht nach wie vor ein Interesse, Serversysteme auf Basis der ARMv8-Architektur anzubieten. Das dies erfolgreich möglich ist, zeigen Unternehmen wie OVH, Paket und Scaleway.

2.4 Webbasierte Programmierschnittstellen auf der Basis von REST

Eine Programmierschnittstelle (engl. API) ermöglicht den externen Zugriff auf Funktionen und Daten einer bestehenden Anwendung. Dadurch kann die Anwendung z. B. von außen durch andere Programme gesteuert und ein Datenaustausch zwischen den Anwendungen durchgeführt werden. Hierzu wird ein gemeinsames Protokoll und ein Datenaustauschformat definiert. In den letzten Jahren geht der Trend verstärkt dahin, APIs für Internetanwendungen nach dem REST-Prinzip zu entwickeln. Der Begriff „Representational State Transfer“, kurz REST, geht auf eine Dissertation von Roy FIELDING aus dem Jahr 2000 zurück und beschreibt einen Architekturstil bzw. ein Entwurfsmuster für die Realisierung von verteilten Anwendungen, die Dienste über das Internet bereitstellen[8]. Für den Datenaustausch zwischen Client und Server kommt das Protokoll HTTP zum Einsatz. REST stellt eine einfacher zu implementierende Alternative für das bis dahin hauptsächlich eingesetzte SOAP in Kombination mit XML-RPC dar.

Die Architektur von REST gliedert sich in drei wesentliche Bestandteile[9]:

- Ressourcen und Repräsentationen
- Hypermedia

6 online: HAJDARBEGOVIC, 2016 (13.07.2017)
7 online: TILLEY, 2017 (13.07.2017)
8 online: FIELDING, 2000 (29.04.2017)
9 online: TILKOV, 2009 (29.04.2017)

- Einheitliche Schnittstelle

Ressourcen kennzeichnen eindeutig identifizierbare Objekte, die innerhalb einer Anwendung verarbeitet werden und nach außen zugänglich sind. Sie ergeben sich in der Regel aus den definierten Klassen im Programm oder aus Entitäten einer bestehenden Datenbank. Betrachtet man bspw. ein Warenwirtschaftssystem, können Artikel, Kunden oder Bestellungen als Ressourcen angesehen werden. Der Zugriff auf die Ressourcen geschieht einheitlich über einen „Uniform Resource Identifier“ (URI) bzw. einen „Uniform Resource Locator“ (URL).

Als Repräsentation wird die Darstellung einer Ressource bezeichnet, die sinnvollerweise in einem Standardformat, wie z. B. JSON oder XML erfolgen sollte. Dabei ist es von Vorteil, wenn die Anwendung stets mehrere dieser Formate unterstützt, um die Kompatibilität mit möglichst vielen Client-Anwendungen sicherzustellen.

Hypermedia beschreibt in REST die Verbindung zu anderen Ressourcen oder erweiterten Attributen. Dies wird über enthaltene Links in den Repräsentationen realisiert, die der Nutzer oder die Client-Anwendung auswerten kann. Ein einfaches Beispiel wäre eine Webseite, die mittels HTML im Browser dargestellt wird und anklickbare Links enthält, welche über das „href-Attribut“ definiert wurden.

Die einheitliche Schnittstelle stellt alle zulässigen Operationen auf die Ressourcen anwendungsübergreifend zur Verfügung. REST benutzt hierfür hauptsächlich die in der RFC 2616[10] festgelegten HTTP-Methoden GET, POST, PUT und DELETE. Damit lassen sich auch die grundlegenden CRUD-Operationen für Datenbanken, wie in Tabelle 1 ersichtlich, sehr komfortabel abbilden.

HTTP-Methode	Bedeutung	sicher/idempotent	CRUD-Operation
GET	Repräsentation der Ressource abfragen	X / X	READ
POST	Neue Ressource anlegen	– / –	CREATE
PUT	Ressource ändern oder neu anlegen	– / X	UPDATE oder CREATE
DELETE	Ressource löschen	– / X	DELETE
X ja – nein			

Tabelle 1 - Vergleich von HTTP-Methoden und CRUD-Operationen

Eine Operation wird als sicher bezeichnet, wenn sie frei von Seiteneffekten ist und aus dem Aufruf keine weiteren Verpflichtungen entstehen. Idempotente Operationen zeichnen sich dadurch aus, dass die mehrfache Ausführung einer Operation dasselbe Ergebnis erzeugt, als wenn sie nur einmal ausgeführt wird. Der Zustand des Sys-

10 online: RFC 2616, 1999 (15.08.2017)

tems kann sich aber durchaus nach der ersten Ausführung ändern.[11] Weiterhin geschieht die Kommunikation zwischen Client und Server grundsätzlich zustandslos, d. h. der Server erhält vom Client alle notwendigen Informationen, die für die Verarbeitung der Anfrage notwendig sind und es wird kein Bezug zu einer vorhergehenden Anfrage hergestellt oder sonstige Informationen für die aktuelle Sitzung gespeichert.

11 vgl. EDLICH; u.a., 2011, S. 53

3. Analyse der bestehenden Umgebung

3.1 Standorte und allgemeine Netzwerkstruktur

Die Hetzner Online GmbH betreibt zur Zeit zwei Datacenterparks mit mehreren Rechenzentren an den Standorten Falkenstein (FSN) und Nürnberg (NBG), die über angemietete Glasfaserleitungen redundant miteinander verbunden sind. Die Standorte bilden zusammen ein sogenanntes autonomes System (AS), welches durch das Unternehmen verwaltet wird. Ende des Jahres soll ein dritter Standort in Finnland in Betrieb genommen werden, der ebenfalls über Glasfaserleitungen in das bestehende AS integriert wird. Innerhalb des öffentlichen Netzwerkes gibt es ein VPN für den Eigenbetrieb, an das auch der Verwaltungsstandort in Gunzenhausen angebunden ist. Das VPN besitzt eine eigene PXE-Infrastruktur, die im Rahmen dieser Bachelorthesis nicht betrachtet wird. In den Rechenzentren befinden sich vorwiegend Serversysteme auf Basis der x86-Architektur, die von den Kunden des Unternehmens und für den Betrieb firmeneigener Systeme genutzt werden.

3.2 PXE-Infrastruktur

3.2.1 Überblick

Für die Provisionierung der Server sowie zur Durchführung von Wartungsarbeiten und zur Fehlerdiagnose existiert eine PXE-Infrastruktur, die historisch gewachsen ist. Die Umgebung besteht aus den typischen Standardkomponenten wie z. B. DHCP-, TFTP- und Dateiservern, weist jedoch einige Besonderheiten auf, die aufgrund der besonderen Anforderungen bei einem Hosting-Provider, verbunden mit der hohen Serveranzahl und getroffenen Vorkehrungen hinsichtlich Redundanz und Verfügbarkeit entstanden sind. Abbildung 4 zeigt die Grundstruktur der Umgebung. Alle Server verwenden ein Linux-Betriebssystem.

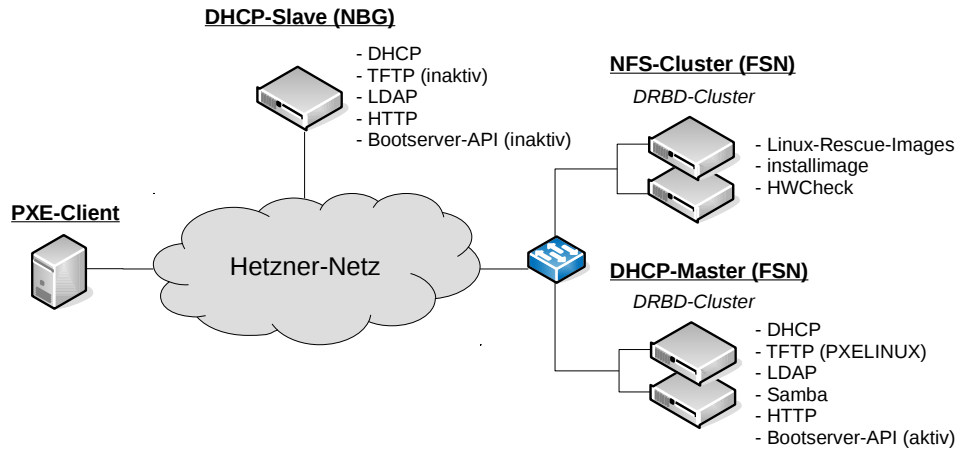


Abbildung 4 - Bestehende PXE-Infrastruktur

3.2.2 DHCP-Cluster

Die DHCP-Server werden mit der Software „ISC-DHCP“ betrieben und arbeiten in einer Master-Slave-Konfiguration mit Failover-IP-Adressen, die auf den Routern in den RZs hinterlegt sind. Der Master-Cluster besteht aus zwei Servern, die sich in Falkenstein befinden, und wird über DRBD in Verbindung mit „Corosync“ und „Pacemaker“ realisiert. Ein weiterer DHCP-Server, der als Slave arbeitet, steht in Nürnberg. Die DHCP-Leases werden in einem LDAP-Verzeichnis auf den Servern abgespeichert, die im Mirror-Mode arbeiten, um einen Schreibzugriff auf mehrere Instanzen zu ermöglichen. Auf den DHCP-Servern läuft zudem ein TFTP-, HTTP- und Samba-Server, deren Verzeichnisse über DRBD synchronisiert werden und die Bootserver-API (B-API). Der TFTP-Server dient zur Auslieferung des Bootloaders PXELINUX und nachfolgender Dateien, wie z. B. Linux-Kernel oder Diagnoseprogramme. Der Samba-Server stellt die XML-Dateien und ISO-Abbilder für die unbeaufsichtigte Windows-Installation bereit. Der Webserver wird von der B-API benötigt. Fällt ein DHCP-Server im Master-Cluster aus, übernimmt der zweite Server dessen Funktion automatisch. Ist der komplette Cluster nicht mehr erreichbar, muss auf dem Slave die DHCP-Konfiguration manuell angepasst werden, damit das TFTP-Verzeichnis dort zur Verfügung steht.

3.2.3 NFS-Cluster

Der NFS-Cluster befindet sich in Falkenstein und wird mit denselben Softwarekomponenten realisiert, wie auch der DHCP-Cluster. Er dient zur zentralen Datenablage von benötigten Anwendungen und Daten, die vor allem nach dem PXE-Boot gebraucht werden. Dazu zählen u. a. die Abbilddateien für das Linux-Rescue-System und das Programm „installimage“, das zur Installation der Betriebssysteme dient sowie das Programm „HWCheck“ zum Inventarisieren und Überprüfen von Serversystemen.

3.2.4 Bootserver-API

Die B-API ist die zentrale Komponente für die automatische Konfiguration bestehender und neuer Serversysteme. Sie besitzt eine REST-Schnittstelle und kommuniziert über das Datenaustauschformat JSON mit anderen Anwendungen, siehe Abbildung 5. Das Programm wurde mit dem Framework „Grape“[12] umgesetzt, welches in der Programmiersprache Ruby geschrieben ist. Die B-API erzeugt die benötigten Textdateien für PXELINUX und legt diese im Verzeichnis des TFTP-Servers ab. Weiterhin kommuniziert sie mit dem LDAP-Verzeichnis, um vorhandene, statische DHCP-Leases anzuzeigen, neue Lease-Einträge anzulegen oder nicht mehr benötigte Einträge zu löschen. Die hierzu nötigen Informationen kommen entweder

12 http://www.ruby-grape.org (06.08.2017)

automatisch aus der Kunden- und Auftragsverwaltung „Hetzner-Robot“ oder werden manuell über die Webanwendung „RZ-Admin“ (Frontend) generiert.

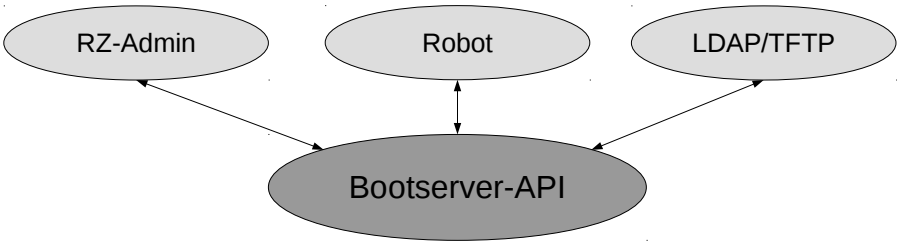


Abbildung 5 - Schnittstellen zur Bootserver-API

3.2.5 PXELINUX

In der derzeitigen Konfiguration wird über die DHCP-Server PXELINUX[13] in der Version 3.82 ausgeliefert. PXELINUX ist ein Netzwerk-Bootloader aus dem SYSLINUX-Projekt, welches eine Sammlung verschiedener Bootloadern umfasst[14]. Aktuell wird für den produktiven Einsatz die Version 6.03 von 2014 empfohlen. PXELINUX wird sehr gerne eingesetzt, weil es recht einfach zu konfigurieren und sehr flexibel ist. Es bietet eine Vielzahl von Konfigurationsoptionen und eine einfache Menüsteuerung. Unterstützung für UEFI gibt es ab Version 6. Allerdings hat PXELINUX auch Nachteile. So wird gegenwärtig nur die x86-Architektur unterstützt und die Menüs haben einen rein statischen Charakter, d. h. es existiert keine Möglichkeit, diese dynamisch zur Laufzeit zu verändern. Die Verwendung der relativ alten Version von PXELINUX aus dem Jahr 2009 resultiert aus der Tatsache, dass es bei bereits früher durchgeführten Tests mit der Version 6.03 zu Problemen mit einem älteren Servermodell kam.

3.2.6 Einschränkungen der bestehenden Infrastruktur

Durch die vorgenommene Analyse, wurden bis jetzt folgende Einschränkungen identifiziert:

- Keine Unterstützung für UEFI über PXE-Boot
- PXELINUX nur für x86-Architektur einsetzbar
- Datentransfer nur über TFTP möglich

Die fehlende Unterstützung für PXE-Boot über UEFI führt aktuell und längerfristig betrachtet zu Problemen. So ist bspw. die automatische Installation aktueller Windows-Betriebssysteme bei der Verwendung von Datenträgern größer als 2 TB nicht möglich, da Windows für die Unterstützung des GPT-Partitionsschemas zwingend UEFI benötigt. Gegenwärtig wird in diesem Fall die Installation von Windows durch einen Mitarbeiter aus dem Support manuell mit einem USB-Stick durchgeführt.

13 http://www.syslinux.org/wiki/index.php?title=PXELINUX (24.04.2017)
14 online: ANVIN; CONNOR, 2008, S. 11 (24.04.2017)

64-Bit Linux-Betriebssysteme unterstützen GPT zwar auch ohne UEFI, es ist aber damit zu rechnen, dass in nicht allzu ferner Zukunft, die UEFI-Firmware zukünftiger x86-Systeme kein „Compatibility Support Module“ (CSM) mehr enthält. Das CSM emuliert ein klassisches PC-BIOS inkl. herkömmlichen PXE. Dies hat zur Folge, dass PXE-Boot dann nur noch über UEFI möglich ist. Die gleiche Problematik ergibt sich beim produktiven Einsatz von ARM64-Systemen, deren UEFI-Firmware, ähnlich wie bei aktuellen Geräten von Apple, generell kein CSM enthält.

Die Verwendung von TFTP zur Übertragung der Startdateien hat in der bestehenden Struktur ebenfalls Nachteile. TFTP ist zwar ein sehr einfaches Protokoll für den Datenaustausch, wird aber aufgrund fehlender Authentifizierungsmechanismen als unsicher eingestuft und sollte deshalb mit entsprechender Vorsicht eingesetzt werden. Als Transportprotokoll kommt das verbindungslose UDP Einsatz. Um zu gewährleisten, dass die Daten korrekt übertragen werden, existiert in TFTP ein Algorithmus, der den Empfang der gesendeten Datenpakete prüft und nach einer gewissen Zeitspanne bestätigt[15]. Kommt ein Datenpaket nicht an, wird der Client aufgefordert, dieses Paket erneut zu senden. Dies führt zu spürbaren Geschwindigkeitseinbußen bei der Datenübertragung, verglichen mit FTP oder HTTP, gerade wenn größere Dateien übertragen werden müssen. Dieser Effekt tritt auch bei hoher Netzwerkauslastung und verstärkten Anfragen an den TFTP-Server auf. Da die Konfiguration des DHCP-Servers für die Angabe des TFTP-Servers (next-server) immer nur eine IP-Adresse erlaubt, müssen alle Standorte denselben TFTP-Server verwenden, wodurch dieser zu einem Flaschenhals wird. Weiterhin wurde der Quellcode des TFTP-Servers von Hetzner dahingehend modifiziert, dass die Konfigurationsdateien nach dem erfolgreichen Abruf automatisch gelöscht werden. Damit können die von der Linux-Distribution bereitgestellten Updates des Programms nicht verwendet werden und der Quelltext muss entsprechend selbst kompiliert werden.

15 online: RIEMERSMA, 2012 (18.07.2017)

4. Planung der neuen Lösung

4.1 Anforderungsdefinition

Bevor mit der Planung der neuen Lösung begonnen werden kann, müssen die Anforderungen klar definiert sein. Unvollständige oder fehlende Anforderungen führen zu Fehlern in der Planung und beeinflussen letztendlich das gewünschte Endergebnis. Zudem muss die Planung bezogen auf das vorliegende Projekt sehr genau und gewissenhaft durchgeführt werden, da die PXE-Infrastruktur eine sehr wichtige Komponente in den Datacenterparks darstellt. Fehler und Ausfälle würden eine starke Einschränkung des Geschäftsbetriebes verbunden mit finanziellen Verlusten für das Unternehmen nach sich ziehen, die es zu vermeiden gilt.

Die Anforderungen wurden in einer Besprechung mit dem Teamleiter der Produktentwicklung erarbeitet und umfassen folgende Punkte:

- Verwendung eines einheitlichen Bootloaders für Legacy-Boot und UEFI
- Unterstützung der x86- und ARMv8-Architektur
- Vorwiegende Nutzung von HTTP zur Datenübertragung der Startdateien und weitestgehender Verzicht auf TFTP
- Erzeugung von dynamischen Bootmenüs über die B-API
- Unterstützung von passwortgeschützten Menüs
- Einfache Erweiterbarkeit und Wartung möglich
- Gewährleistung von Ausfallsicherheit und Skalierbarkeit

Die ersten drei Anforderungen ergeben sich aus den bereits in der Analyse festgestellten Einschränkungen. Der Einsatz von dynamisch erzeugten Bootmenüs ist beim Einsatz von unterschiedlichen Rechnerarchitekturen sehr vorteilhaft, da nur eine initiale Menüstruktur existieren muss, die dann je nach vorhandener Firmware und Architektur auf dem PXE-Client entsprechend ausgewertet und zur Anzeige gebracht wird. Somit werden spätere Anpassungen wesentlich einfacher, als wenn für jede Architektur ein unterschiedlicher Satz an Dateien zu pflegen ist. Zudem kann durch die Nutzung von HTTP in Verbindung mit der B-API die Startkonfiguration direkt an den PXE-Client ausgeliefert werden, anstatt diese wie bisher als Textdatei auf dem TFTP-Server abzuspeichern und später wieder zu löschen. Da die PXE-Infrastruktur generell hohen Anforderungen an die Verfügbarkeit unterliegt, müssen die bisher getroffenen Maßnahmen zur Ausfallsicherheit und Skalierbarkeit in Bezug auf die geplante Anbindung des Standorts in Finnland noch einmal überprüft und ggf. angepasst werden. Mögliche Optionen sind bspw. das Hinzufügen weiterer DHCP- und LDAP-Server oder die Reorganisation des DRBD-Clusters.

4.2 Betroffene Systeme und Anwendungen

Bezogen auf die Anforderungen und der bisherigen Analyse, sind bei der Umstellung auf eine neue Lösung zunächst die in der Tabelle 2 aufgeführten Anwendungen und Systeme betroffen, welche bei der Planung gesondert zu betrachten sind.

Serverdienste und Anwendungen	DHCP-Server, Rescue-Linux, installimage, HWCheck, Bootserver-API, Hetzner-Robot
PXE-Clients	Rootserver und virtuelle Server der Kunden und im Eigenbetrieb

Tabelle 2 - Betroffene Systeme und Anwendungen

4.3 Auswahl des Bootloaders für PXE

4.3.1 Vorüberlegungen und Eingrenzung

Ausgehend von den definierten Anforderungen empfiehlt sich die Planung in einzelne Teile zu untergliedern. Der erste Teil umfasst die Auswahl eines neuen Bootloaders. Darauf aufbauend ergeben sich die durchzuführenden Anpassungen an den beteiligten Serverdiensten und Anwendungen. Für die Suche nach existierenden Bootloadern, bietet die englische Seite von Wikipedia eine sehr gute Übersicht[16]. Betrachtet man die Liste, scheint die Auswahl zunächst sehr groß zu sein. Sie lässt sich jedoch sehr effektiv reduzieren, wenn die vier wichtigsten funktionalen Anforderungen betrachtet werden. Diese sind die Unterstützung für UEFI in Kombination mit PXE-Boot und die Kompatibilität zu x86- und ARM64-Systemen. Zudem muss die Software unter einer freien Lizenz, wie z. B. GPL oder BSD, verfügbar sein. Zuerst wird geprüft, ob UEFI unterstützt wird. Dies reduziert die Liste auf sechs Einträge, von denen der Windows-Bootloader entfällt, da er proprietär ist. Nun betrachtet man die Spalte „LAN“, um die Möglichkeit von PXE-Boot zu prüfen. Danach verbleiben nur noch zwei Möglichkeiten – SYSLINUX und GRUB2. Da SYSLINUX aufgrund der Forderung nach einer architekturübergreifenden Lösung entfällt, bleibt am Ende nur GRUB2 übrig. GRUB2 erfüllt aber eine andere wesentliche Anforderung nicht, nämlich die Unterstützung von HTTP für den Datentransfer. Somit käme nach dieser Betrachtung zunächst nur ein Mischbetrieb mit zwei Bootloadern in Frage.

Allerdings gibt es noch eine Alternative, die tatsächlich alle Anforderung zu erfüllen scheint, aber in der Liste von Wikipedia leider fehlt. Es ist die Open-Source-Software „iPXE“, die im nächsten Kapitel noch etwas genauer betrachtet werden muss, ob sie sich als Basis für die neue Lösung eignet. Dabei spielen neben den funktionalen An-

16 online: WIKIPEDIA, 2017 (19.07.2017)

forderungen, auch nichtfunktionale Anforderungen, wie z. B. Releasezyklen, Konfigurationsaufwand, verfügbare Dokumentation und Support eine wichtige Rolle.

4.3.2 Der Netzwerk-Bootloader iPXE

iPXE ist eine Open-Source-Implementierung des PXE-Standards. Das Projekt entstand im Jahr 2010 aus „gPXE“, das seit 2011 nicht mehr aktiv weiterentwickelt wird[17]. iPXE bietet einen deutlich größeren Funktionsumfang wie PXELINUX oder GRUB2, da es z. B. Blockzugriff auf Speichernetzwerke über iSCSI, „Fibre Channel over Ethernet“ (FCoE) und „ATA over Ethernet“ (AoE) unterstützt. Durch die enthaltene Skriptsprache ist es möglich, dynamische Menüs zu generieren. Als weitere Besonderheit, kann die Software mit Hilfe spezieller Programme in den ROM von Netzwerkkarten transferiert werden und somit den vorhandenen UNDI-Treiber inkl. Netzwerkstack vollständig ersetzen. Die Unterstützung für die ARM-Architektur (32-Bit und 64-Bit) wurde im Mai 2016 integriert.

Für die Installation von iPXE empfiehlt es sich, immer den aktuellen Quellcode von der Webseite herunterzuladen und mit dem Linux-Programm „make“ zu übersetzen. Benötigte Funktionen können über Header-Dateien aktiviert und deaktiviert werden. Die Dokumentation des Programms ist sehr gut, so dass die Einarbeitungszeit für einen Administrator mit Linux-Kenntnissen recht kurz ausfallen dürfte. Zusätzliche Hilfe bietet ein Forum, ein Chat und die Mailing-Liste.

Das Zusammenspiel mit der B-API sollte sich durch die Unterstützung von HTTP in Kombination mit serverseitigem Scripting ohne größere Probleme realisieren lassen. Allerdings müssen die bisherigen Bootmenüs aufgrund einer anderen Syntax der Skriptsprache von iPXE komplett überarbeitet und angepasst werden. Zudem sind Kompatibilitätstests mit jedem vorhandenen Servermodell erforderlich. Da insgesamt betrachtet die Vorteile von iPXE jedoch überwiegen und derzeit kein anderer Bootloader alle Anforderungen erfüllt, wird iPXE der Ausgangspunkt für die weitere Planung sein.

4.4 Erforderliche Softwareanpassungen

4.4.1 DHCP-Server

Bei einem Wechsel auf iPXE müssen diverse Änderungen am DHCP-Server durchgeführt werden, da in der bisherigen Konfiguration nur ein PXE-Boot für PC-BIOS berücksichtigt wird. Zukünftig kommen durch die Unterstützung von PXE-Boot über UEFI (x86 und ARM64) zwei weitere Möglichkeiten hinzu. Damit existieren drei mögliche Bootpfade, die jeweils ein eigenes NBP benötigen, siehe Abbildung 6. Dieses wird weiterhin über TFTP ausgeliefert. Um die Architektur des anfragenden Clients

17 online: IPXE-FAQ, 2017 (19.07.2017)

zu ermitteln, können die in der RFC 4578[18] und RFC 5970[19] für PXE definierten Protokolloptionen am DHCP-Server ausgewertet werden. Der gesuchte Wert wird in der Option „Client System Architecture Type Option“ (Code 93 bei DHCPv4) hinterlegt. Die hierfür gültigen Werte werden von der IANA verwaltet und sind über deren Webseite[20] einsehbar.

```
...
option arch code 93 = unsigned integer 16;
...
if exists user-class and option user-class = "iPXE" {
        filename "http://192.168.0.1/api/v1/config";  ------>  Bootserver-API
} elsif option arch = 00:0b {
        filename "boot/ipxe/ipxe_arm64.efi";  ------>  PXE-Client (ARM64, UEFI)
} elsif option arch = 00:07 {
        filename "boot/ipxe/ipxe_x64.efi";    ------>  PXE-Client (x64, UEFI)
} else {
        filename "boot/ipxe/undionly.kpxe";   ------>  PXE-Client (x86, PC-BIOS)
      }
}
...
```

Abbildung 6 - Auszug einer möglichen DHCP-Konfiguration (ISC-DHCP)

Durch die Verwendung von iPXE als NBP ergibt sich eine weitere Besonderheit. Wird iPXE über TFTP geladen und ausgeführt, startet das Programm, im Gegensatz zu PXELINUX, eine erneute DHCP-Anfrage, wodurch eine Endlosschleife entsteht. Um diesem Verhalten entgegenzuwirken, sind laut der iPXE-Webseite zwei Ansätze möglich[21]:

- Abbruch der Endlosschleife über den DHCP-Server
- Verwendung eines eingebetteten Skripts

Für die hier zu entwickelnde Lösung wird die erste Möglichkeit bevorzugt, da ansonsten bei einer Änderung der URL zum Abrufen der Konfiguration das komplette iPXE für jede Architektur wieder neu übersetzt und ausgerollt werden muss. Für die erste Variante spricht weiterhin, dass die angebotenen vServer schon mit iPXE arbeiten und somit die festgelegte URL direkt abrufen können, ohne wie bisher PXELINUX zu laden.

Über die eingetragene URL erfolgt eine GET-Anfrage an die B-API, um die Konfiguration des Clients abzurufen. Ist keine Konfiguration hinterlegt, wird ein einheitliches Bootmenü ausgeliefert und durch den PXE-Client ausgewertet.

4.4.2 Rescue-System

Das Hetzner Resuce-System ist ein Live-Linux, das über PXE gestartet wird und zur Installation von Betriebssystemen sowie für Diagnose- und Wartungszwecke verwen-

18 online: RFC 4578, 2006 (19.07.2017)
19 online: RFC 5970, 2010 (19.07.2017)
20 online: IANA, 2017 (20.7.2017)
21 online: IPXE chainloading, 2017 (20.07.2017)

det wird[22]. Ein Wechsel des Bootloaders erfordert beim PXE-Boot mit PC-BIOS keine weiteren Anpassungen des bestehenden Linux-Kernels. Soll jedoch UEFI verwendet werden, ist zu prüfen, ob der Kernel mit dem „EFI Boot Stub“ übersetzt wurde. Der EFI-Stub kennzeichnet den Linux-Kernel als gültige EFI-Anwendung, so dass dieser von der UEFI-Firmware ausgeführt werden kann[23]. Dies gilt sowohl für die x86-, als auch für die ARM-Architektur. Weiterhin sollte das Root-Dateisystem von Linux das Programm „efibootmgr“ enthalten, um ggf. die Booteinträge im NVRAM bearbeiten zu können. Das Rescue-System wurde für die ARM-Architektur im Rahmen einer Studienarbeit durch den Autor bereits portiert und kann produktiv eingesetzt werden. Auch der x86-Kernel erfüllt alle Voraussetzungen, so dass am Rescue-System momentan keine weiteren Arbeiten notwendig sind.

4.4.3 Installimage

Mit dem von Hetzner Online zur Verfügung gestellten Tool „installimage“ können verschiedene Linux-Betriebssysteme mit wenigen Schritten auf einem Server installiert werden[24]. Dies kann sowohl automatisch, z. B. nach der Bestellung eines neuen Servers durch einen Kunden, als auch manuell geschehen. Die Anwendung besteht aus verschiedenen Shell-Skripten. Bei den erforderlichen Anpassungsarbeiten für die UEFI-Installation im Rahmen der Studienarbeit, kam es bei den Tests zu Problemen, die gegenwärtig zu Einschränkungen im produktiven Betrieb und folglich erhöhtem Support-Aufwand führen. Ursache ist primär das unterschiedliche Verhalten der UEFI-Firmware im Umgang mit den Booteinträgen.

Standardmäßig wird jeder Server vor der Zuweisung an einen Kunden so konfiguriert, dass die Netzwerkkarte der erste Booteintrag ist. Befindet sich keine Startkonfiguration für den Server im TFTP-Verzeichnis, erscheint ein Bootmenü. Erfolgt hier keine Interaktion durch den Nutzer, wird nach einem Timeout von fünf Sekunden der Bootloader beendet und zum nächsten Booteintrag, i. d. R. einem lokalen Datenträger, gewechselt und versucht vom diesem ein BS zu starten. In PXELINUX wird diese Funktion durch das Kommando „LOCALBOOT 0“ ausgelöst, unter iPXE durch „exit“. Dies funktioniert in der Regel problemlos bei einem PC-BIOS, bereitet aber unter UEFI nach wie vor größere Schwierigkeiten. So ist es bei schon früher durchgeführten Tests mit verschiedenen Serversystemen beider Architekturen vorgekommen, dass die Firmware, anstatt zum nächsten Booteintrag zu wechseln, stattdessen ihr eigenes Konfigurationsmenü anzeigte oder die EFI-Shell startete. Weiterhin änderte sich teilweise die Reihenfolge der Booteinträge nach erfolgter Installation des BS und anschließendem Neustart. Leider kann der Nutzer bei Problemen mit der

22 online: HETZNER - DokuWiki, 2017 (20.07.2017)
23 online: KERNEL.org, 2017 (20.07.2017)
24 online: HETZNER - DokuWiki, 2017 (21.07.2017)

Firmware kaum Einfluss nehmen und ist letztendlich darauf angewiesen, dass der Hersteller des Serversystems ihm eine möglichst fehlerfreie Version liefert.

4.4.4 HWCheck

„HWCheck“ ist ein von Hetzner Online entwickeltes Programm zur Hardwarediagnose und Inventarisierung von Serversystemen und besteht wie „installimage“ aus mehreren Shell-Skripten. Die Anwendung umfasst mehrere Einzeltests, wie bspw. einen Stresstest und eine Datenträgerüberprüfung, die nacheinander ausgeführt werden. Abschließend wird ein Bericht generiert, der über eine Webseite abrufbar ist. Für den Einsatz in der veränderten Produktionsumgebung, muss das Programm unter einer UEFI-Umgebung getestet und für die ARM-Architektur portiert werden.

4.4.5 Bootserver-API

4.4.5.1 Vorbetrachtungen

Durch die festgelegten Anforderungen sind diverse Änderungen an der B-API erforderlich. Um sich einen groben Überblick über die anstehenden Arbeiten zu verschaffen, hilft die Anfertigung einer Skizze, die das neue Funktionsprinzip, wie in Abbildung 7 ersichtlich, darstellt.

Dabei wird, im Gegensatz zur bisherigen Arbeitsweise, zunächst deutlich, dass der PXE-Client zukünftig direkt mit der B-API kommuniziert. Weiterhin erfordert der Verzicht auf das Abspeichern der Konfigurationsdaten in einer Textdatei, das Vorhandensein einer entsprechenden Datenbank. Darauf aufbauend müssen neue Ressourcen und Repräsentationen für die B-API definiert werden. Die bisherigen Bootmenüs sind umzustellen und die Konfiguration des Webservers muss geändert werden.

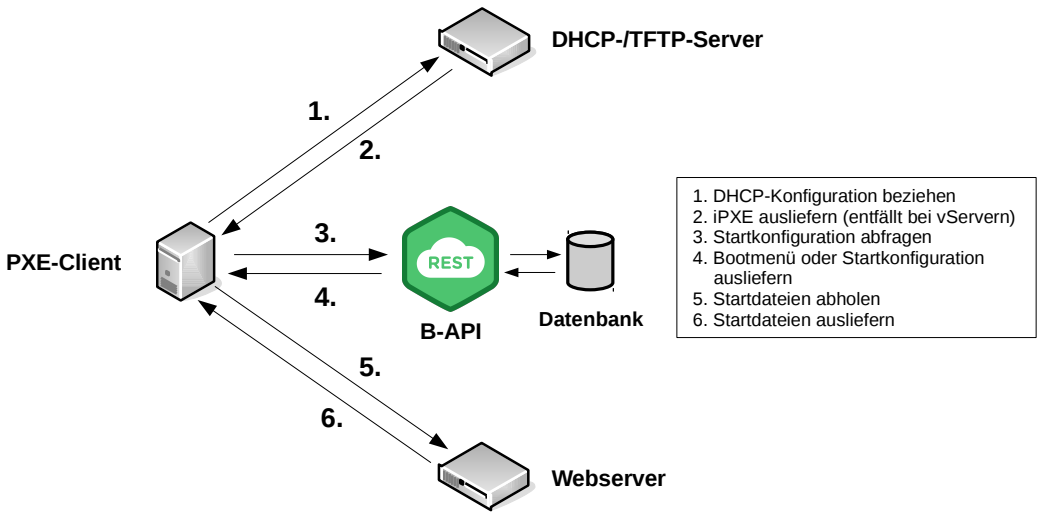


Abbildung 7 - Funktionsprinzip der neuen Lösung

4.4.5.2 Auswahl der Datenbank

Bei der Auswahl der Datenbank zur Speicherung der Startkonfigurationen sind verschiedene Gesichtspunkte zu berücksichtigen. So muss vorab geklärt sein, welche Daten generell zu speichern sind, welche Datenmenge grob zu erwarten ist und ob durch parallele Zugriffe auf den Datenbestand Transaktionen erforderlich werden. Ausgehend von diesen Überlegungen, kann aus einer großen Anzahl verfügbarer Datenbanken ausgewählt werden, die sich in die vier folgenden Kategorien einordnen lassen:

- Hierarchische Datenbanken
- Netzwerkdatenbanken
- Relationale Datenbanken
- Objektorientierte und NoSQL-Datenbanken

Jede einzelne Kategorie hat ihre Vor- und Nachteile, die bezogen auf den Anwendungsfall gegeneinander abzuwägen sind.

Die für die B-API notwendigen Daten zum Erstellen der Konfigurationsdateien bestehen aus einer Menge von Attributen mit entsprechenden Werten, die im Datenformat JSON verarbeitet und ausgetauscht werden. Dazu gehören bspw. die IP-Adresse des Servers, der Dateipfad für die jeweilige Startdatei im Verzeichnis des TFTP-Servers, ein Hash für das Anmeldepasswort und ein Zeitstempel. In der bisherigen Logik, wird in der B-API das für die Startkonfiguration notwendige JSON-Objekt in seine einzelnen Bestandteile zerlegt und anschließend in eine Textdatei geschrieben, die PXELINUX verarbeiten kann. Die Textdateien werden nicht dauerhaft vorgehalten, sondern nach einer Stunde über ein Shell-Skript gelöscht, wenn sie bis dahin nicht abgerufen worden sind.

Unter Beibehaltung dieser Logik sind daher mindestens die IP-Adresse und das zugehörige JSON-Objekt als Zeichenkette in der Datenbank zu speichern. Optional kommen die MAC-Adresse und ein Zeitstempel hinzu, der für das automatische Löschen von Einträgen benötigt wird. Erfolgt durch einen PXE-Client eine Anfrage an die B-API, wird in der Datenbank nachgesehen, ob für die anfragende IP-Adresse eine Startkonfiguration vorliegt. Ist dies der Fall, wird das JSON-Objekt in die für iPXE gültige Syntax aufbereitet und über HTTP als Text an den Client übertragen. Derzeit arbeitet die B-API bereits mit einem LDAP-Verzeichnis und einer Redis-Datenbank, die prinzipiell beide zum Abspeichern der Daten genutzt werden könnten.

Redis zählt zur Kategorie der NoSQL-Datenbanken und ist ein Key/Value-Store. Bei einem Key/Value-Store geschieht die Datenspeicherung in einem Paar aus Schlüssel und dazugehörigem Wert, welches durch den Schlüssel eindeutig identifizierbar ist.

Weiterhin arbeitet Redis als In-Memory-Datenbank, d. h. die Daten werden nur im Hauptspeicher des Computers abgelegt, wodurch die Verarbeitungsgeschwindigkeit im Vergleich zu relationalen Datenbanken sehr hoch ist. Persistenz lässt sich optional aktivieren. Redis ist sehr einfach zu konfigurieren und benötigt kein Schema zum Abspeichern der Daten. Weiterhin können Datensätze mit einer Ablaufdauer versehen werden, die das System automatisch löscht. Somit wäre Redis als Datenbank für die Konfigurationsdaten gut geeignet. Die IP-Adresse wird als eindeutiger Schlüssel verwendet und der Wert ist das JSON-Objekt, welches als Zeichenkette oder Hash gespeichert wird. Da die Redis-Datenbank in der jetzigen Konfiguration jedoch nur als lokaler Cache arbeitet, müsste für den gemeinsamen Zugriff von mehreren Standorten eine einfache Replikation bzw. ein Cluster eingerichtet werden. Redis unterstützt beide Varianten, wobei in einer Master/Slave-Konfiguration nur auf den Master geschrieben werden kann. Fällt der Master aus, kann ein Slave mit wenigen Befehlen im laufenden Betrieb zu einem Master gemacht werden, der dann mit den noch existierenden Slaves repliziert. In einem Cluster kann auf allen Instanzen geschrieben werden.

Nachteilig an dieser Variante ist, dass die Redis-Server für eine Replikation über mehrere Standorte im Internet erreichbar sein müssen und dementsprechend über eine Firewall abzusichern sind. Ferner gibt es im Unternehmen keine praktische Erfahrung im Umgang mit Redis-Clustern. Problematisch beim Einsatz von Redis unter Ubuntu-Linux ist auch die Tatsache, dass die notwendigen Installationspakete aus dem „Universe-Zweig“ kommen, für den es keine offizielle Unterstützung von Canonical (Entwicklerfirma von Ubuntu) gibt. Damit hat der Nutzer keine Garantie, dass Sicherheitsupdates rechtzeitig verfügbar sind. Notfalls muss der Quelltext selbst übersetzt werden.

Als zweite Möglichkeit, kann das bestehende LDAP-Verzeichnis für die DHCP-Leases zum Speichern der Konfigurationsdaten verwendet werden. LDAP ist ein Protokoll zum Zugriff auf Verzeichnisdienste, die zu den hierarchischen Datenbanken zählen. Die Speicherung der Daten erfolgt in einer Baumstruktur mit einem vorgegebenen Schema. Für den Einsatz von LDAP spricht, dass es sich im bisherigen Einsatz mit der B-API als sehr stabil, robust und leistungsfähig erwiesen hat. Zudem ist die Replikation schon eingerichtet. Allerdings muss beim Einsatz von LDAP das bestehende Schema angepasst oder neu erstellt werden, was für Einsteiger, die vorher noch nicht mit LDAP in Berührung gekommen sind, einen höheren Einarbeitungsaufwand bedeutet. Der Einsatz einer zusätzlichen relationalen Datenbank ist aufgrund der Einfachheit der Daten nicht sinnvoll und wäre ebenfalls mit einem erhöhten Konfigurationsaufwand verbunden, weil für den hier vorliegenden Anwendungsfall ein Datenbank-Cluster gebraucht wird.

Nach Abwägung aller Vor- und Nachteile wurde entschieden, das LDAP-Verzeichnis zur Speicherung der Konfigurationsdaten zu benutzen und die alternative Lösung mit Redis zunächst nicht weiter zu verfolgen.

4.4.5.3 Ressourcendefinition und Versionierung

Nach Auswahl der Datenbank werden die Ressourcen, ausgehend vom bisherigen Funktionsumfang der B-API und mit Hilfe der Dokumentation, die sich im Anhang 1 befindet, definiert. Daraus geht hervor, dass die B-API mit einem internen Versionsschema arbeitet. Die jeweilige Version wird über die URL angesprochen und ist aktuell „v1“. Beim Entwurf einer API sollte grundsätzlich immer ein Versionsschema verwendet werden, da sich hierdurch später notwendige Änderungen besser integrieren lassen und das Testen sowie die Integration in die Produktionsumgebung vereinfacht wird. So besteht nach dem Ausrollen der neuen Version bspw. keine Notwendigkeit, alle beteiligten Client-Anwendungen sofort umzustellen, da die bestehenden Funktionen zunächst weiter in der gewohnten Art und Weise nutzbar sind. Dies schafft einen gewissen Freiraum und Flexibilität, weil nicht immer davon ausgegangen werden kann, dass sich alle Client-Anwendungen im eigenen Verantwortungsbereich befinden und somit direkt umgestellt werden können. Zudem ist beim Auftreten größerer Probleme ein Wechsel auf die alte Version relativ unkompliziert möglich. Sind die Tests erfolgreich verlaufen und alle Anwendungen auf die neue Version umgestellt, können die alten Funktionen schrittweise entfernt werden. Damit bleibt der Quelltext übersichtlich und enthält keine unnötigen „Altlasten“.

Durch die neue Funktionsweise erhält die B-API bei der Implementierung die Version „v2“. Diese ist durch den Wegfall der TFTP-Funktionen nicht abwärtskompatibel zur Version „v1“. Tabelle 3 zeigt die geänderten Ressourcen.

Ressource	Operationen	Funktion
/api/v2/dhcp	GET, DELETE, POST	DHCP-Einträge im LDAP-Verzeichnis Abrufen, Löschen und Anlegen
/api/v2/bootconfig	GET	Konfiguration oder Bootmenü abrufen
/api/v2/bootconfig/request_cache	GET	Cache der Redis-Datenbank abrufen
/api/v2/bootconfig/commands	GET	Liste aller möglichen Kommandos abrufen
/api/v2/bootconfig/commands	DELETE	Konfiguration löschen
/api/v2/bootconfig/commands/active	GET	Alle gespeicherten Konfigurationen abrufen
/api/v2/bootconfig/commands/{command}	GET	Beschreibung des Kommandos abrufen
/api/v2/bootconfig/commands/{command}/config	POST	Konfiguration speichern
/api/v2/bootconfig/commands/{command}/info	POST	Konfiguration testen (ohne zu speichern)
/api/v2/bootconfig/protected	GET	Passwortgeschützte Bootmenüs abrufen

Tabelle 3 - Ressourcen der B-API Version „v2“

Das neue Ressourcenschema orientiert sich weitestgehend an der alten Version, um die spätere Umstellung der Client-Anwendungen in Bezug auf die Anpassung der API-Routen möglichst einfach zu gestalten. Die Ressource „ldap“ wurde in „dhcp“ umbenannt, weil zukünftig zwei Ressourcen auf das LDAP-Verzeichnis zugreifen und deren Funktionalität dadurch besser erkennbar ist. Die Ressourcen „wol“, „logs“ und „validate“ werden ohne Änderungen von der Version „v1“ übernommen.

4.4.5.4 Bootmenüs

Die bisherigen Bootmenüs müssen infolge einer anderen Syntax von iPXE gegenüber PXELINUX komplett umgestellt werden. Darüber hinaus gibt es bei iPXE zur Zeit noch keine Möglichkeit, versteckte Booteinträge zu erzeugen, sondern nur passwortgeschützte Menüs.

Den Einstiegspunkt bildet ein gemeinsames Bootmenü für die x86- und ARMv8-Architektur, das der PXE-Client auswerten kann. Weitere Untermenüs werden entsprechend zur Laufzeit nachgeladen. Zur Erzeugung der Menüs werden mit „Embedded Ruby“ (ERB)[25] generierte Vorlagen eingesetzt, die sich in separaten Dateien befinden, wodurch eine spätere Anpassung erleichtert wird. Die B-API lädt bei einer Anfrage die Vorlage, befüllt die entsprechenden Variablen mit konkreten Werten und sendet das Ergebnis per HTTP an den Client.

Die Realisierung der passwortgeschützten Menüs setzt eine serverseitige Authentifizierung voraus. Sowohl iPXE als auch das Grape-Framework unterstützen die über HTTP möglichen Verfahren Basis-Authentifizierung und Digest-Authentifizierung. Die Basis-Authentifizierung hat den Nachteil, das Benutzername und Passwort unverschlüsselt übertragen werden. Demnach können diese Informationen leicht von anderen Personen mitgelesen werden, wenn die HTTP-Verbindung nicht mit TLS gesichert ist. Da in der derzeitigen Umsetzung aus Sicherheitsgründen nur eine verschlüsselte Kommunikation über HTTPS mit der B-API möglich ist, sollte auch bei iPXE die Verschlüsselung aktiviert werden. Somit würde die Basis-Authentifizierung ausreichen. Eine Digest-Authentifizierung wäre nur dann notwendig, wenn Probleme bei der verschlüsselten Kommunikation zwischen iPXE und der B-API auftreten.

4.4.6 LDAP-Server

Durch die getroffene Entscheidung, das bestehende LDAP-Verzeichnis zum Speichern der Konfigurationsdaten zu verwenden, müssen einige Veränderungen an der derzeitigen Konfiguration der Server vorgenommen werden. Das LDAP-Verzeichnis für die DHCP-Server wird auf eigenständigen Servern betrieben und hat keine Verbindung zu den anderen LDAP-Servern in der Firma, die bspw. für die Authentifizierung der Benutzer zuständig sind. Das erforderliche Schema zum Anlegen der Ob-

25 https://apidock.com/ruby/ERB (26.07.2017)

jekte inkl. der zugehörigen Objektklassen wird bei der Installation des entsprechenden Programmpaketes, z. B. „isc-dhcp-server-ldap“ unter Ubuntu-Linux, bereitgestellt. In einer Standardkonfiguration befindet sich die entsprechende Organisationseinheit zum Speichern der DHCP-Objekte direkt unterhalb der Wurzel und verzweigt danach weiter, siehe Abbildung 8.

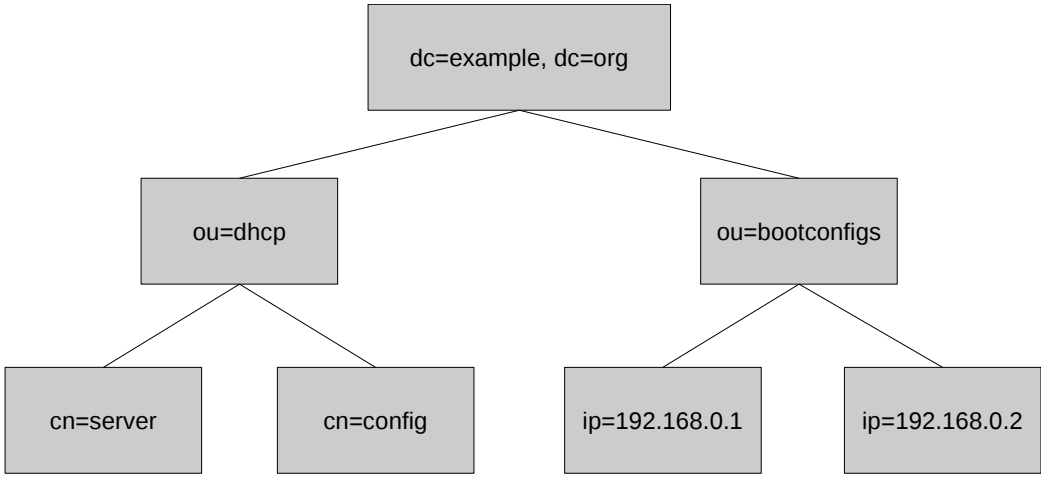


Abbildung 8 - Struktur des LDAP-Verzeichnisses

Zur Speicherung der Konfigurationsdaten wird eine weitere Organisationseinheit (OU) mit einem passenden Schema angelegt. Die OU bekommt den Namen „bootconfigs“ und liegt auf derselben Ebene wie die DHCP-Objekte. Im Schema wird eine neue Objektklasse gebildet, die die benötigten Attribute enthält. Die Attribute können selbst definiert oder aus den mitgelieferten Schemata von OpenLDAP übernommen werden.

4.4.7 Webserver

Da in der neuen Lösung die Startdateien größtenteils über HTTP übertragen werden, wird ein Webserver mit entsprechender Verzeichnisstruktur benötigt. Zur Durchführung von ersten Tests kann dazu der bereits auf den DHCP-Servern installierte Webserver NGINX verwendet werden. Er arbeitet gegenwärtig als Reverse-Proxy und leitet eingehende Anfragen an die B-API weiter.

4.5 Ausfallsicherheit und Skalierbarkeit

4.5.1 Vorüberlegungen

Da die Integration der neuen Lösung zu Veränderungen in der Produktivumgebung führt, muss bei der Planung darauf geachtet werden, dass die derzeit hohe Verfügbarkeit und Ausfallsicherheit auch nach den Veränderungen noch gegeben ist und wie sich beide Punkte ggf. noch weiter verbessern lassen. Damit verbunden sollte auch eine entsprechende Skalierbarkeit der Gesamtlösung mit Blick auf eine mögli-

che Erweiterung des Unternehmens an anderen Standorten relativ unkompliziert möglich sein. Im hier vorliegenden Fall betrifft dies die geplante Anbindung des neuen Datacenterparks in Finnland.

Als prinzipielle Herangehensweise empfiehlt es sich zunächst zu prüfen, welche Veränderungen durch die neue Lösung eintreten und welche Auswirkungen dadurch entstehen können. Dabei sind folgende Fragen hilfreich:

- Was ändert sich?
- Was bleibt gleich?
- Welche Ressourcen sind besonders stark ausgelastet?
- Gibt es einen „Single Point of Failure“ (SPOF)?

Durch die diese Überlegungen lassen sich danach entsprechende Maßnahmen ableiten, die hinsichtlich ihrer Vor- und Nachteile einzeln zu bewerten sind. Nachfolgende Überlegungen und Ansätze erfolgen anhand der bisher bekannten Informationen.

4.5.2 Auswirkungen auf die bestehende Infrastruktur

Ein TFTP-Server wird nach wie vor benötigt, um mit Ausnahme der virtuellen Server, das iPXE initial auszuliefern. Die Zahl der Anfragen wird wegen der stetig steigenden Menge von Servern längerfristig betrachtet nicht abnehmen, dafür aber das übertragene Datenvolumen. Die Verzeichnisgröße bzw. der Umfang der Dateien auf dem TFTP-Verzeichnis wird sich stark verringern, da sich ein Großteil der Dateien zukünftig auf dem Webserver befindet. Der tatsächliche Umfang lässt sich erst nach Durchführung der Kompatibilitätstest mit der Bestandshardware und Umstellung aller Bootmenüs exakt ermitteln. Die Zahl der Anfragen und das übertragene Datenvolumen wird auf dem für die Auslieferung der Startdateien verantwortlichen Webserver entsprechend hoch sein. Sollen mehrere Webserver zum Einsatz kommen, müssen die bereitgestellten Daten entsprechend synchronisiert werden. Die B-API ist momentan nur in Falkenstein aktiv und wird durch die direkte Kommunikation mit den PXE-Clients mit wesentlich mehr Anfragen konfrontiert als zuvor. Die LDAP-Server werden durch das Speichern und Löschen der Konfigurationsdaten stärker belastet als bisher. In der aktuell vorhandenen Konfiguration kann nur auf einen Master-Server geschrieben werden, wodurch es zu Engpässen kommen kann. Ein SPOF ist bezogen auf die Verfügbarkeit der einzelnen Serverdienste in der derzeitigen Infrastruktur nicht vorhanden. Jedoch würden beim kompletten Ausfall des Standorts Falkenstein größere Probleme auftreten, wenn z. B. der NFS-Cluster nicht mehr zur Verfügung steht.

4.5.3 Maßnahmen

Da der TFTP-Server mit Einführung der neuen Lösung nur noch wenige Dateien und Verzeichnisse enthält, besteht keine Notwendigkeit mehr, diese im zentralen DRBD-Cluster in Falkenstein bereitzustellen. Stattdessen wird für jeden Standort ein eigener TFTP-Server installiert. Dadurch ergeben sich zwei Vorteile. Die Latenz bei der Datenübertragung sinkt und alle Anfragen können im selben Standort beantwortet werden, weil zu diesem Zeitpunkt keine Verbindung mehr nach Falkenstein erforderlich ist. Der Eintrag „next-server“ ist in der jeweiligen DHCP-Konfiguration entsprechend zu ändern. Um die TFTP-Verzeichnisse an allen Standorten synchron zu halten, gibt es unter Linux mehrere Ansätze. Für den hier vorliegenden Fall reicht eine einfache Lösung, wie z. B. „lsyncd“[26]. Fällt der TFTP-Server an einem Standort aus, kann durch Anpassung der DHCP-Konfiguration problemlos auf einen anderen Standort gewechselt werden.

Um die größere Anzahl von Anfragen an die B-API und den Webserver problemlos verarbeiten zu können, wird pro Standort ein Loadbalancer-Cluster eingesetzt, siehe Abbildung 9.

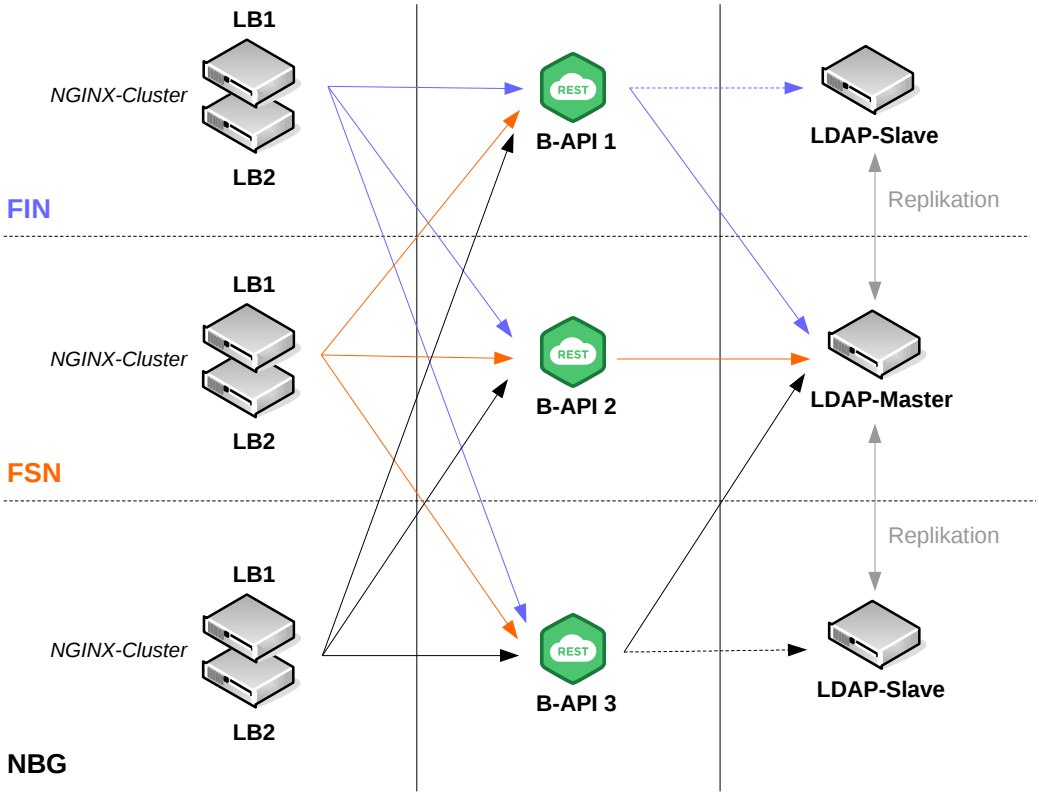


Abbildung 9 - Verteilung der Anfragen durch Loadbalancer

Dieser besteht aus zwei Servern, die mit NGINX[27], Corosync und Pacemaker realisiert sind. Auf dem Loadbalancer werden alle drei Standorte in die Konfiguration ein-

26 online: KITTENBERGER, 2017 (31.07.2017)
27 online: NGINX.org, 2017 (31.07.2017)

getragen, um die Last entsprechend dem ausgewählten Verfahren gleichmäßig zu verteilen. Dabei ist zu beachten, dass der Datenbestand auf allen Webservern jederzeit gleich sein muss. Dies kann mit verschiedenen Lösungsansätzen realisiert werden und hängt letztendlich davon ab, wie oft sich der Datenbestand ändert und ob die Clients nur Lesezugriff haben oder auch Schreibzugriff erlaubt ist. Ändern sich die Daten sehr häufig, empfiehlt sich das Verzeichnis für die Webserver über ein zentrales Speichersystem, z. B. mit iSCSI oder über eine Netzwerkfreigabe mittels CIFS bzw. NFS zugänglich zu machen. Denkbar wäre auch der Einsatz eines verteilten Dateisystems über alle drei Standorte, z. B. mittels „Ceph“[28]. Werden die Daten kaum verändert und durch die Clients nur abgerufen, ist es möglich, den Datenstand lokal auf jedem Webserver vorzuhalten und bei Änderungen gegenseitig zu replizieren. Durch die Befragung von Mitarbeitern in der Firma hat sich ergeben, dass über einen Monat betrachtet, nur wenige Änderungen im TFTP- und NFS-Verzeichnis durchgeführt werden. Somit können die Daten pro Webserver lokal gespeichert und zwischen den Standorten synchronisiert werden. Für den Standort in Finnland wird weiterhin ein zusätzlicher NFS-Cluster eingerichtet, um eine weitgehende Unabhängigkeit von Falkenstein sicherzustellen. Beide Cluster müssen sich ebenfalls gegenseitig replizieren.

Die derzeit bestehende Einschränkung, dass nur auf einen LDAP-Server schreibend zugegriffen werden darf, kann durch Aktivierung der „N-Way-Multi-Master“-Replikation, die seit Version 2.4 von OpenLDAP verfügbar ist, beseitigt werden. Allerdings ist diese Variante mit gewissen Risiken verbunden und bringt nicht nur Vorteile, wie in einigen Quellen nachzulesen ist. So wird im Buch von Oliver LIEBEL darauf hingewiesen, dass die Multi-Master-Replikation (MMR) dem X.500-Standard widerspricht, welcher Schreibzugriffe nur auf einem Master vorsieht und das jeder zusätzliche Master zu einem exponentiellen Anstieg des zu übertragenden Datenvolumens bei der Replikation führt[29]. Auch hätten sich einige OpenLDAP-Entwickler strikt gegen diese Funktion ausgesprochen. Die offizielle Dokumentation von OpenLDAP führt explizit Punkte auf, die gegen den Einsatz der MMR sprechen. Auch hier wird auf den Verlust der Garantie bzgl. der Datenkonsistenz gegenüber einer Single-Master-Lösung hingewiesen[30]. Der Maßnahmenkatalog des BSI zu Partitionierung und Replikation bei OpenLDAP empfiehlt auf den MMR zu verzichten, wenn die Integrität der Daten im Vordergrund steht[31]. Damit der MMR überhaupt ansatzweise in Betracht kommt, müssen die Zeituhren der Server dieselben Zeitserver benutzen, um synchron arbeiten zu können. Zusammenfassend betrachtet ist demnach eine Inkonsistenz des Datenbestandes das größte Risiko, was beim Aktivieren der MMR auftreten

28 online: RED HAT, Inc., 2017 (06.08.2017)
29 vgl. LIEBEL; UNGAR, 2009, S. 147-149
30 online: OPENLDAP Foundation, 2012, S. 170 (01.08.2017)
31 online: BSI, 2013 (30.07.2017)

kann, mit der Folge eines nicht mehr funktionierenden DHCP-Dienstes. Zwar wird die DHCP-Datenbank regelmäßig gesichert und es existieren Anleitungen zur Wiederherstellung eines LDAP-Verzeichnisses – eine Erfolgsgarantie dafür gibt es allerdings nicht. Somit wird diese Möglichkeit nicht in Betracht gezogen.

Eine denkbare Alternative könnte ein Wechsel des DHCP-Servers von ISC-DHCP zu dessen Nachfolger „Kea“[32] sein. Dieser unterstützt zur Speicherung der DHCP-Konfiguration gegenwärtig die Datenbanken MySQL, PostgreSQL und Cassandra, die alle auch in einem Cluster betrieben werden können. Dies garantiert die Ausfallsicherheit und den Zugriff von mehreren Kea-Servern. Gerade in Verbindung mit der NoSQL-Datenbank Cassandra ergeben sich interessante Möglichkeiten in Bezug auf die neue PXE-Infrastruktur. Cassandra ist eine Entwicklung von Facebook und wurde von Beginn an auf Skalierbarkeit und Robustheit ausgelegt[33]. Die interne Abfragesprache CQL besitzt eine gewisse Ähnlichkeit mit SQL. Für alle populären Programmiersprachen, darunter auch Ruby, existieren entsprechende Bibliotheken. Datensätze können, wie bei Redis, mit einer Ablaufdauer versehen werden. So könnten mit Cassandra alle notwendigen Daten für die DHCP-Server und die B-API in einer Datenbank gespeichert werden. Auch das Redis könnte entfallen. Für einen Produktivbetrieb sind jedoch ausführliche Tests notwendig, die im Rahmen dieser Bachelorthesis aus Zeitgründen nicht möglich sind. Ein großes Unternehmen, welches seine DHCP-Server erfolgreich von ISC-DHCP auf Kea umgestellt hat, ist Facebook[34].

4.6 Integration in die Produktivumgebung

Bis zur vollständigen Inbetriebnahme der neuen Lösung sind neben den Anpassungen an der B-API weitere Vorarbeiten notwendig, siehe Anhang 2. Ein großer Teil davon muss bereits vor Beginn der ersten Tests in der Produktivumgebung abgeschlossen sein, da die neuen Funktionalitäten zuerst nur bei Servern mit aktiviertem UEFI-Netzwerkstack überprüft werden. Dadurch ist gewährleistet, dass es zu keinen Einschränkungen im Produktivbetrieb kommt. Verläuft alles wie geplant, werden einzelne Testserver im Legacy-Mode überprüft. Die Tests sollten idealerweise bis zur Anbindung des Standorts in Finnland abgeschlossen sein, damit dort schon das neue Setup zum Einsatz kommen kann. Einige Punkte lassen sich durch entsprechenden Personaleinsatz parallel abarbeiten. Die durchzuführenden Anpassungen für die ARM64-Server können im Moment noch hinaus gezögert werden, da von Seiten der Geschäftsleitung noch keine Entscheidung über eine Produkteinführung getroffen wurde.

32 online: ISC Inc., 2017 (06.08.2017)
33 vgl. EDLICH; u.a., 2011, S. 93
34 online: STORAGE SERVERS, 2015 (31.07.2017)

5. Praktische Umsetzung

5.1 Vorgehensweise und Zielsetzung

Für die praktische Umsetzung stehen im Rahmen dieser Bachelorthesis drei Ziele im Vordergrund:

- Überprüfung der Kompatibilität von iPXE mit bestehender Hardware
- Erstellung des LDAP-Schemas für die Konfigurationsdaten
- Anpassung und Test der Bootserver-API

Der Kompatibilitätstest wird zuerst durchgeführt, weil die Ergebnisse direkten Einfluss auf die weitere Umsetzung des Projekts haben. Sind ein oder mehrere Servermodelle inkompatibel zu iPXE, muss eine Möglichkeit gefunden werden, die den weiteren Betrieb dieser Modelle in der Produktionsumgebung sicherstellt. Um die angedachte Funktionsweise der neuen Lösung praktisch zu überprüfen, werden anschließend die notwendigen Änderungen in der B-API implementiert und getestet sowie das LDAP-Schema erstellt.

5.2 Testumgebung

Alle durchzuführenden Tests werden in einer eigenen Testumgebung durchgeführt, um den laufenden Betrieb nicht zu beeinträchtigen. Der Aufbau ist in Abbildung 10 dargestellt. Zum Testen der API-Funktionalität und Verträglichkeit von iPXE mit bestehender Hardware, wird ein Server mit zwei NICs benötigt, der die notwendigen Dienste bereitstellt. Die zu überprüfenden Serversysteme werden über einen Switch mit der ersten Netzwerkkarte verbunden. Die zweite Netzwerkkarte stellt einen Internetzugang über das firmeneigene VPN bereit. Die API-Funktionalität kann unabhängig von der bestehenden Hardware und wahlweise auch in einer virtuellen Maschine getestet werden.

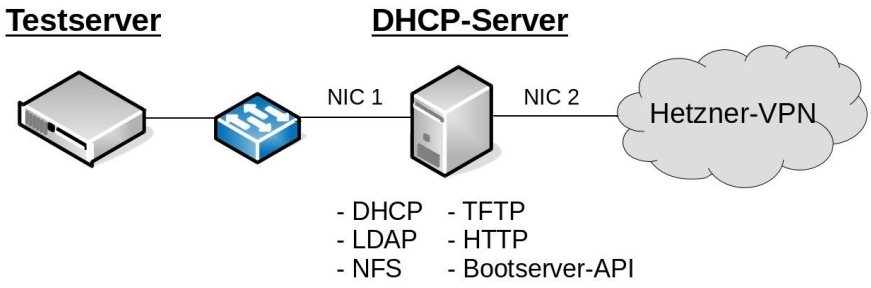


Abbildung 10 - Aufbau der Testumgebung

Als Betriebssystem für den DHCP-Server wird, wie in der Produktivumgebung, Ubuntu 16.04 verwendet. Alle zusätzlich installierten Softwarepakete befinden sich in Tabelle 4.

Name des Pakets	Funktion
isc-dhcp-server	ISC DHCP-Server
isc-dhcp-server-ldap	LDAP-Erweiterung für ISC DHCP-Server
hetzner-bootserver_api	B-API
nfs-kernel-server	NFS-Server
nginx	NGINX-Webserver
ruby	Ruby-Interpreter (für B-API)
slapd	OpenLDAP-Server
tftpd-hpa	TFTP-Server

Tabelle 4 - Installierte Softwarepakete auf dem Testserver

5.3 Kompatibilitätstest mit Bestandshardware

5.3.1 Testablauf

Durch den Kompatibilitätstest wird überprüft, ob alle derzeit in den RZs eingesetzten Serversysteme mit iPXE kompatibel sind. Dabei muss gewährleistet sein, dass der Bootloader startet, über DHCP eine IP-Adresse bezieht und das Bootmenü ordnungsgemäß angezeigt wird. Erfolgt keine Interaktion durch den Benutzer, muss beim automatischen Beenden des Bootloaders (Timeout) zum nächsten Booteintrag gewechselt werden. Danach wird getestet, ob das Rescue-System startet und die Installation eines Betriebssystems fehlerfrei möglich ist. Beim anschließenden Neustart darf die Bootreihenfolge nicht verändert werden. Bei Systemen mit einer UEFI-Firmware erfolgen die Tests zusätzlich mit aktiviertem UEFI-Netzwerkstack. Damit lässt sich feststellen, welches System später überhaupt für einen Betrieb mit UEFI freigegeben werden kann. Dieser Test ist durch die große Anzahl verschiedener Systeme sehr zeitaufwendig, muss aber infolge der bei früheren Test mit neueren Versionen von PXELINUX aufgetretenen Probleme unbedingt durchgeführt werden.

Vor Beginn des Tests sind die notwendigen iPXE-Dateien zu erstellen. Insgesamt werden drei unterschiedliche Versionen benötigt:

- „undionly.kpxe“ für PC-BIOS
- „snponly.efi“ für x86-Systeme mit UEFI-Firmware
- „snponly.efi“ für ARM64-Systeme mit UEFI-Firmware

Auf der Webseite von iPXE sind die Unterschiede der einzelnen Versionen und deren Erstellung erklärt[35]. Die Entscheidung „undionly.kpxe“ zu verwenden, erfolgt aus zwei Gründen. Einerseits wird bei dieser Variante der ursprüngliche PXE-Code des Herstellers entladen und durch den von iPXE ersetzt, was insbesondere bei einem fehlerhaft implementierten PXE-Stack älterer Hardware von Vorteil ist. Andererseits

35 online: IPXE buildtargets (02.08.2017)

bleibt der UNDI-Treiber des Herstellers aktiv, wodurch iPXE auch mit den neusten Netzwerkkarten verwendet werden kann. Die Variante „ipxe.pxe“ kann nur benutzt werden, wenn iPXE einen eigenen Treiber für die jeweilige NIC bereitstellt. In gleicher Weise greift „snponly.efi“ über NII oder SNP auf den bereits in der UEFI-Firmware oder im OpROM integrierten Netzwerktreiber zu. Wenn die Version für ARM64 auf einem x86-PC erstellt wird, muss ein Cross-Compiler inkl. der notwendigen Abhängigkeiten installiert sein. Der Befehl zum Übersetzen des Quelltextes lautet:

```
make CROSS_COMPILE=aarch64-linux-gnu- ARCH=arm64 bin-arm64-efi/snponly.efi
```

Neben den oben aufgeführten Dateien, wurde ein einfaches Bootmenü erstellt und die Konfiguration des DHCP-Servers, wie in Kapitel 4.4.1 beschrieben, angepasst. Das Bootmenü soll bei späteren Tests von der B-API automatisch erzeugt werden. Beide Dateien und das Testprotokoll befinden sich in Anhang 3 bis Anhang 5.

5.3.2 Auswertung der Testergebnisse

Zuerst wurden die Servermodelle der ersten Generation aus dem Jahr 2005 überprüft. Dort zeigte bereits der zweite Testserver nicht das gewünschte Ergebnis. Beim Beenden vom iPXE erschien die Fehlermeldung

```
Reboot and Select proper boot device
or Insert Boot media in selected Boot device and press a key
```

und das installierte BS startete erst nach Betätigung der ENTER-Taste.

Die Ursache dieses Verhaltens ist laut dem iPXE-Forum auf ein fehlerhaftes BIOS zurückzuführen, das den Interrupt (INT) 18h nicht richtig verarbeitet, der beim Beenden des iPXE-Stacks aufgerufen wird[36]. Dieser Interrupt wurde bei älteren IBM-kompatiblen PCs zum Aufruf des BASIC-Interpreters im ROM benutzt, wenn der Startvorgang von einer Diskette nicht erfolgreich war. Gemäß der „BIOS Boot Specification“ (BBS) wird in neueren BIOS-Versionen INT 18h als Fehlerbehandlungsroutine genutzt, wenn das Starten des BS von einem Datenträger oder einem OpROM fehlschlägt[37]. Der Aufruf von INT 18h bewirkt einen Rücksprung zu INT 19h, der nach Beendigung des POST beim Einschalten des Rechners die Startroutine auslöst. Eine Fehlermeldung wird laut BBS angezeigt, wenn von keinem Datenträger ein BS erfolgreich geladen werden konnte. Bei dem Mainboard des zweiten Testservers passiert dies offenbar schon, wenn über die NIC kein BS gestartet wird. Beim anschließenden Wechsel der iPXE-Version zu „undionly.kkpxe“, wo UNDI-Treiber und PXE-Stack des Herstellers benutzt werden, trat dieser Fehler nicht auf. Diese Version löst beim Beenden von iPXE keinen INT 18h aus, sondern gibt die Kontrolle an den PXE-Stack zurück. Gemäß BBS ist es bei der Implementierung eines OpROM möglich, die Rückgabe zum BIOS sowohl über INT 18h als auch über INT 19h zu realisieren.

36 online: IPXE-Forum, 2013 (12.08.2017)

37 online: COMPAG Corp; u.a, 1996, S. 29-31 (05.08.2017)

Demnach liegt die Vermutung nahe, dass bei dem Mainboard des zweiten Servers beim Beenden des OpROM standardmäßig INT 19h aufgerufen wird. Aufgrund dieser Erkenntnis wurden alle weiteren Servermodelle mit dieser Version getestet, was bis auf zwei Mainboards (siehe Testprotokoll im Anhang), auch erfolgreich war. Diese funktionierten nur mit der Version „undionly.kpxe“ einwandfrei. Warum bei diesen Mainboards die Rückgabe zum PXE-Stack nach Beendigung von iPXE zu einem „Einfrieren“ des Rechners führt, lässt sich nicht eindeutig nachvollziehen.

Bei dem Test mit aktiviertem UEFI-Netzwerkstack kam es ebenfalls bei den älteren Servermodellen zu Problemen. Dort konnte nach dem Start von iPXE keine Verbindung zur NIC hergestellt werden. Die Modelle der aktuellen Generation zeigten keine Auffälligkeiten. Auch der Wechsel zum nächsten Booteintrag erfolgte beim Verlassen von iPXE problemlos.

Bei den zwei zur Verfügung stehenden ARM64-Systemen konnte nur das Mainboard MT30-GS0 erfolgreich getestet werden. Beim zweiten Mainboard schlug nach dem Beenden von iPXE der Start des Rescue-Systems fehl. Dies ist auf einen Fehler in der Firmware zurückzuführen, der bis heute (Stand August 2017) noch nicht vom Hersteller beseitigt wurde.

5.3.3 Auswirkungen auf die geplante Lösung

Durch die aufgetretenen Probleme beim Kompatibilitätstest, ist die geplante Lösungsvariante nicht mehr so einfach umsetzbar wie bisher angenommen. Eine einfache Überprüfung auf die verwendete Architektur am DHCP-Server ist demnach nicht ausreichend, wenn, wie im vorliegenden Fall, nicht alle Mainboards mit ein und derselben iPXE-Version kompatibel sind. Erschwerend kommt hinzu, dass die betroffenen Mainboards aus jeweils unterschiedlichen Servergenerationen in nicht unerheblicher Stückzahl (jeweils mehr als 5000) nach wie vor im Einsatz sind und von Kunden aktiv genutzt werden. Somit scheidet die an sich denkbare Möglichkeit, die betroffenen Modelle einfach auszusondern, von vornherein aus, da das Unternehmen durch diese Maßnahme sichere Einnahmen und zufriedene Kunden verlieren würde. Ein BIOS-Update kann die Probleme lösen, ist aber wegen des Alters der Hardware nicht mehr möglich, da der Support für diese Mainboards ausgelaufen ist. Somit müssen andere Lösungsansätze gesucht werden. Dahingehende Überlegungen, am DHCP-Server noch eine weitere Eingrenzung vorzunehmen, gestalten sich schwierig, weil es bei der ersten DHCP-Anfrage eines PXE-Clients mit standardmäßigem OpROM, außer der MAC-Adresse, keinen weiteren Parameter gibt, der sich entsprechend filtern lässt, um für verschiedene Mainboards unterschiedliche Startdateien auszuliefern. Die DHCP-Option 60 (Vendor class identifier) ist bei allen betroffenen Mainboards gleich und auch die Option 97 (UUID/GUID) bietet keinen vernünftigen Ansatzpunkt, um bestimmte Mainboard-Typen gezielt zu unterscheiden.

Wäre die Anzahl der Server geringer, könnte man die betreffenden MAC-Adressen zum Filtern direkt auf dem DHCP-Server hinterlegen. Bei über 5000 Einträgen führt dies allerdings zu einer unübersichtlichen und großen Konfigurationsdatei, die sich bei sehr vielen Anfragen negativ auf die Geschwindigkeit auswirkt. Wenn iPXE schon bei der ersten DHCP-Anfrage zum Einsatz kommt, können bestimmte DHCP-Optionen gezielt überschrieben werden, was die Eingrenzung am DHCP-Server dann sehr einfach macht. Dazu muss sich iPXE aber direkt im OpROM der betreffenden NIC befinden. Das Flashen einer Onboard-NIC ist jedoch weitaus komplizierter, als bei einer PCIe-Steckkarte, und wegen der hohen Stückzahl schlichtweg nicht durchführbar. Hat der betreffende Server über TFTP initial eine für ihn inkompatible iPXE-Version bekommen, kann die funktionierende Version nur durch anschließendes Nachladen ausgeliefert werden. Hierzu kann z. B. über iPXE mit Hilfe eines Skripts der Name des Mainboards über das SMBIOS ausgelesen und mit der aktiven iPXE-Version verglichen werden. Denkbar wäre auch den Vorgang über die B-API zu steuern. Jedoch ist dieses Nachladen insgesamt betrachtet nicht besonders komfortabel, wenn eine neue Lösung eingeführt werden soll.

Um das Problem effektiv zu beseitigen, muss an der Stelle angesetzt werden, wo die Anfrage an den TFTP-Server stattfindet, um das NBP abzuholen. Eine Möglichkeit könnte ein alternativer und quelloffener TFTP-Server sein, der so angepasst wird, dass die anfragende IP-Adresse nochmals mit einer Datenbank verglichen wird, um zu ermitteln, welcher Server bzw. welches Mainboard sich dahinter verbirgt. Ist die angefragte Startdatei inkompatibel zum betreffenden Servermodell, wird stattdessen die kompatible Version übertragen. Die benötigten Informationen, wie IP-Adresse und Servertyp, können über den „Hetzner-Robot“ mittels einer GET-Anfrage abgerufen und verglichen werden.

Für die weiteren Tests wird die Variante mit einem zusätzlichen Skript, siehe Anhang 6, verwendet, welches nach Prüfung des Mainboardtyps die kompatible iPXE-Version nachlädt. Damit kann bei ersten Tests in der Produktivumgebung ebenfalls verhindert werden, dass die vServer direkt bei der B-API anfragen.

In der DHCP-Konfiguration ist, anstatt der URL für die B-API, das Startskript einzutragen. Diese Konfiguration muss am Standort in Finnland nicht verwendet werden, da dort nur neue Hardware zum Einsatz kommt, die mit iPXE kompatibel ist.

5.4 Erweiterung des LDAP-Verzeichnisses

Für die Änderung des LDAP-Verzeichnisses, müssen zunächst zwei Dateien angelegt werden. Als Erstes wird die nachfolgende Schemadatei mit den Attributen und Objektklassen, wie in Kapitel 4.4.5.2 und Kapitel 4.4.6 festgelegt, erstellt.

```
# bootconfig.schema
# Schema for boot configuration entries
# Depends upon core.schema, cosine.schema, dhcp.schema


 attributetype ( 1.1.3.5.1 NAME 'ip'
      DESC 'Host IPv4 address'
      EQUALITY caseIgnoreIA5Match
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )

 attributetype ( 1.1.3.5.2 NAME 'mac'
      DESC 'Host MAC address'
      SUP dhcpHWAddress )

 attributetype ( 1.1.3.5.3 NAME 'timestamp'
      DESC 'Timestamp when entry was created'
      EQUALITY integerMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )

 objectclass ( 1.1.3.7.1 NAME 'host' SUP top STRUCTURAL
      DESC 'Host with IP address, MAC address,
            configuration string and timestamp'
      MUST ( ip $ description )
      MAY  ( mac $ timestamp ))
```

Jeder Attributtyp besteht aus einem eindeutigen Bezeichner (OID) in Form einer Nummer und einem Namen (NAME). Hinzu kommen die Beschreibung (DESC), der eigentliche Datentyp (SYNTAX) und die Vergleichsregeln (EQUALITY). Ausführliche Informationen zu diesen Direktiven befinden sich in den RFC 2252[38] und RFC 4517[39]. Über „SUP“ können die Eigenschaften eines Attributtyps aus einem bereits vorhandenen Schema übernommen werden. In diesem Fall wird auf das Attribut „dhcpHWAddress“ im DHCP-Schema zurückgegriffen. Die Bezeichner „MUST“ und „MAY“ legen in der Objektklasse fest, ob die entsprechenden Attributtypen beim Anlegen einen Wert enthalten müssen oder nicht. Dies ist vergleichbar mit zulässigen „NULL-Werten“ in einer relationalen Datenbank.

Zum Erzeugen der OU „bootconfigs“ wird eine weitere Datei im LDIF-Format benötigt.

```
# bootconfigs.ldif
# create bootconfigs container in existing tree

dn: ou=bootconfigs,dc=example,dc=org
ou: bootconfigs
objectClass: organizationalUnit
description: Bootconfigs
```

Das „LDAP Data Interchange Format“ (LDIF) beschreibt das Datenaustauschformat zum Lesen und Ändern von Einträgen in einem LDAP-Verzeichnis. Um die Änderungen auf dem LDAP-Server zu testen, muss im ersten Schritt die Schemadatei ins

38 online: RFC 2252, 1997 (13.08.2017)
39 online: RFC 4517, 2006 (13.08.2017)

LDIF-Format konvertiert werden. Dazu wird eine Konfigurationsdatei „bootconfig.conf“ angelegt.

```
# bootconfig.conf
include /etc/ldap/schema/dhcp.schema
include /etc/ldap/schema/bootconfig.schema
```

Mit den folgenden Anweisungen auf der Linux-Konsole erfolgt die Umwandlung und Einbindung der Schemadatei, sowie das Erstellen der OU „bootconfigs“.

```
$ slaptest -f /tmp/bootconfig/bootconfig.conf -F /tmp/bootconfig

$ ldapadd -x -H ldap://openldap -D cn=admin,cn=config -w config -f \
bootconfig.ldif

$ ldapadd -x -H ldap://localhost -D cn=admin,dc=example,dc=org -w admin \
-f bootconfigs.ldif
```

Zum Testen der vorgenommenen Änderungen wird eine LDIF-Datei mit einer Beispielkonfiguration erstellt und wieder über das „ldapadd“-Kommando zum Verzeichnis hinzugefügt.

```
# host.ldif
# create single entry, ou "bootconfigs" must already exist

dn: ip=10.17.4.71,ou=bootconfigs,dc=example,dc=org
objectClass: host
ip: 10.17.4.71
description: {ip: "10.17.4.71", ip6: "fe80::d250:99ff:fe32:d949/64",
ip6_gateway: "fe80::1",  command: "startrescue64", password: "default",
created_at: "2017-07-27 09:00:03 +0200"}
```

Mit dem folgenden Befehl lässt sich der angelegte Eintrag abrufen.

```
$ ldapsearch -x -D cn=admin,dc=example,dc=org -b \
'ou=bootconfigs,dc=example,dc=org' -w admin '(ip=10.17.4.71)'
```

5.5 Anpassung der Bootserver-API

5.5.1 Vorbetrachtungen

Die B-API wurde mit dem Framework „Grape“ erstellt, welches in der Programmiersprache Ruby geschrieben ist. Mit Grape können Web-APIs sehr komfortabel und vergleichsweise einfach entwickelt werden, ohne dabei ein komplettes Web-Framework wie bspw. „Ruby on Rails“[40] verwenden zu müssen. Dies wird u. a. durch die integrierte „Domain Specific Language“ (DSL) unterstützt. Eine DSL dient primär dazu, die Syntax einer Programmiersprache so zu vereinfachen, das Anweisungen für eine bestimmte Aufgabe mit möglichst einfachen und aussagekräftigen Konstrukten geschrieben werden können.

40 http://rubyonrails.org (17.08.2017)

Obwohl es in Grape nicht vorgeschrieben ist, die Quelltextdateien nach einem bestimmten Schema zu strukturieren, sollte dies der Entwickler zu Beginn der Programmerstellung aus Gründen der Übersichtlichkeit und Wartbarkeit stets durchführen. Die Struktur der B-API, siehe Anhang 7, orientiert sich an dem Entwurfsmuster MVC (Model-View-Controller), welches 1979 durch Trygve Reenskaug für Benutzeroberflächen in Smalltalk geschaffen wurde[41].

Im Model wird die Verbindung zu einer Datenbank und alle zulässigen Operationen auf die Daten, auch als Geschäftslogik bezeichnet, definiert. Der Controller verarbeitet die eingehenden Anfragen, die entweder durch den Benutzer oder ein anderes Programm ausgelöst werden und interagiert dabei mit dem Model. Die View stellt die Benutzeroberfläche dar, über die der Nutzer mit der Anwendung kommuniziert. Das Zusammenspiel aller Bestandteile zeigt Abbildung 11. Im Gegensatz zu normalen Web-Anwendungen, müssen APIs nicht zwingend eigene Views besitzen, weil die Kommunikation häufig von Maschine zu Maschine stattfindet.

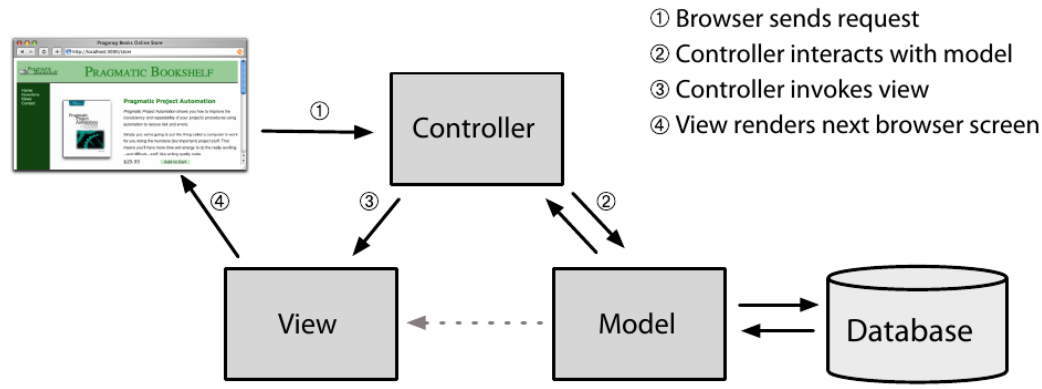


Abbildung 11 - MVC-Entwurfsmuster
(RUBY; u.a., 2016, S. 40)

5.5.2 Versionierung

Für die Erstellung der neuen Version „v2“, wird das Controller-Verzeichnis der B-API in zwei Teile untergliedert und der Ordner „view“ hinzugefügt, wo sich später die Vorlagen für die iPXE-Menüs befinden.

```
...
  lib/
    hetzner/
      bootserver_api/
        controller/
          v1/
            ...
          v2/
            ...
          model/
```

41 online: WIKIPEDIA, 2017 (17.08.2017)

```
        view/
          ...
...
```

Models werden i. d. R. nicht versioniert, da die zugrunde liegende Datenbank und die Zugriffsmethoden versionsübergreifend benutzt werden. Bei Erweiterung des Datenbankschemas, sollte die Kompatibilität zu bestehenden Anwendungen möglichst erhalten bleiben.

5.5.3 Anpassungen am Model

Im Model-Ordner befinden sich vier Klassendateien. Die Klasse „ldap.rb“ enthält die Methoden für den Zugriff auf das LDAP-Verzeichnis zum Anlegen und Ändern der DHCP-Leases sowie die notwendige LDIF-Vorlage. Mit Hilfe der Klasse „wol.rb“ kann ein Server über die B-API aus der Ferne gestartet werden. In „validator.rb“ wird geprüft, ob eine IP-Adresse zum internen VPN gehört oder aus dem öffentlichen Netzwerk stammt. Diese Funktion ist in der B-API nicht mehr notwendig, da die Prüfung seit einiger Zeit durch die Anwendung „RZ-Admin“ (Frontend) selbst vorgenommen wird. In den Klassen „ldap.rb“ und „wol.rb“ sind für die Version „v2“ keine Anpassungen notwendig.

Die Klasse „tftp.rb“ beinhaltet alle Methoden für die Kommunikation mit der Redis-Datenbank, zum Serialisieren der JSON-Objekte und zum Anlegen und Löschen der Textdateien für PXELINUX. Da sich diese Funktionalität in der neuen Version ändert, wird eine neue Klassendatei „bootconfig.rb“ angelegt.

Die vorhandenen Methoden zur Generierung der Inhalte für die Textdateien werden soweit wie möglich übernommen und so angepasst, das die erzeugten Einträge der korrekten Syntax für iPXE entsprechen. Im Unterschied zur bisherigen Funktionsweise wird der entstehende Text jedoch nicht mehr abgespeichert, sondern im zugehörigen Controller direkt an den Client per HTTP übermittelt.

Die Methoden für den Zugriff auf die Konfigurationsdaten im LDAP-Verzeichnis werden neu erstellt. Als Hilfe dient das bisherige Modell „ldap.rb“ für die DHCP-Einträge. Tabelle 5 zeigt die neu erstellten und geänderten Methoden sowie deren Funktion.

Model „bootconfig.rb“	
Name	**Funktion**
neue Methoden	
def ldif_host (ip: nil, mac: nil, description: nil)	LDIF-Vorlage für Konfigurationseintrag
def add (**kwargs)	Konfiguration im LDAP-Verzeichnis anlegen
def get (ip: nil, mac: nil)	Konfiguration aus dem LDAP-Verzeichnis abrufen
def all	alle gespeicherten Konfigurationen abrufen
def delete (ip: nil, mac: nil)	Konfiguration aus dem LDAP-Verzeichnis löschen

def conn	Enthält die Parameter zum Herstellen der Verbindung mit LDAP-Server
geänderte Methoden	
def self.all (type: ipxe)	alle aktiven Konfigurationen anzeigen
def self.delete ip	Konfiguration löschen
def append	erforderliche Parameter für die Kommandozeile des Linux-Kernels ermitteln
def command	Zusatzinformationen für Booteintrag ergänzen
def write!	Konfiguration an iPXE-Client ausliefern

Tabelle 5 - Geänderte und neue Methoden im Model „Bootconfig“

5.5.4 Erstellung der Bootmenüs (Views)

Vergleicht man eine klassische MVC-Anwendung mit der B-API, sind die Bootmenüs für iPXE als Views zu betrachten. Die Bootmenüs werden bei einer Anfrage von iPXE an die B-API immer dann ausgeliefert, wenn keine Startkonfiguration für den jeweiligen Server im LDAP-Verzeichnis gespeichert ist. Abbildung 12 zeigt das in der Testumgebung verwendete Startmenü. Vorhandene Untermenüs können über das iPXE-Kommando „chain“ direkt vom Webserver nachgeladen werden, sofern sie nicht passwortgeschützt sind.

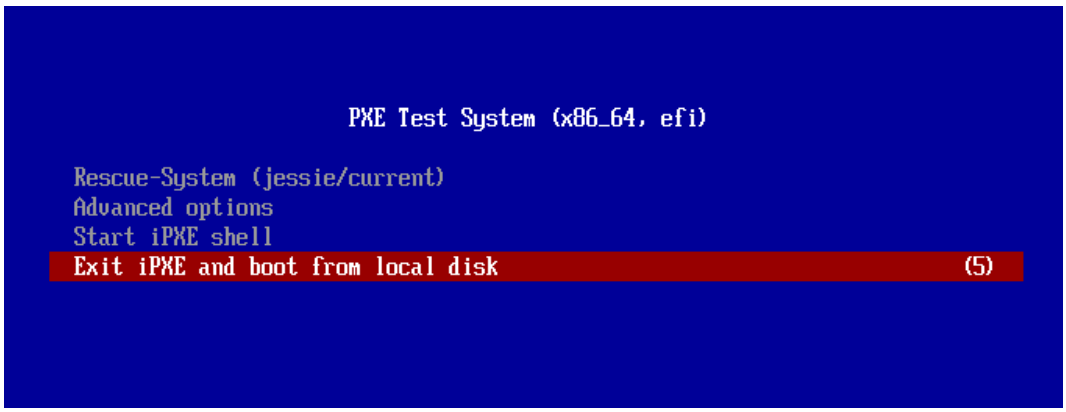


Abbildung 12 - iPXE-Startmenü in der Testumgebung

Das Bootmenü besteht aus einer Vorlage (engl. template) mit entsprechenden Platzhaltern. Die Vorlage wird zur Laufzeit im Controller geladen und die Platzhalter mit konkreten Werten aus den definierten Variablen ersetzt. Anschließend wird das Menü zum PXE-Client als Text übertragen. Alle Vorlagen sind im Ordner „view“ gespeichert und besitzen die Dateiendung „erb“ für „Embedded Ruby“.

Platzhalter lassen sich gut einsetzen, wenn in verschiedenen Menüeinträgen gleiche Informationen vorhanden sind. Dies kommt in den derzeit verwendeten Menüs für PXELINUX in der Produktivumgebung recht häufig vor. Nach einer ersten Analyse lassen sich Platzhalter für die folgenden Informationen verwenden:

- Passwort-Hash für Rescue-Linux (HASH=...)
- Basisadresse des NFS-Servers (nfsdir=...)

Da sich die Syntax zur Erzeugung der Menüs von iPXE gegenüber PXELINUX unterscheidet, müssen die Einträge anders aufgebaut werden, so dass noch weitere Platzhalter hinzukommen:

- MAC-Adresse der NIC (BOOTIF=...)
- Basisadresse des Webservers (http://...)

Der folgende Textblock zeigt die Unterschiede zwischen iPXE und PXELINUX für einen Booteintrag:

```
#PXELINUX-Eintrag

LABEL rescue_x86
MENU LABEL ^3.  -> 64-Bit Rescue-System
KERNEL rescue-x64/4.10.3-64/vmlinuz
APPEND initrd=rescue-x64/4.10.3-64/initramfs.cpio.xz \
nfsdir=192.168.0.1:/nfs RFILE=rescue64-dev.ext2 vga=792 HASH={secret} quiet
IPAPPEND 2


#iPXE-Eintrag

:rescue_x86
kernel http://192.168.0.1/rescue-x64/4.10.3-64/vmlinuz \
initrd=initramfs.cpio.xz nfsdir=192.168.0.1:/nfs RFILE=rescue64-dev.ext2 \
BOOTIF=${net0/mac} vga=792 HASH={secret} quiet
initrd http://192.168.0.1/rescue-x64/4.10.3-64/initramfs.cpio.xz
boot || goto failed
```

Das Kommando „IPAPPEND 2“ bewirkt unter PXELINUX, dass die MAC-Adresse der NIC, von der das BS gestartet wurde, zur Kommandozeile für den Linux-Kernel automatisch hinzugefügt wird (BOOTIF={mac}). Diese Angabe wird beim Starten des Rescue-Linux ausgewertet. Da es keinen vergleichbaren Befehl unter iPXE gibt, muss diese Information von Hand hinzugefügt werden. Durch die Verwendung von HTTP, ist vor der auszuführenden Datei die URL des Webservers anzugeben.

In der ERB-Datei sieht der Booteintrag dann wie folgt aus:

```
:rescue_x86
kernel <%= BASE_URL %>rescue-x64/4.10.3-64/vmlinuz initrd=initramfs.cpio.xz
nfsdir=<%= NFSDIR %> RFILE=rescue64-dev.ext2 BOOTIF=<%= BOOTIF %> vga=792
HASH=<%= RESCUE_HASH %> quiet
initrd <%= BASE_URL %>rescue-x64/4.10.3-64/initramfs.cpio.xz
boot || goto failed
```

Zum Anzeigen des kompletten Bootmenüs ist im Controller bei der entsprechenden Ressource die folgende Anweisung in der get-Methode zu verwenden.

```
ERB.new(File.read('.../bootserver_api/view/start.erb')).result(binding)
```

5.5.5 Anpassungen am Controller

In der gegenwärtigen Struktur der B-API gibt es je einen Controller für die LDAP-, TFTP- und WOL-Funktionalität sowie einen Basis-Controller (bootserver_api.rb), indem alle einzelnen Klassen eingebunden sind. Der erste Controller muss immer von der Klasse „Grape::API“ abgeleitet werden und erbt dessen öffentliche Attribute und Methoden, siehe Abbildung 13.

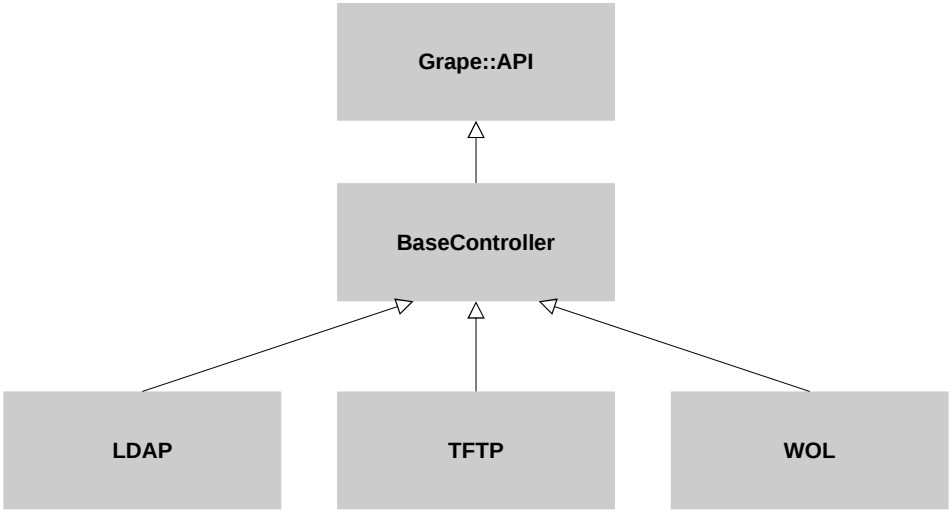


Abbildung 13 - Klassendiagramm der Bootserver-API (Version „v1“)

Um auf das jeweilige Model zuzugreifen, wird im zugehörigen Controller die Anweisung „require“ inkl. dem Dateipfad verwendet. Die Verbindung zwischen einer Ressource und der zugehörigen URL wird über das Schlüsselwort „namespace“ oder „resource“ hergestellt. In diesem Namensraum können dann durch die vorgegebenen Standardmethoden, wie z. B. „get“ oder „delete“, die gewünschten Operationen auf den Daten, je nach Art der Anfrage, durchgeführt werden. Nachfolgend ist ein einfaches „Hello World“-Beispiel dargestellt.

```
# helloworld.rb

require 'grape'

class HelloWorld < Grape::API
  prefix 'api'
  version 'v1'

    namespace 'test' do

      get do
        "Hello World.\n"
      end

    end

end
```

Zum Testen der Funktionalität kann die zugehörige URL mit dem Programm „curl“[42] auf der Kommandozeile wie folgt aufgerufen werden:

```
curl http://localhost:9292/api/v1/test
```

Der notwendige Webserver wird durch das Grape-Framework beim Ausführen der Anwendung automatisch gestartet.

Durch die Einführung der neuen API-Version muss die bisherige Klassenstruktur für die Controller geändert werden. Dazu wird vom bisherigen „BaseController“ je ein Basis-Controller pro Version abgeleitet und von diesem wiederum die zugehörigen Controller für die jeweilige Version. Abbildung 14 zeigt das neue Klassendiagramm.

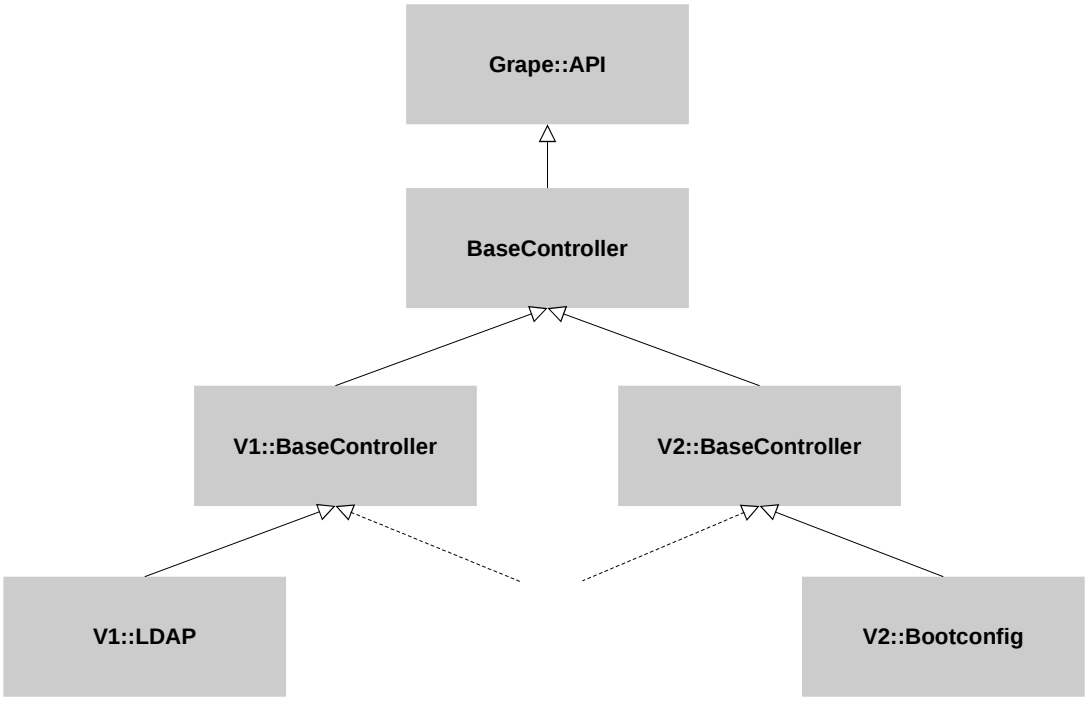


Abbildung 14 - Klassendiagramm der Bootserver-API (Version „v2“)

Die bisherigen Controller „ldap.rb“ und „wol.rb“ werden unverändert in die Version „v2“ übernommen, da deren Funktionalität weiterhin gebraucht wird. Lediglich der Name der Ressource wird im LDAP-Controller „v2“ gemäß der Ressourcendefinition aus Kapitel 4.4.5.3 in „dhcp“ geändert.

Um die neuen Ressourcen und Funktionen zu integrieren, wird ein neuer Controller mit dem Namen „Bootconfig“ erstellt, der von der Klasse „V2::BaseController“ abgeleitet ist. Die Struktur des neuen Controllers orientiert sich am bisherigen TFTP-Controller, da die bestehenden Ressourcen und ihre Funktionalitäten in der neuen Version weiterhin gebraucht werden. Die Bezeichnung des ersten Namensraums wird gemäß der Ressourcendefinition von „tftp“ auf „bootconfig“ geändert. Die zugehörigen Methodenaufrufe und ihre Beschreibungen (Schlüsselwort „desc“) innerhalb der Namensräume werden so angepasst, dass sie mit dem zugehörigen Model „Bootconfig“

42 https://curl.haxx.se (21.08.2017)

korrespondieren. Die neu zu erstellenden Methoden und ihre Funktion sind in Tabelle 6 dargestellt.

Controller „bootconfig.rb“		
Namespace	**Methode**	**Funktion**
bootconfig	get do	Konfiguration aus LDAP-Verzeichnis abrufen
		Konfiguration im LDAP-Verzeichnis löschen
		Bootmenü ausliefern
bootconfig/protected	http_basic do |….|	Überprüfung von Benutzername und Passwort für geschützte Bootmenüs
	get do	Geschütztes Bootmenü ausliefern

Tabelle 6 - Neue Methoden im Controller „Bootconfig“

Im Anhang 8 ist der zugehörige Algorithmus bei der Anfrage eines iPXE-Clients als PAP dargestellt.

Die Anfrage an die B-API wird durch das Startskript, welches der Client vom DHCP-Server abholt, an die folgende URL durchgeführt:

```
http://192.168.0.1/api/v2/bootconfig/?arch=${buildarch}&mac=${net0/mac}\
&ip=${net0/ip}
```

Dabei werden die Architektur (arch), die MAC-Adresse (mac) und die IP-Adresse (ip) als zusätzliche Parameter übergeben.

Passwortgeschützte Menüs werden im iPXE-Skript über das „login“-Kommando realisiert. Der eingegebene Benutzername und das Passwort werden über eine URL an die B-API übertragen. In der Methode „http_basic do“ wird aus dem Passwort ein Hashwert gebildet und mit dem auf dem Server gespeicherten Wert verglichen.

```
login
http://${username:uristring}:${password:uristring}@192.168.0.1/api/v2/\
bootconfig/protected/?menu=extended1
```

Der Parameter „menu“ gibt die Bezeichnung des angeforderten Menüs an. Ist das Passwort korrekt, wird das Menü wieder im Controller über eine Vorlage erzeugt und zum Client übermittelt. War das Passwort falsch, bekommt der Nutzer eine Fehlermeldung angezeigt.

Das automatische Löschen von nicht abgerufenen Konfigurationseinträgen wird über ein Linux-Skript realisiert, welches über das „ldapdelete“-Kommando alle noch vorhandenen Konfigurationseinträge im LDAP-Verzeichnis löscht. Das Skript wird jede volle Stunde über den Linux-Dämon „crond“[43] gestartet.

Am Ende werden die Basis-Controller der alten und neuen Version im übergeordneten Controller eingebunden.

43 https://linux.die.net/man/8/crond (27.08.2017)

5.5.6 Test der Änderungen

Beim Test der durchgeführten Anpassungen ist darauf zu achten, dass durch die Änderung der Klassenstruktur sowohl die ursprüngliche Version, als auch die neue Version, fehlerfrei funktionieren müssen. Um die Version „v1“ zu testen, können die bereits vorhandenen Tests genutzt werden. Für die ersten Tests mit der Version „v2“, wird das Kommandozeilenprogramm „curl“ verwendet. Folgende Testfälle sind für die neue Version zu prüfen:

- DHCP-Eintrag im LDAP-Verzeichnis abrufen
- DHCP-Eintrag im LDAP-Verzeichnis anlegen
- DHCP-Eintrag im LDAP-Verzeichnis löschen
- Anzeige des standardmäßigen Bootmenüs, wenn keine Konfiguration im LDAP-Verzeichnis vorhanden ist
- Startkonfiguration im LDAP-Verzeichnis abrufen
- Startkonfiguration im LDAP-Verzeichnis anlegen
- Startkonfiguration im LDAP-Verzeichnis löschen
- Passwortgeschütztes Bootmenü abrufen

Die Tests konnten nicht erfolgreich durchgeführt werden, da sich die B-API nach den durchgeführten Änderungen auf dem Testserver nicht mehr ordnungsgemäß starten ließ. Die angezeigte Fehlermeldung ist dem Anhang 9 zu entnehmen.

Anscheinend wird der übergeordnete Basis-Controller nicht gefunden, obwohl die Syntax zum Ableiten der Klasse korrekt ist und die zugehörige Klassendatei mit „require“ eingebunden wurde. Die Grape-Dokumentation[44] lieferte hierzu keine brauchbaren Informationen. Bei den im Rahmen einer durchgeführten Recherche im Internet gefundenen Beispielen, siehe URLs im Anhang 10, wird jedes Modul bzw. jede Klasse immer von „Grape::API“ abgeleitet. Offenbar sind die Klassen und die zugehörigen Module bei einer Versionierung anders aufzubauen, als bisher angenommen.

Zur Eingrenzung des Problems wurde der Controller „Bootconfig“ umgeschrieben und in die bisherige Version „v1“ eingebunden. Beim anschließenden Versuch die B-API zu starten, trat der ursprüngliche Fehler nicht mehr auf, jedoch erschienen neue Fehlermeldungen bzgl. undefinierter Methoden im Model „Bootconfig“. Dies deutet darauf hin, dass das Überschreiben der Methoden aus einer anderen Basisklasse nicht so funktioniert wie erwartet. Aufgrund der zahlreichen Fehlermeldungen wurde entschieden die Testphase an diesem Punkt abzubrechen.

Als Folge daraus, ist die ursprüngliche Vorgehensweise in Bezug auf die Implementierung noch einmal sorgfältig zu überprüfen. Weiterhin sind die notwendigen Ände-

44 online: GITHUB.com - ruby-grape/grape, 2017 (27.08.2017)

rungen schrittweise zu integrieren und die Module einzeln zu testen. Der bisher gewählte Ansatz, gleich zu Beginn die komplette Struktur aller Klassendateien zu verändern, hat sich als nicht zielführend erwiesen.

Die kompletten Quelltexte befinden sich auf der beigelegten CD.

5.6 Offene Punkte

Die in der praktischen Umsetzung geplanten Änderungen an der B-API sind nicht erfolgreich umgesetzt und getestet. Die korrekte Funktionsweise zwischen B-API und iPXE bei aktivierter Verschlüsselung ist ebenfalls noch nicht überprüft worden.

Bis zum ersten Test in der Produktivumgebung, müssen alle aufgeführten Punkte im Anhang 2 abgearbeitet sein.

6. Fazit und Ausblick

Das angestrebte Ziel der Bachelorthesis, eine architekturübergreifende PXE-Infrastruktur mit webbasierter Schnittstelle zu entwickeln, wurde insofern erreicht, dass ein theoretisches Lösungskonzept erarbeitet und vorgestellt werden konnte. Jedoch ist es in der praktischen Umsetzung nicht gelungen, alle geplanten Vorhaben erfolgreich umzusetzen. Der Kompatibilitätstest von iPXE mit der bestehenden Hardware wurde abgeschlossen und lieferte wichtige Erkenntnisse für die weitere Umsetzung der entwickelten Lösung. Weiterhin konnte das Schema für die Speicherung der Konfigurationsdaten im bestehenden LDAP-Verzeichnis erstellt und getestet werden. Die Implementierung der notwendigen Änderungen an der bestehenden Bootserver-API ist als gescheitert anzusehen, da es aufgrund diverser Fehlermeldungen beim Start der B-API nicht möglich war, die erforderlichen Tests durchzuführen. Die Fehlerursachen ließen sich zwar eingrenzen, konnten aber in der noch zur Verfügung stehenden Zeit nicht zufriedenstellend behoben werden. Offensichtlich muss die bisherige Vorgehensweise bei der Implementierung noch einmal überdacht und neu geplant werden. Zudem wurde der erforderliche Zeitaufwand für die Analyse des bestehenden Quellcodes der B-API unterschätzt, so dass erst sehr spät mit der Implementierung und den Tests begonnen werden konnte.

Wenn es gelingt die Änderungen erfolgreich zu implementieren, kann die erarbeitete Lösungsvariante dennoch mit hoher Wahrscheinlichkeit produktiv im Unternehmen eingesetzt werden.

Ziel ist es, die hierzu notwendigen Arbeiten fortzusetzen und erfolgreich abzuschließen.

Quellenverzeichnis

ANVIN, H. Peter; CONNOR, Marty: x86 Network Booting: Integrating gPXE and PXELINUX, 2008, In: http://kernel.org/doc/ols/2008/ols2008v1-pages-9-18.pdf (24.04.2017)

BSI: M 4.389 Partitionierung und Replikation bei OpenLDAP, 2013, In: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/m/m04/m04389.html (30.07.2017)

COMPAG Corp.; Phoenix Ltd.; INTEL: BIOS Boot Specification, Version 1.01, 1996, In: https://acpica.org/sites/acpica/files/specsbbs101.pdf (05.08.2017)

DELL Inc.: Understanding Preboot Networking on Dell PowerEdge Servers, 2014, In: http://en.community.dell.com/cfs-file/__key/telligent-evolution-components-attachments/13-4491-00-00-20-44-08-37/Understanding_5F00_Preboot_5F00_Networking_5F00_on_5F00_Dell_5F00_PowerEdge_5F00_Servers.pdf (11.07.2017)

EDLICH; u.a.: NoSQL - Einsteig in die Welt nichtrelationaler Datenbanken. 2., aktualisierte und erweiterte Auflage. München, 2011

FIELDING, Roy Thomas: Architectural Styles and the Design of Network-based Software Architectures, 2000, In: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (29.04.2017)

GITHUB.com - ruby-grape/grape: Grape Documentation, 2017, In: https://github.com/ruby-grape/grape/blob/master/README.md (27.08.2017)

HAJDARBEGOVIC, Nermin: ARM Servers: Mobile CPU Architecture For Datacentres?, 2016, In: https://www.toptal.com/back-end/arm-servers-armv8-for-datacentres (13.07.2017)

HETZNER Online GmbH: Installimage, 2017, In: https://wiki.hetzner.de/index.php/Installimage (21.07.2017)

HETZNER Online GmbH: Hetzner Rescue-System, 2017, In: https://wiki.hetzner.de/index.php/Hetzner_Rescue-System (20.07.2017)

IANA: Dynamic Host Configuration Protocol for IPv6 (DHCPv6), 2017, In: https://www.iana.org/assignments/dhcpv6-parameters/dhcpv6-parameters.xhtml#processor-architecture (20.7.2017)

INTEL Corporation: Preboot Execution Environment (PXE) Specification, 1999, In: http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf (11.07.2017)

INTEL Corporation: Driver Writer's Guide for UEFI 2.3.1, 2012, In: https://github.com/tianocore-docs/Docs/raw/master/Driver_Developer/UEFI_Driver_Writer_Guide_V1.0.1_120308.pdf (11.07.2017)

IPXE: iPXE Build targets, 2017, In: http://ipxe.org/appnote/buildtargets (02.08.2017)

IPXE: PXE chainloading, 2017, In: http://ipxe.org/howto/dhcpd#pxe_chainloading

(20.07.2017)

IPXE: Frequently asked questions, 2017, In: http://ipxe.org/faq (19.07.2017)

IPXE Forum: iPXE exit fails, 2013, In: http://forum.ipxe.org/showthread.php?tid=6955 (12.08.2017)

ISC Inc.: Kea DHCP server, 2017, In: https://www.isc.org/kea/ (06.08.2017)

KERNEL.org: The EFI Boot Stub, 2017, In: https://www.kernel.org/doc/Documentation/efi-stub.txt (20.07.2017)

KITTENBERGER, Axel: Lsyncd (Live Syncing Daemon) synchronizes local directories with remote targets, 2017, In: https://github.com/axkibe/lsyncd (31.07.2017)

LIEBEL, Oliver; UNGAR John Martin: OpenLDAP 2.4 - Das Praxisbuch. Bonn, 2009

NGINX.org: Using nginx as HTTP load balancer, 2017, In: http://nginx.org/en/docs/http/load_balancing.html (31.07.2017)

OPENLDAP Foundation: OpenLDAP Software 2.4 Administrator's Guide, 2012, In: https://www.openldap.org/doc/admin24/OpenLDAP-Admin-Guide.pdf (01.08.2017)

RED HAT, Incorporation: Can Ceph Support Multiple Data Centers, 2017, In: http://tracker.ceph.com/projects/ceph/wiki/Can_Ceph_Support_Multiple_Data_Centers (06.08.2017)

RFC 2252: Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions, 1997, In: https://www.ietf.org/rfc/rfc2252.txt (13.08.2017)

RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1, 1999, In: https://www.ietf.org/rfc/rfc2616.txt (15.08.2017)

RFC 4517: Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules, 2006, In: https://www.ietf.org/rfc/rfc4517.txt (13.08.2017)

RFC 4578: Dynamic Host Configuration Protocol (DHCP) Options for the Intel Preboot eXecution Environment (PXE), 2006, In: https://tools.ietf.org/html/rfc4578 (19.07.2017)

RFC 5970: DHCPv6 Options for Network Boot, 2010, In: https://tools.ietf.org/html/rfc5970 (19.07.2017)

RIEMERSMA, Thiadmer: Extending TFTP, 2012, In: https://www.compuphase.com/tftp.htm (18.07.2017)

RUBY, Sam; THOMAS, Dave; HANSSON David: Agile Web Development with Rails 5. Raleigh, 2016

STORAGE SERVERS: Facebook data centers switch from ISC DHCP to KEA to speedup up their servers!, 2015, In: https://storageservers.wordpress.com/2015/07/24/facebook-data-centers-switch-from-isc-dhcp-to-kea-to-speedup-up-their-servers/ (31.07.2017)

TILKOV, Stefan: REST – Der bessere Web Service?, 2009, In: https://jaxenter.de/rest-der-bessere-web-service-8988 (29.04.2017)

TILLEY, Aaron: Microsoft Partners With Qualcomm To Open Floodgates For ARM-Based DataCenter, 2017, In: https://www.forbes.com/sites/aarontilley/2017/03/08/microsoft-partners-with-qualcomm-to-open-floodgates-for-arm-based-data-center/#259dd4e6397c (13.07.2017)

UEFI Forum: Unified Extensible Firmware Interface Specification. Version 2.6, 2016, In: http://www.uefi.org/sites/default/files/resources/UEFI%20Spec%202_6.pdf (11.07.2017)

WIKIPEDIA: Model View Controller, 2017, In: https://de.wikipedia.org/wiki/Model_View_Controller (17.08.2017)

WIKIPEDIA: Comparison of boot loaders, 2017, In: https://en.wikipedia.org/wiki/Comparison_of_boot_loaders (19.07.2017)

ZIMMER, Vincent; ROTHMAN, Michael; MARISETTY Suresh: Beyond BIOS - Developing with the Unified Extensible Firmware Interface. Second Edition. Santa Clara, 2010

Anhangsverzeichnis

Anhang 1 – Dokumentation der Bootserver-API „v1“

BootserverApi

TFTP and DHCP-LDAP REST frontend.

ldap : Operations about ldaps — Show/Hide | List Operations | Expand Operations

DELETE	/api/v1/ldap	Remove dhcp entry
GET	/api/v1/ldap	Fetch dhcp entry
POST	/api/v1/ldap	Create new dhcp entry

tftp : Operations about tftps — Show/Hide | List Operations | Expand Operations

GET	/api/v1/tftp/request_cache	receive cached info if available
DELETE	/api/v1/tftp/commands	Remove tftp command
GET	/api/v1/tftp/commands	List all command descriptions
GET	/api/v1/tftp/commands/active	Get all active tftp commands
GET	/api/v1/tftp/commands/{command}	get command description
POST	/api/v1/tftp/commands/{command}/config	write tftp config
POST	/api/v1/tftp/commands/{command}/info	generate tftp config and return params (dry-run)

wol : Operations about wols — Show/Hide | List Operations | Expand Operations

POST	/api/v1/wol	Create new dhcp entry

logs : Operations about logs — Show/Hide | List Operations | Expand Operations

GET	/api/v1/logs	filter logs

validate : Operations about validates — Show/Hide | List Operations | Expand Operations

GET	/api/v1/validate	validate ip mac assignment

[BASE URL: , API VERSION: V1]

Anhang 2 – Arbeitsschritte zur Integration in die Produktivumgebung

1. Kompatibilitätstest mit Bestandshardware durchführen und Ergebnisse dokumentieren
2. Implementieren und Testen der neuen Funktionalitäten für die B-API
3. Erstellung der neuen Bootmenüs für die Produktivumgebung
4. Prüfung und Anpassung der benötigten Anwendungen für den Betrieb unter UEFI
    - automatische Windows-Installation für UEFI
    - „installimage"
        - UEFI-Branch in Master-Branch einbinden
        - Betriebssystem-Images anpassen
    - HWCheck
        - Test unter UEFI
        - Portierung für ARMv8-Architektur
5. Installieren und Konfigurieren der notwendigen Server (Webserver, DHCP, LDAP)
6. Einbinden des neuen Schemas auf den LDAP-Servern in FSN und NBG
7. Anpassung RZ-Admin
8. Test in der Produktivumgebung in FSN durchführen
    - zunächst mit Version „v2“ unter UEFI, um den Betrieb nicht zu beeinträchtigen
    - Parallelbetrieb PXELINUX und iPXE mit einigen Testservern über DHCP-Filterung
    - Tests nach Möglichkeit fertigstellen, solange Finnland noch nicht im Netz
9. Abklären der notwendigen Änderungen am „Hetzner-Robot“ (UEFI, ARM64), Anpassung der API-Routen
10. Installation und Konfiguration der notwendigen Server in NBG und Finnland

Anhang 3 – Konfiguration des DHCP-Servers

```
# dhcpd.conf


INTERFACES="eth1";

allow booting;
allow bootp;

option arch code 93 = unsigned integer 16;

option routers 192.168.0.1;
option domain-name-servers 213.133.100.100;

next-server 192.168.0.1;

log-facility local7;
ddns-update-style none;
authorative;
default-lease-time 600;
max-lease-time 172800;

subnet 192.168.0.0 netmask 255.255.255.0 {
            range 192.168.0.10 192.168.0.20;
            if exists user-class and option user-class = "iPXE" {
                  filename "http://192.168.0.1/ipxe/scripts/start.ipxe";
            } elsif option arch = 00:0b {
                  filename "boot/ipxe/efi/snponly_arm64.efi";
            } elsif option arch = 00:07 {
                  filename "boot/ipxe/efi/snponly_x64.efi";
            } else {
                  filename "boot/ipxe/legacy/undionly.kkpxe";
            }
}
```

Anhang 4 – iPXE-Bootmenü

```
#!ipxe

set pass Passw0rd

## Architektur ermitteln ##
:start
set arch ${buildarch}
iseq ${arch} i386 && set arch x86_64 ||
iseq ${arch} arm64 && goto menu_arm64 ||
goto menu_x86_64

## ARM64-Menü ##
:menu_arm64
menu PXE Test System (${arch}, ${platform})
item rescue_arm64_1 Rescue-System (jessie/beta, Kernel 4.9.36)
item restricted Advanced options
item shell Start iPXE shell
item localboot Exit iPXE and boot from local disk

choose --default localboot --timeout 5000 target && goto ${target}

## x86_64-Menü ##
:menu_x86_64
menu PXE Test System (${arch}, ${platform})
item rescue_x86 Rescue-System (jessie/current)
item restricted Advanced options
item shell Start iPXE shell
item localboot Exit iPXE and boot from local disk

choose --default localboot --timeout 10000 target && goto ${target}

## ARM64-Ziele ##

:rescue_arm64_1
kernel http://192.168.0.1/rescue-arm64/4.9.36/vmlinuz
initrd=initramfs.cpio.xz nfsdir=192.168.0.1:/nfs RFILE=rescue_arm64-
current.ext2 BOOTIF=${net0/mac} vga=792 HASH={secret} quiet
initrd http://192.168.0.1/rescue-arm64/4.9.36/initramfs.cpio.xz
boot || goto failed

## x86-Ziele ##
:rescue_x86
kernel http://192.168.0.1/rescue-x64/4.10.3-64/vmlinuz
initrd=initramfs.cpio.xz nfsdir=192.168.0.1:/nfs RFILE=rescue64-dev.ext2
BOOTIF=${net0/mac} vga=792 HASH={secret} quiet
initrd http://192.168.0.1/rescue-x64/4.10.3-64/initramfs.cpio.xz
boot || goto failed

## gemeinsame Ziele ##
:localboot
exit

:shell
```

```
echo Type 'exit' to get the back to the menu
shell
set menu-timeout 0
set submenu-timeout 0
goto menu_${arch}

:failed
echo Booting failed, dropping to shell
goto shell

## Login für passwortgeschütztes Menü ##
:restricted
login
iseq ${password} ${pass} && goto menu_restricted || prompt Wrong username
or password, Press key to continue ...
goto menu_${arch}

## geschütztes Menü ##
:menu_restricted
menu Advanced options
item test1 Test
item back Back to main menu

choose --default test1 target && goto ${target}

:test1
prompt Only a test, Press key to continue ...
goto menu_restricted

:back
goto menu_${arch}
```

Anhang 5 – Ergebnisse Kompatibilitätstest

Kompatibilitätstest von iPXE mit bestehender Serverhardware

Servermodell	Start iPXE mit Standardmenü (ohne B-API)		Wechsel auf nächsten Booteintrag nach dem Beenden von iPXE		BS-Installation mit "installimage"		Bootreihenfolge nach Neustart verändert	
	BIOS[1]	UEFI[2]	BIOS	UEFI	BIOS	UEFI	BIOS	UEFI
DX 151/291	X	X	X	X	X	X	nein	nein
PX 91/121	X	X	X	X	X	X	nein	nein
PX 61	X	X	X	X	X	X	nein	nein
EX 41/51	X	X	X	X	X	X	nein	nein
AX 50/60	X	X	X	X	X	X	nein	nein
DX 150/290	X	X	X	X	X	X	nein	nein
PX 90/120	X	X	X	X	X	X	nein	nein
PX 80	X	–[3]	X	–	X	–	nein	–
PX 60/70	X	X	X	X	X	X	nein	nein
EX 40	X	O[4]	X[5]	–	X	–	nein	–
EX 10	X	–[3]	O	–	X	–	nein	–
EX 6/6S/8	X	–[3]	X	–	X	–	nein	–
EX 4/4S	X	O[4]	X	–	X	–	nein	nein
EQ 4/6/8/10	X	–	X	–	X	–	nein	–
DS 3000/5000/8000	X	–	X	–	X	–	nein	–
Gigabyte MT30-GS0[6]	–	X	–	X	–	X	–	nein
Gigabyte MP30-AR1[6]	–	X	–	X	–	O	–	–

–	nicht unterstützt
X	ok
O	nicht ok
1	undionly.kkpxe
2	snponly.efi (nur IPv4)
3	UEFI-Firmware ohne Netzwerkstack
4	alle verwendeten Mainboards
5	nicht mit Asus B85M-E
6	ARM64-Testsystem

Anmerkungen:

Bei einigen Servermodellen sind verschiedene Mainboardtypen verbaut.

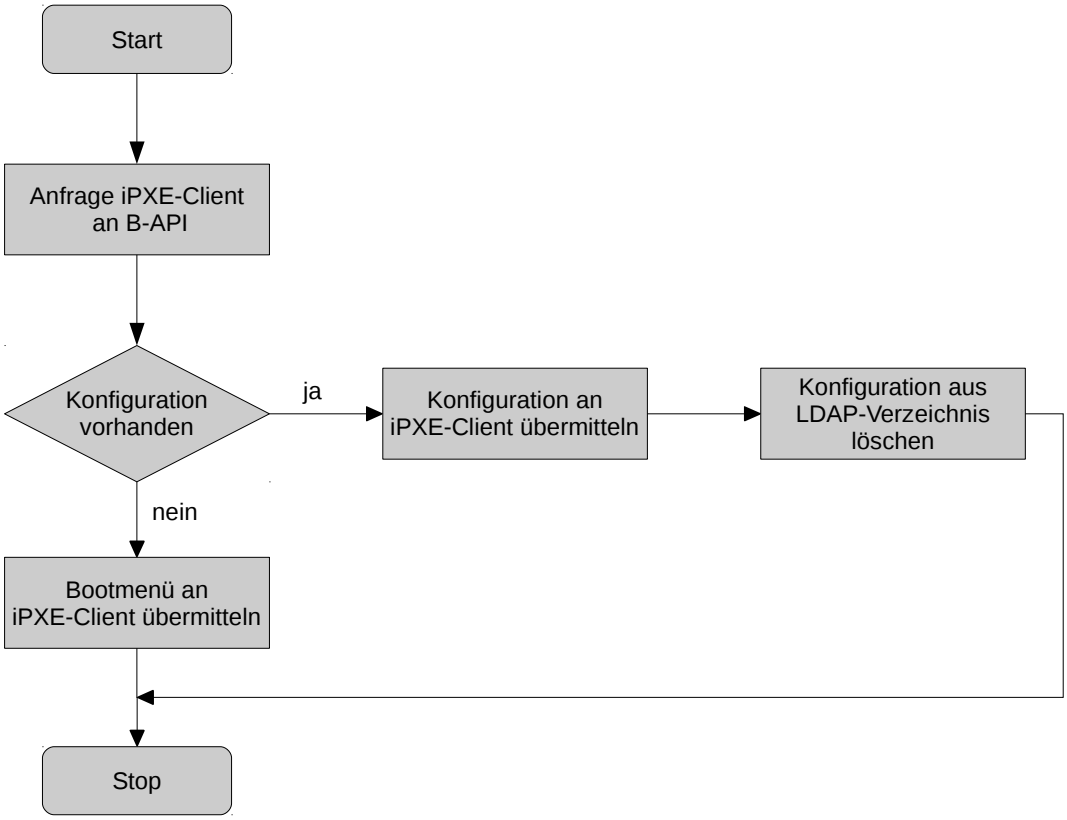
Anhang 6 – iPXE-Startskript

```
#!ipxe

# get mainboard type
set mainboard-type ${smbios/2.5.0}
# get network device driver (we start with undionly.kkpxe)
set nic-driver ${net0/chip}
# set URL for Bootserver-API
set api-url http://192.168.0.1/api/v2/bootconfig/?arch=${buildarch}&mac=$
{net0/mac}&ip=${net0/ip}

# if mainboard incompatible with current iPXE version,
# chainload ipxe.pxe, else call B-API
iseq ${mainboard-type} DX79TO && iseq ${nic-driver} undionly && chain
http://192.168.0.1/ipxe/bin/ipxe.pxe ||
iseq ${mainboard-type} B85M-E && iseq ${nic-driver} undionly && chain
http://192.168.0.1/ipxe/bin/ipxe.pxe ||

# if vServer detected load PXELINUX
# for testing in production environment
iseq ${manufacturer} QEMU && chain tftp://boot/pxelinux.0 ||

chain -ar ${api-url}
```

Anhang 7 – Verzeichnisstruktur der Bootserver-API „v1“

```
BootserverAPI
.
├── bin
│   ├── console
│   └── setup
├── bootserver_api_doku.png
├── config.ru
├── dhcp.ldif
├── docker-compose.yml
├── Dockerfile
├── exe
│   └── bootserver_api_wol
├── extconf.rb
├── Gemfile
├── hetzner-bootserver_api.env
├── hetzner-bootserver_api.gemspec
├── hetzner-bootserver_api.service
├── lib
│   └── hetzner
│       ├── bootserver_api
│       │   ├── controller
│       │   │   ├── ldap.rb
│       │   │   ├── tftp.rb
│       │   │   └── wol.rb
│       │   ├── model
│       │   │   ├── ldap.rb
│       │   │   ├── tftp.rb
│       │   │   ├── validator.rb
│       │   │   └── wol.rb
│       │   └── version.rb
│       └── bootserver_api.rb
├── puma.conf
├── Rakefile
├── README.md
├── spec
│   ├── fixtures
│   │   └── testfile.tftp
│   ├── hetzner
│   │   └── bootserver_api_spec.rb
│   ├── spec_helper.rb
│   └── templates
│       └── template_w2012_r2.xml
└── tftp_commands.rb
```

Anhang 8 – Ablauf einer Anfrage eines PXE-Clients an die B-API



Anhang 9 – Fehlermeldung beim Start der B-API „v2“

```
andre@pxe-test:~/hetzner-bootserver_api$ bundle exec rackup
/home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api/controller/v1/basecontroller.rb:9
:in `<module:V1>': uninitialized constant
Hetzner::BootserverApi::BaseController (NameError)
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api/controller/v1/basecontroller.rb:7
:in `<module:Controller>'
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api/controller/v1/basecontroller.rb:6
:in `<module:BootserverApi>'
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api/controller/v1/basecontroller.rb:5
:in `<module:Hetzner>'
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api/controller/v1/basecontroller.rb:4
:in `<top (required)>'
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api.rb:14:in `require'
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api.rb:14:in `<module:BootserverApi>'
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api.rb:6:in `<module:Hetzner>'
        from /home/andre/hetzner-
bootserver_api/lib/hetzner/bootserver_api.rb:5:in `<top (required)>'
        from /home/andre/hetzner-bootserver_api/config.ru:2:in `require'
        from /home/andre/hetzner-bootserver_api/config.ru:2:in `block in
<main>'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/builder.rb:55:in
`instance_eval'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/builder.rb:55:in
`initialize'
        from /home/andre/hetzner-bootserver_api/config.ru:in `new'
        from /home/andre/hetzner-bootserver_api/config.ru:in `<main>'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/builder.rb:49:in
`eval'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/builder.rb:49:in
`new_from_string'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/builder.rb:40:in
`parse_file'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/server.rb:319:in
`build_app_and_options_from_config'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/server.rb:219:in
`app'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/server.rb:354:in
`wrapped_app'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/server.rb:283:in
`start'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/lib/rack/server.rb:148:in
`start'
        from /var/lib/gems/2.3.0/gems/rack-2.0.3/bin/rackup:4:in `<top
(required)>'
```

```
from /usr/local/bin/rackup:23:in `load'
from /usr/local/bin/rackup:23:in `<main>'
```

Anhang 10 – URLs der gefundenen Beispiele

http://dreamingecho.es/create-a-super-fancy-api-with-grape/
http://funonrails.com/2014/03/building-restful-api-using-grape-in-rails/
https://www.monterail.com/blog/2014/introduction-to-building-apis-with-grape
http://www.thegreatcodeadventure.com/making-a-rails-api-with-grap/

Zugriff auf alle URLs am 28.08.2017

Ehrenwörtliche Erklärung

„Ich erkläre hiermit ehrenwörtlich“,

1. dass ich meine Bachelorthesis mit dem Thema

   „Entwicklung einer architekturübergreifenden PXE-Infrastruktur mit webbasierter Schnittstelle“

   ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Bachelorthesis bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Muldenhammer, den 04.09.2017_____ _____

Ort, Datum Unterschrift

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
GLAUCHAU
UNIVERSITY OF COOPERATIVE EDUCATION

Erklärung zur Prüfung wissenschaftlicher Arbeiten

Die Bewertung wissenschaftlicher Arbeiten erfordert die Prüfung auf Plagiate. Die hierzu von der Staatlichen Studienakademie Glauchau eingesetzte Prüfungskommission nutzt sowohl eigene Software als auch diesbezügliche Leistungen von Drittanbietern. Dies erfolgt gemäß § 7 des Gesetzes zum Schutz der informationellen Selbstbestimmung im Freistaat Sachsen (Sächsisches Datenschutzgesetz – SächsDSG) vom 25. August 2003 (Rechtsbereinigt mit Stand vom 31. Juli 2011) im Sinne einer Datenverarbeitung im Auftrag.
Der Studierende bevollmächtigt die Mitglieder der Prüfungskommission hiermit zur Inanspruchnahme o.g. Dienste. In begründeten Ausnahmefällen kann der Datenschutzbeauftragte der Staatlichen Studienakademie Glauchau sowohl vom Verfasser der wissenschaftlichen Arbeit als auch von der Prüfungskommission in den Entscheidungsprozess einbezogen werden.

Name:	Röder
Vorname:	André
Matrikelnummer:	4001610
Studiengang:	Technische Informatik
Titel der Arbeit:	Entwicklung einer architekturübergreifenden PXE-Infrastruktur mit webbasierter Schnittstelle
Datum:	04.09.2017
Unterschrift:	