Evaluierung eines Messpunkte-Clusters für Netzwerktests auf dem Campus der TU Clausthal

Christian Rebischke
Technische Universität Clausthal
Rechenzentrum
Matrikelnummer: 432108
Studiengang: Informatik (Bachelor)
E-Mail: Christian.Rebischke@tu-clausthal.de

10. Februar 2019

Danksagung

Ich bedanke mich bei dem Rechenzentrum der TU Clausthal, insbesondere bei Dipl.-Math. Christian Strauf. Das Thema dieser Bachelorarbeit beruht auf seiner Idee und war mein Ansporn, mich mit diesem Thema näher auseinanderzusetzen. Desweiteren danke ich Herrn Prof. Dr.-Ing. Dr. rer. nat. habil. Harald Richter für die Unterstützung aus akademischer Seite. Besonderen Dank bekommt von mir auch die Opensource-Gemeinschaft. Ohne die harte Arbeit der Opensource-Gemeinschaft würden die grundlegenden Werkzeuge, die ich zur Vollendung dieser Bachelorarbeit benutzt habe, nicht existieren. Besonders hilfreich waren das Softwarepaket LaTeX und die Zeichensoftware *Draw IO* (`https://www.draw.io/`).

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertig habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach anderen gedruckten oder im Internet verfügbaren Werken entnommen sind, habe ich durch genaue Quellenangaben kenntlich gemacht. Außerdem wurde diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsstelle im Sinne von §11 Absatz 5 lit. b) der allgemeinen Prüfungsordnung vorgelegt.

Clausthal-Zellerfeld, der 10. Februar 2019

Christian Rebischke

Sperrvermerk

Hiermit erkläre ich mich damit einverstanden, dass meine Bachelorarbeit in der Instituts- und/oder der Universitätsbibliothek ausgelegt und zur Einsichtnahme aufbewahrt werden darf.

Clausthal-Zellerfeld, der 10. Februar 2019

Christian Rebischke

Inhaltsverzeichnis

1 Vorwort **6**

2 Problemstellung **7**

3 Technische Grundlagen **11**
3.1 Netzwerkgrundlagen 11
3.2 Domain Name System (DNS) 13
3.3 Hypertext Transport Protocol (HTTP) 15
3.4 Hypertext Transport Protocol Secure (HTTPS) 18
3.5 Common Internet File System (CIFS) 19

4 Herleitung eines Lösungsansatzes **21**
4.1 Anforderungsanalyse 21
4.1.1 Funktionale Anforderungen 22
4.1.2 Nicht-funktionale Anforderungen 23
4.2 Systemarchitektur 24
4.3 Datenbanken 25
4.3.1 Relationale Datenbanken 26
4.3.2 NoSQL-Datenbanken 27
Erwartete Daten und Wahl der geeigneten Datenbank 28
4.4 Prothemeus 29
4.4.1 Datenspeicherung in Prometheus 29
4.4.2 Komponenten 31
4.4.3 PromQL 34
4.5 Grafana 35
4.6 Konfigurationsmanagement 36
4.7 Ansible 37
4.7.1 Funktionsweise 37
4.7.2 Organisationsstruktur 38
4.7.3 Dateistruktur 39
4.7.4 Sicherheit 41
4.8 Wahl der Hardware 41

5 Projektumsetzung **44**
5.1 Erfüllung der Anforderungen 44
5.2 Test- und Entwicklungsumgebung 46
5.3 Versionsverwaltung 48
5.4 Prototyp 50
5.4.1 Hinzufügen von CIFS Unterstützung 50
5.4.2 Bau eines Grafana Dashboards 56
5.5 Einsatz im Rechenzentrum 59
5.5.1 Vorbereitungen 59
5.5.2 Deployment 61
5.5.3 Abschließende Anforderungsanalyse 65

6 Fazit und Ausblick **68**

Kapitel 1

Vorwort

Es gibt nun das Internet seit circa 45 Jahren und das World Wide Web (WWW) seit 24 Jahren und keine Technologie ist über so kurze Zeit so alltäglich geworden. Das Internet hat es geschafft, Einzug zu erhalten in Arbeit, Privatleben und auch Forschung und Lehre. Nahezu in allen wissenschaftlichen Diszplinen spielt das Internet und die damit verknüpfte Informationstechnologie eine Rolle. Sei es die Industrie 4.0 mit ihren cyber-physischen Systemen, dem schnellen Abgleich von DNA-Informationen über das Netz in der angewandten Biologie, dem Sammeln von Krankheitsdaten in der Medizin oder das Verarbeiten von Datenmengen gigantischen Ausmaßes im Finanzsektor. All diese Beispiele sind nur möglich durch immer größere Technologiesprünge in der Informatik und dem immer größeren Ausbau des Internets. Da ist es nicht verwunderlich, dass der UN-Menschenrechtsrat das Internet zu einem Menschenrecht[17] erklärt hat und umso weniger verwunderlich ist es, dass die Vernetzung von Computersystemen auch auf dem Campus der TU Clausthal eine Rolle spielt, nicht nur für Forschung und Lehre, sondern auch für den täglichen Betrieb. Eine Schlüsselposition nimmt dabei das Rechenzentrum der TU Clausthal ein. Das Rechenzentrum bildet die Basis für die Vernetzung der einzelnen Fakultäten untereinander, der Vernetzung zwischen Fakultäten und Firmen aus der freien Wirtschaft, sowie auch die Vernetzung zwischen der TU Clausthal und anderen Universitäten weltweit. Dementsprechend wichtig ist ein stabiles Netz für den täglichen Betrieb. In dieser Bachelorarbeit widme ich mich deshalb der technischen Umsetzung eines verteilten Monitoring-Systems zur Überwachung der Netzwerkqualität zwischen einzelnen Endpunkten und Kernsystemen, die für einen problemlosen Netzbetrieb nötig sind. Das Rechenzentrum der TU Clausthal dient bei dieser Bachelorarbeit als Auftraggeber.

Kapitel 2

Problemstellung

Das Netz der TU Clausthal erstreckt sich über mehrere Standorte. Teilweise liegen diese Standorte nicht in Clausthal selbst, wie beispielsweise das Energie-Forschungszentrum Niedersachsen (EFZN) in Goslar. Dementsprechend schwierig gestaltet sich die Wartung und der Betrieb des Netzes. So kann auf Netzeinbrüche etwa nur reaktiv nach Meldung des Problems reagiert werden. Es existiert zwar ein Monitoring-System, welches die Verfügbarkeit von einzelnen Diensten überprüft, jedoch erfolgt diese Messung von einem Punkt aus und gibt binäre Statuswerte zurück (Dienst läuft oder Dienst läuft nicht). Dementsprechend fehlen Informationen um die Verfügbarkeit von Diensten und deren vollständige Funktion von mehreren Messpunkten aus zu garantieren. Beispielsweise ist es möglich, dass ein Dienst zwar vom zentralen Monitoring-Server aus erreichbar ist, aber aus einem einzelnen Institut der Zugriff auf den Dienst nur eingeschränkt oder sogar gar nicht möglich ist. Das Rechenzentrum der TU Clausthal bietet mehrere Kerndienste an. Dazu gehören:

- Domain Name System (DNS)
- Diverse Webdienste basierend auf:
  - Hypertext Transport Protocol (HTTP)
  - Hypertext Transport Protocol Secure (HTTPS)
- Common Internet File System (CIFS)
- Voice Over Internet Protocol (VoIP)
- Simple Mail Transfer Protocol (SMTP)

Abbildung 2.1 zeigt die aktuelle Netzstruktur der TU Clausthal. Zu sehen sind die drei *Core-Router*, welche auf die Gebiete Rechenzentrum, Feldgraben und Tannenhöhe verteilt sind. *Core-Router* stellen das Rückgrat, den Backbone, des Local Area Network (LAN) der TU Clausthal dar und leiten den Datenverkehr zwischen den einzelnen am Netz angeschlossenen Geräten. Sie trennen

außerdem das Netz in logische Abschnitte. Zu sehen ist ebenfalls das aktuell existierende Monitoring-System, welches einzelne Kerndienste überwacht. Diese Kerndienste und das aktuelle Monitoring-System sind im Rechenzentrum beheimatet. Dadurch entsteht eine physikalische und auch logische Nähe der Systeme. Durch diese Nähe verlässt der Datenverkehr, welcher die Kerndienste überprüft, niemals die LANs des Rechenzentrum. Dies führt dazu, dass einzelne Kerndienste zwar als in Betrieb und fehlerfrei angezeigt werden, aber durchaus die Möglichkeit besteht, dass einzelne Dienste nicht von jedem Rechner der TU Clausthal der TU Clausthal Zustand ermittelt wird, ist unklar, wie groß die Latenz zwischen den Diensten und den jeweiligen Endkunden ist. Diese Latenz ist allerdings entscheidend und hat Einfluss auf die Nutzererfahrung der Endkunden. Besonders Dienste wie DNS sind auf Verbindungen mit einer geringen Latenz (der Zeit zwischen dem Absenden des ersten Bytes eines Pakets und dessen Empfang) angewiesen. Eine zu hohe Latenz zwischen einem Client und dem Dienst führt unweigerlich zu für den Nutzer spürbaren Konsequenzen (zum Beispiel zu verzögerten Seitenaufrufe beim Web-Browsing). Noch mehr ins Gewicht fallen Latenzen bei VoIP, dort sind Latenzen oder ein Jitter (die Varianz der Laufzeit der Datenpakete[6]) leicht auszumachen. Außerdem ist anzumerken, dass bei VoIP eine Mindestbandbreite vorhanden ist, da sonst der Codec (Deutsch: Dekodierer) schlecht arbeitet. Was fehlt, ist ein System aus verteilten Messpunkten in den LANs der TU Clausthal, das es ermöglicht, die Erreichbarkeit einzelner Dienste periodisch und über einen längeren Zeitraum zu beobachten (siehe Abbildung 2.2). Dies hätte mehrere Vorteile: Zum einen ließe sich so der Zustand des Campus-Netzwerks besser erfassen, da Tests nicht nur von einem zentralen Monitoring-System aus gestartet werden. Zum anderen können die gewonnenen Daten weiter verwertet, grafisch aufbereitet und zum Beispiel für die Erstellung von Langzeitstatistiken über die Gesundheit des Netzwerks genutzt werden. Der Zustand des Netzwerkes wird somit als Funktion der Zeit angesehen. Weiterhin könnten im Fall eines Ausfalls die zuständigen Netzadministratoren benachrichtigt werden, im Idealfall durch gewohnte Kommunikationswege wie E-mail, aber auch durch neue Kommunikationswege wie in etwa das Absetzen von Tweets oder Chatnachrichten (beispielsweise an den Chat-Server der TU Clausthal: `https://chat.rz.tu-clausthal.de`). Außerdem wäre es durch einen dezentralen Aufbau einfacher, das System beliebig zu skalieren und auf Wachstum und Schrumpfen des Netzes zu reagieren. Mit dem Wandel von einem zentralen zu einem dezentralen Monitoring-System entsteht allerdings auch mehr Arbeitsaufwand, denn auch diese Systeme müssen gewartet werden. Dies umfasst das Verwalten von Konfigurationsdateien, das Aktualisieren und Installieren von Software und die Installation des Grundsystems. Als zentrale Forschungsfrage ergibt sich somit: *Wie lassen sich Rechnernetze dezentral und skalierbar überwachen?* Insgesamt lassen sich daraus folgende Herausforderungen an diese Arbeit ableiten:

- Es müssen Server im Campusnetz gefunden werden, welche sich als verteilter Messpunkt eignen oder diese durch den Einsatz von neuer Hardware im Campusnetz platziert werden.

- Es muss eine Software-Plattform gebaut oder eine vorhandene Software-Plattform erweitert werden, sodass sie den Anforderungen bezüglich Monitoring aus der Problemstellung gerecht wird.
- Es muss ein Modell für die gespeicherten Daten gefunden werden.
- Es muss ein Weg zur Datenvisualisierung gefunden werden.
- Das System muss horizontal skalierbar sein. Das heißt, es muss um Messpunkte erweiterbar sein.
- Das System muss bei wachsendem Komplexitätsgrad und Anzahl von Messpunkten steuerbar und kontrollierbar bleiben.
- Das System muss über Transport Control Protocol (TCP)/Internet Protocol (IP) oder User Datagram Protocol (UDP)/IP basierte Rechnernetze kommunizieren. Dies schließt die Kommunikation über logische Teilnetze wie Virtual Local Area Network (VLAN)s oder Virtual Extended Local Area Network (VXLAN)s mit ein.

Abbildung 2.1: Veranschaulichung der Problemstellung anhand der aktuellen Netzstruktur der TU Clausthal ohne die Anbindungen Goslar und Celle über das öffentliche Internet.

Abbildung 2.2: Veranschaulichung der Problemlösung anhand der aktuellen Netzstruktur der TU Clausthal ohne die Anbindungen Goslar und Celle über das öffentliche Internet.

Kapitel 3

Technische Grundlagen

In diesem Kapitel werden die für die Lösung des Problems nötigen technischen Grundlagen erläutert. Außerdem werden die Fehlerfälle für die einzelnen Protokolle definiert. Unter technische Grundlagen fallen einige wichtige Netzwerkprotokolle, sowie eine allgemeine Einführung in Computernetzwerke.

3.1 Netzwerkgrundlagen

Um auf Basis der vorangegangenen Problemstellung eine Lösung zu erarbeiten, ist es notwendig, einen groben Überblick über die Grundlagen von Computernetzwerken zu bekommen. Als Basis dafür dient das Open Systems Interconnection Model (OSI-Modell). Das OSI-Modell ist de facto das bis heute gängige Referenzmodell, wenn es darum geht, mehrere Systeme miteinander zu vernetzen. Ein solches System wird *offenes System* genannt, wenn es den im OSI-Modell spezifizierten Standards entspricht[56, Siehe Abschnitt 4.1.2]. Diese *offenen Systemen* sind mit einem physischen Medium verbunden und bilden so ein Computernetzwerk. Abbildung 3.1 zeigt eine solche Verbindung mehrerer *offener Systeme*.

Abbildung 3.1: Stark vereinfachte Darstellung der Assoziationen verschiedenener offener Systeme mit diversen Anwendungen über ein physisches Medium.

Innerhalb eines dieser *offener Systeme* sind sieben Netzwerkschichten oder auch *OSI-Schichten* definiert[56, Siehe Abschnitt 6.1.2]. Zur Kommunikation zwischen zwei *offenen Systemen* werden mindestens die beiden untersten Schichten durchlaufen, die Bitübertragungsschicht für den Transport der Bits und die Sicherungsschicht für den Zugriff auf den Bittransport. Abbildung 3.2 listet alle sieben Schichten, deren Protokolle, Einheiten und Einordnung auf. Die Funktionsweise des OSI-Modells lässt sich am besten durch ein kurzes Beispiel erklären:

Es wird angenommen, dass von einem Laptop eine Website aufgerufen wird. Nachfolgend wird nur der Netzverkehr zwischen dem Laptop und dem Webserver betrachtet. Wenn der Client eine HTTP-basierte Website aufrufen möchte, sendet dieser eine HTTP-Anfrage in Form von eines HTTP-Requests, der den Beginn des HTTP-Protokolls darstellt (Anwendungsschicht). Diese HTTP-Daten werden wiederum in über TCP in 64 Kilobyte große TCP-Pakete verpackt (Transportschicht), welches wiederum über 64 Kilobyte große Pakete des IP-Protokolls dem Ziel zugestellt werden (Vermittlungsschicht). Diese IP-Pakete enthalten die Adresse des Empfängers und werden in viele Ethernet-Datenrahmen mit einer Länge von 1500 Bytes eingebettet (Sicherungsschicht), welche in Form von kodierten Bits über ein Netzwerkkabel übertragen werden (Bitübertragungsschicht). Dieses Matrjoschka-ähnliche Gebilde wird über das physische Medium bitseriell versendet und der Empfänger packt es, angefangen bei der Bitübertragungsschicht, aufsteigend wieder aus.

Anmerkung: Es handelt sich hierbei um eine starkvereinfachte Darstellung. In der Realität kommen diverse andere Faktoren dazu, wie in etwa:

- Das Auflösen eines Hostnamen mit DNS
- Das Routing über IP

- Der Aufbau einer Session mit TCP
- Der ungesicherte Transport mit UDP

<table>
<tr><th>OSI-Schicht</th><th>Einordnung</th><th>Protokollbeispiele</th><th>Einheiten</th></tr>
<tr><td>Anwendungen</td><td rowspan="5">Ende-zu-Ende-Verbindungen über mehrere Hops</td><td rowspan="3">HTTP
HTTPS
SMTP
XMPP
DNS
LDAP
SSH
NRPE
NSCA</td><td rowspan="3">Daten</td></tr>
<tr><td>Darstellung</td></tr>
<tr><td>Sitzung</td></tr>
<tr><td>Transport</td><td>TCP
UDP</td><td>TCP (Segmente)
UDP (Datagramme)
(Pakete)</td></tr>
<tr><td>Vermittlung</td><td>ICMP
IGMP
IP</td><td>Pakete (Fragmente)</td></tr>
<tr><td>Sicherung</td><td rowspan="2">Punkt-zu-Punkt-Verbindungen (gilt nicht für WLAN)</td><td rowspan="2">Ethernet
Token Ring
FDDI
MAC
ARCNET</td><td>Frames (Rahmen)</td></tr>
<tr><td>Bitübertragung</td><td>Bits</td></tr>
</table>

Abbildung 3.2: Das OSI-Schichtenmodell mit Protokollbeispielen und verwendeten Einheiten

3.2 Domain Name System (DNS)

Aus Erfahrungen mit dem Internetvorgänger Advanced Research Projects Agency Network (ARPANET) wurde abgeleitet, dass ein manuelles Eingeben von IP-adressen mit steigender Anzahl von Knoten im Netzwerk immer unübersichtlicher wurde. Hinzukommend sind IP-Adressen für den Menschen schwer zu merken. Mit dieser Problemstellung als Grundlage arbeitete der Ingenieur Peter Mockapetris an einem ersten Lösungsansatz: Domain Name System (DNS). Bei DNS handelt es sich um einen mehr als 30 Jahre alten Verzeichnisdienst, welcher über das gleichnamige Protokoll für Menschen merkbare symbolische Namen auf IP-Adressen abbildet. DNS wurde erstmalig im Jahr 1983 in den beiden Request For Comments (RFC)s RFC 882[51] und RFC 883[52] beschrieben. Damals befanden sich die DNS-Einträge, die Abbildungen von lesbarer Adresse auf IP-Adresse, noch verteilt auf allen Servern des frühen Internets und wurden vom Network Information Center (NIC) verwaltet und mit dem Dateiübertragungsprotokoll File Transfer Protocol (FTP) synchronisiert[49]. In der damaligen Zeit stellte sich dies als ein Flaschenhals für das Internet heraus, da die Anzahl der Server im Netzwerk exponentiell zunahm. Deshalb wurden nur vier Jahre später Überarbeitungen von DNS veröffentlicht. Diese Überarbeitungen wurden in den RFCs 1034 und 1035 erläutert und bilden die Grundlage für DNS wie es heute bekannt ist und auch eingesetzt wird. Der heutige Ansatz

verläuft dezentraler, als es damals der Fall gewesen ist. Anstatt die DNS-Einträge auf allen Knoten des Internets zu verteilen und zentral vom NIC aus zu steuern, existieren heute mehrere hierarchische Verwaltungsebenen. Dazu wird der Fully-Qualified Domain Name (FQDN) hierarchisch gegliedert. Ein Beispiel für einen FQDN ist *akira.rz.tu-clausthal.de*. Die Abbildung 3.3 veranschaulicht die Gliederung dieses FQDN in die einzelnen Bestandteile.

Abbildung 3.3: Bestandteile eines FQDN mit optionalem Hostname und Third-Level Domain

Eine Angabe des Hostname oder einer Third-Level Domain ist optional. Es können auch weitere Domains hinzugefügt werden. So ist zum Beispiel auch eine 6th-Level Domain möglich. Die maximale Anzahl der Subdomains ist in keinem der DNS RFCs spezifiziert. Deshalb ist die maximale Anzahl abhängig vom DNS-Server, der die Domains ausliefert. Nach RFC 1035 ist die maximale Länge eines FQDN aber auf 255 Bytes begrenzt[50, siehe Section 2.3.4], was die maximale Anzahl von Subdomains zumindest stark einschränkt. Der Begriff Subdomain umfasst alle Domains unter der Top-Level Domain (TLD). FQDNs unterliegen einer Baumstruktur. Dessen Wurzel ist die *Root Domain*, meistens symbolisiert durch einen einfachen Punkt. Eine Ebene direkt darunter sind die TLDs. Diese werden in der Regel von den NICs einzelner Länder verwaltet. Von Ländern verwaltete TLDs haben meist Länderabkürzungen wie *de* (Deutschland), *jp* (Japan) oder *cn* (Volksrepublik China). Es existieren aber auch militärische oder akademische TLDs wie *edu* und *mil*. In Deutschland ist die DENIC eG verantwortlich für Domains mit *de*-Endung. In den letzten 5 Jahren kamen auch noch neue TLDs hinzu, welche nicht an Länder geknüpft sind. Darunter fallen Markennamen wie *BMW*, *Audi* oder *Deutschepost* oder markenfreie Namen wie *academy*, *fun* oder *house*[46]. Die Vergabe der TLDs erfolgte direkt über die Internet Corporation for Assigned Names and Numbers (ICANN). Der jeweilige Käufer ist Nutzer der jeweiligen TLD, solange er die jährlichen Gebühren dafür bezahlt. Ein Nutzer einer TLD kann beliebig viele Subdomains erstellen für den Eigenbedarf, diese weiterverkaufen oder gar verschenken. Weitere wichtige Elemente des DNS-Protokolls sind *Name Server* und *Resolver*[49, siehe Section 2.4]. *Name Server* sind Server-Programme, welche Informationen zur Namensauflösung enthalten. Ergo halten *Name Ser-*

ver Domain-IP-Tabellen für ihre hierarchische Ebene vor und cachen sonstige Abbildungen, wie in etwa die Abbildung von DNS-Name auf IP-Adressen, die durch sie hindurch geschleust werden. Wenn kein Eintrag zu dem angefragten FQDN existiert, wird der nächsthöhere DNS-Server gefragt. An oberster Stelle steht ein *Root Nameserver Cluster*[81]. Die *Root Name Server* kennen die Namen und IP-Adressen aller *Name Server*, die für die TLD verantwortlich sind. *Resolver* übernehmen die Rolle des Clients, sie senden DNS-Anfragen an *Name Server*. Diese Anfragen erfolgen in den meisten Fällen über das UDP an den Zielport 53 des Name-Servers[50, Siehe Section 4.2.1]. Es ist allerdings auch DNS über TCP spezifiziert[50, Siehe Section 4.2.2]. Letzteres ist nötig für die DNS-Erweiterungen DNS-over-Transport Layer Security (TLS) (Port 853) und DNS-over-HTTPS (Port 443), welche das Protokoll HTTPS für DNS nutzt und somit Verschlüsselung und Authentifizierung bereitstellt [35] (DNS-over-HTTPS ist zum Zeitpunkt dieser Arbeit noch nicht endgültig spezifiziert). Dieses Bestreben, das DNS-Protokoll sicherer zu machen, unterstreicht dessen Wichtigkeit für das Internet. Ohne DNS wären diverse vernetzte Anwendungen nicht möglich. Jeder Webbrowser oder Mailclient beispielsweise erzeugt diverse Namensauflösungen durch den lokalen Resolver. Ist nur eine dieser Anfragen fehlerhaft oder stark verzögert ist das für den Nutzer augenblicklich zu merken. Im Fall von fehlerhaften Anfragen wäre eine Auflösung der Domain nicht möglich und eine Verbindung zu der assoziierten IP-Adresse würde fehlschlagen. Der Webbrowser würde in diesem Fall einen Fehlerbildschirm anzeigen. Bei stark verzögerten DNS-Auflösungen dauert der Aufbau einer frisch angeforderten Webseite länger als üblich. Wie wichtig schnelle Antwortzeiten eines Computers sind, hat Walter Doherty bereits im Jahre 1982 mit seinem Paper *The Economic Value of Rapid Response Time* deutlich gemacht. Doherty beobachtete einen signifikanten Anstieg an Benutzerinteraktionen, wenn die Latenz der Anfrage (in diesem Fall bezogen auf normale Computereingaben) unter einen bestimmten Wert fiel[16]. Bezogen auf die Latenz von DNS-Anfragen hieße das, dass der Benutzer bei zu hoher Latenz massiv bei seiner Arbeit ausgebremst wird. Dies führt nicht nur zu Frustration beim Nutzer, sondern auch zu Verschwendung seiner bezahlten Arbeitszeit und damit zu einer geringeren Produktivität.

3.3 Hypertext Transport Protocol (HTTP)

Hypertext Transport Protocol (HTTP) ist ein auf TCP/IP aufbauendes Datenübertragungsprotokoll, welches in der Andwendungsschicht des OSI-Modells operiert. Der am meisten verbreiteste Anwendungszweck für das Protokoll ist das Ausliefern von Webseiten über Port 80. HTTP wurde erstmals im Jahre 1996 als RFC 1945 spezifiziert, nachdem es bereits fast sechs Jahre im Internet im Einsatz gewesen war[11]. Im Laufe der Jahre kamen weitere Versionen, sowie diverse Erweiterungen für das Protokoll hinzu. Darunter Erweiterungen für Kompression (um die Datenübertragungsrate zu erhöhen), Verschlüsselung, Caching und Authentifikation. Seit 2015 ist die aktuelle Version HTTP/2, welche in RFC 7540 standardisiert werden[9]. HTTP ist ein zustandsloses Pro-

tokoll, dementsprechend werden mehrere Anfragen getrennt voneinander und ohne Kontext zu einander bearbeitet. Es wird keine Session aufgebaut und keine Sitzungsinformationen verwaltet. Persistente Sitzungsinformationen werden durch *Cookies* übertragen. *Cookies* sind im HTTP-Header des HTTP-Response befindliche Schlüssel-Werte-Paare, welche der Sitzung eine eindeutige *Session ID* zuweisen[43][4][42]. Zur Interaktion mit dem Server besitzt HTTP mehrere Methoden. Die häufigsten sind:

GET um an mit der Uniform Resource Locator (URL) verknüpfte Informationen zu kommen. Die Antwort auf einen **GET**-Aufruf beinhaltet den Header und den Body des HTTP-Response[25, Siehe Section 9.3].

POST wird benutzt, um Informationen an den Server zu senden. Diese Informationen werden im HTTP-Header als Schlüssel-Wert-Paare übermittelt[25, Siehe Section 9.5]. Dies eignet sich besonders für kritische Informationen, wie zum Beispiel Passwörter. Da der HTTP-Header bei HTTPS verschlüsselt wird. Die eigentliche Funktion des **POST**-Aufrufs ist dem Server überlassen[25, Siehe Section 9.3]

HEAD ähnlich wie **GET** mit dem Unterschied, dass der Server bei einer Antwort keinen Inhalt mitliefert. Es wird nur der HTTP-Header übertragen[25, Siehe Section 9.4], welcher meistens einen Titel und eine Kurzbeschreibung der Webseite umfasst.

Die Methoden **DELETE**, **PUT** sind nur für eine Representational State Transfer Application Programming Interface (REST-API) relevant und umfassen das Löschen und hinzufügen. Um herauszufinden, welche Methoden ein Server unterstützt, kann die Methode **OPTIONS** verwendet werden. Im Quelltext 3.1 ist eine HTTP-Anfrage mit Antwort und Verbindungsaufbau an den Host `http://tu-clausthal.de` dargestellt. Die eigentliche HTTP-Anfrage beginnt ab Zeile 5 und endet in Zeile 9. Aus Zeile 5 wird ersichtlich, dass es sich um die Version 1.1 von HTTP handelt und der Client via *GET* den Index der Seite *tu-clausthal.de* (siehe Zeile 6) anfragt. Zeile 7 übermittelt den Namen und Version der Software mit, der diese HTTP-Anfrage gesendet worden ist und Zeile 8 definiert, welche Mediatypen in einer Antwort erlaubt sind[25]. Ab Zeile 10 beginnt der HTTP-Response des Servers. Dort werden Protokoll und die Version nochmal bestätigt und in diesem Beispiel ein Statuscode hinzugefügt (301 weist darauf hin, dass der Inhalt der Seite verschoben worden ist und an einem neuen Ort liegt (siehe Zeile 13)). Dazu wird auf eine neue URL verwiesen. Die URL dient als Pfadangabe zur gewünschten Web-Ressource oder Datei. Bei HTTP haben diese Pfadangaben den Präfix *http://*. Zeile 11 gibt das aktuelle Datum, die Uhrzeit und die Zeitzone an und Zeile 12 die Software des Servers und deren Version. Die Zeilen 14 bis 16 enthalten die Information, dass der Server diverse Encodings unterstützt (zum Beispiel Kompression mit dem Kompressionsalgorithmus *gzip*), die Länge des übermittelten Inhalts und der Typ des Inhalts sowie dessen Zeichenkodierung. Die Zeilen 18 bis 21 enthalten den übermittelten Inhalt, hier gekürzt dargestellt.

Quelltext 3.1: Eine HTTP-Anfrage an http://tu-clausthal.de

```
* Rebuilt URL to: http://tu-clausthal.de/
*   Trying 2001:638:605:20:1::2...
* TCP_NODELAY set
* Connected to tu-clausthal.de (2001:638:605:20:1::2)
  port 80 (#0)
> GET / HTTP/1.1
> Host: tu-clausthal.de
> User-Agent: curl/7.59.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Date: Fri, 11 May 2018 23:48:50 GMT
< Server: Apache/2.2.22 (Ubuntu)
< Location: http://www.tu-clausthal.de/
< Vary: Accept-Encoding
< Content-Length: 235
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
[...]
</body></html>
* Connection #0 to host tu-clausthal.de left intact
```

Bei der Verwendung von HTTP können unter anderem folgende Fehlerfälle auftreten:

- Eine zu geringe Bandbreite oder eine zu hohe Latenz führen zu erheblichen Einschränkungen beim Seitenaufbau und damit in einem für den Nutzer zu langsamen Tempo. Im schlimmsten Fall kommt es zu einem Timeout und es wird der HTTP-Statuscode 408 zurückgegeben[24, Siehe Section 10.4.9].

- Der Webserver ist nicht in der Lage die Anfrage zu verarbeiten. Dieser Fehler terminiert mit dem Statuscode 503[24, Siehe Section 10.5.4].

- Die mit der angegebenen URL verknüpfte Web-Ressource konnte nicht gefunden werden. Als Statuscode wird 404 verwendet[24, Siehe Section 10.4.5].

  Weitere Fehler und Statuscodes können dem RFC 2068 entnommen werden[24, Siehe Section 10].

3.4 Hypertext Transport Protocol Secure (HTTPS)

Bei HTTPS handelt es sich um das Protokoll HTTP mit einer zusätzlichen Schicht zur Verschlüsselung und Herstellung von Datenintegrität. Dafür wird das Verschlüsselungsprotokoll TLS benutzt, teilweise auch noch unter der Bezeichnung Secure Sockets Layer (SSL) bekannt. Dabei ist anzumerken, dass SSL das Vorgängerprotokoll von TLS ist. Für den Verbindungsaufbau wird bei HTTPS standardmäßig Port 433 benutzt[74, Siehe Section 2.3]. Als URL-Präfix dient *https://*. Bei der Verschlüsselung und Herstellung der Datenintegrität bleibt die eigentliche HTTP-Syntax wie sie ist, ergo handelt es sich um eine Verschlüsselung der einzelnen HTTP-Pakete für Request und Response. Dazu verschickt der Client beim Verbindungsaufbau über Port 433 ein *TLS ClientHello* an den Server, woraufhin ein TLS-Handshake initiiert wird. Dieser Handshake beinhaltet die Überprüfung der Integrität des WebServers unter Betrachtung des TLS-Zertifikats. Das Zertifikat ist ein öffentlicher Schlüssel, welcher mit einer digitalen Signatur einer Zertifizierungsstelle beglaubigt worden ist. Der Client besitzt eine Datenbank mit gültigen Zertifizierungsstellen und vergleicht so das signierte Zertifikat des Servers mit allen gespeicherten Zertifizierungsstellen. Wurden keine Mängel festgestellt, beispielsweise ein abgelaufenes Zertifikat oder eine fremde Zertifizierungsstelle, geht der Client davon aus, dass der Server die Identität besitzt, die er vorgibt zu haben. Dies ist insofern problematisch, als in der Vergangenheit wiederholt bei Zertifizierungsstellen eingebrochen worden ist und private Schlüssel zum Signieren von gültigen Zertifikaten gestohlen worden sind[1]. Nach der Integritätsprüfung folgt der eigentliche Aufbau einer verschlüsselten Verbindung. Für den Aufbau existieren derzeit zwei mögliche Verfahren. Zum einen der RSA-Handshake und zum anderen der Diffie-Hellman-Handshake[83]. Beim RSA-Handshake wird vom Client ein symmetrischer Schlüssel erzeugt, dieser wird mit dem öffentlichen Schlüssel des Servers verschlüsselt und das Ergebnis an den Server übermittelt. Der Server entschlüsselt den symmetrischen Schlüssel unter Zuhilfenahme seines privaten Schlüssels. Damit ist eine sichere Verbindung aufgebaut und der Client und der Server sind in der Lage, sich mithilfe des symmetrischen Schlüssels verschlüsselte Nachrichten zu senden. Beim Diffie-Hellman-Handshake dagegen werden die öffentlichen Schlüssel beider Gesprächspartner ausgetauscht und mit dem jeweiligen im Besitz befindlichen privaten Schlüssel ein gemeinsamer symmetrischer Schlüssel berechnet. Dieser symmetrische Schlüssel verlässt im Gegensatz zum RSA-Handshake niemals den Server oder Client. Außerdem ist es möglich, für jede Session einen neuen flüchtigen symmetrischen Key zu erzeugen. Dies wird ermöglicht, in dem bei jeder neuen Session ein neues Schlüsselpaar erzeugt wird. Deshalb ist der Diffie-Hellman-Handshake als sicherer anzusehen, da der gemeinsame symmetrische Schlüssel niemals übertragen wird (auch nicht verschlüsselt) und neue Sessions immer mit einem neuen Schlüssel versehen werden. Letzteres ermöglicht Perfect Forward Secrecy (PFS). Durch PFS ist es einem Angreifer nicht möglich, ältere aufgezeichnete Verbindungen zu entschlüsseln, auch wenn er in den

Besitz eines privaten Schlüssels gelangt. Die Fehlerfälle bei HTTPS entsprechen denen von HTTP, erweitert um einige Fehler bezogen auf TLS:

- Ein TLS-Zertifikat ist ungültig
- Es kommt keine Verbindung auf Port 443 zustande, weil der Webserver kein HTTPS unterstützt.

3.5 Common Internet File System (CIFS)

CIFS ist ein von der Firma Microsoft 1996 eingeführtes Datentransferprotokoll auf Basis von Server Message Block[8] und Network Basic Input Output System (NetBIOS) über TCP/IP. Das Protokoll ist nicht nur auf beschränkt, sondern kann auch für Druckerfreigaben, Windows-RPC (ein von Microsoft eingeführtes Protokoll, um Code aus der Ferne auszuführen) und den NT-Domänendienst (ein von Microsoft eingeführter Dienst zur Authentifizierung von Computern und Nutzern) verwendet werden. Für die in dem vorherigen Kapitel genannte Problemstellung ist allerdings nur der Dateitransfer via SMB von Relevanz. Im Gegensatz zu HTTP ist CIFS ein sessionbehaftetes Protokoll. Der CIFS-Server ordnet also jeder Verbindung eine Sitzung zu, die einem Client genau zugeordnet werden kann. Darüber sind diverse Operationen möglich wie Authentifizierung, Verschlüsselung oder *Locking*[77, S. 16]. Beim *Locking* wird der Zugriff auf einen Nutzer beschränkt, um deren Beschädigung durch mehrfaches Schreiben an derselben Stelle zu vermeiden. Um dies zu verhindern, setzt der CIFS-Server ein *Lock* auf diese Datei und lässt nur einen Client in diese Datei schreiben. Der eigentliche Transfer der Dateien wird mit TCP vor Datenübertragungsfehlern geschützt. Die von der Internet Assigned Numbers Authority (IANA) für CIFS vergebene Portnummer ist: 445[77, S. 19]. Desweiteren hat TCP den Vorteil, dass es *full-duplex* ist. *Full-duplex* bedeutet, dass TCP in der Lage ist, gleichzeitig Daten zu empfangen und zu senden. Abbildung 3.4 zeigt den Aufbau eines solchen SMB-Pakets mit TCP-Header. Der Header beginnt mit einem Byte aus Nullen und der anschließenden Länge des SMB-Pakets. Danach folgt in 32-Byte-Blöcken die eigentliche Nachricht. Die Länge des Pakets wird als drei Byte Integer in *Network Byte Order* repräsentiert[77, S. 21]. *Network Byte Order* entspricht dem *Big Endian Format*, bei dem das höchstwertige Byte zuerst gespeichert wird. Bei der Verwendung von CIFS können folgende

Abbildung 3.4: Aufbau eines SMB-Pakets in der TCP-Nutzlast

Fehlerfälle auftreten:

- Die Verbindung zum Server, welcher den CIFS-Share anbietet, weist eine hohe Latenz auf. Dies führt zu langsamen Schreib- und Leseoperationen auf dem Share.
- Der CIFS-Share ist nicht erreichbar.

Kapitel 4

Herleitung eines Lösungsansatzes

4.1 Anforderungsanalyse

Nachfolgend werden die ermittelten funktionalen und nicht-funktionalen Anforderungen erläutert. Funktionale Anforderungen stellen das "eigentliche Systemverhalten und die jeweiligen Funktionen des zu erstellenden Produkts"[75, S. 20] dar, also die grundlegenden Aufgaben der Software im Bezug auf die Problemstellung. Die nichthypfunktionalen Anforderungen dagegen sind besonders. Sie umfassen Anforderungen wie Sicherheit, nachträgliche Erweiterbarkeit, Testbarkeit, also Anforderungen, welche erst nach der Entwicklung mess- oder testbar werden[23, S. 292]. Um die funktionalen und nichthypfunktionalen Anforderungen besser einordnen zu können, werden folgende Schlüsselwörter zum Kennzeichnen für Anforderungen nach RFC 2119[13] definiert:

MUSS ist eine absolute Anforderung an die Software. Alle Anforderungen, die mit **MUSS** markiert sind, **MÜSSEN** implementiert werden.

DARF NICHT ist gleichbedeutend zu **VERBOTEN**.

SOLL ist eine Anforderung, die implementiert werden **SOLLTE**, aber nicht **MUSS**. Dies ist der Fall bei Anforderungen, welche aus nachvollziehbaren Gründen nicht implementiert werden.

SOLL NICHT ist gleichbedeutend zu **NICHT EMPFOHLEN** und beschreibt Anforderungen, die nicht erfüllt werden sollten, wenn sie vermieden werden können.

KANN ist eine Anforderung die implementiert werden **KANN**. Diese Art von Anforderungen sind zusätzliches Extra und nicht nötig für die Grundfunktion der Software.

Anmerkung: Das RFC 2119 ist im Original in Englisch. Ich habe mich zur übersetzung der Schlüsselwörter auf die Übersetzung der Schweizer Firma Adfinis SyGroup AG gestützt[26].

4.1.1 Funktionale Anforderungen

FA1: Überwachung von DNS	**Priorität: MUSS**
Die Software **muss** die Erreichbarkeit mehrerer DNS-Server überprüfen können. Ebenfalls **muss** die Latenz einer DNS-Anfrage gemessen werden können.	

Tabelle 4.1: Funktionale Anforderung FA1

FA2: Überwachung von HTTP	**Priorität: MUSS**
Die Software **MUSS** die Erreichbarkeit mehrerer HTTP-Server überprüfen können. Ebenfalls **muss** die Latenz einer HTTP-Anfrage gemessen werden können.	

Tabelle 4.2: Funktionale Anforderung FA2

FA3: Überwachung von HTTPS	**Priorität: MUSS**
Die Software **MUSS** die Erreichbarkeit mehrerer HTTPS-Server überprüfen können. Ebenfalls **muss** die Latenz einer HTTPS-Anfrage gemessen werden können.	

Tabelle 4.3: Funktionale Anforderung FA3

FA4: Überwachung von CIFS	**Priorität: SOLL**
Die Software **SOLL** die Zeit messen können, die zwischen einer CIFS-Abfrage und der Antwort von einem CIFS-Server vergeht.	

Tabelle 4.4: Funktionale Anforderung FA4

FA5: Speicherung von Performance-Daten in einer Datenbank	**Priorität: MUSS**
Das System **MUSS** die gesammelten Performance-Daten zur weiteren Auswertung an eine Datenbank übertragen.	

Tabelle 4.5: Funktionale Anforderung FA5

FA6: Grafische Aufbereitung	**Priorität: MUSS**
Die vom System zur Datenbank gesendeten Performance-Daten **MÜSSEN** für die Administratoren grafisch aufbereitet werden. Diese Graphen **MÜSSEN** via Port 80 (HTTP) und Port 443 (HTTPS) erreichbar sein.	

Tabelle 4.6: Funktionale Anforderung FA6

FA7: Bandbreitenmessung	**Priorität: MUSS**
Das System **MUSS** in der Lage sein Bandbreitenmessungen anhand des Durchsatzes vorzunehmen.	

Tabelle 4.7: Funktionale Anforderung FA7

FA8: Native 1 Gigabit Ethernet Schnittstelle	**Priorität: SOLL**
Die Hardware der Messpunkte **SOLL** über eine native 1 Gigabit Ethernet Schnittstelle verfügen.	

Tabelle 4.8: Funktionale Anforderung FA8

FA9: Kontrollierbarkeit	**Priorität: MUSS**
Das zu entwickelnde System **MUSS** bei wachsendem Komplexitätsgrad und Anzahl von Messpunkten steuerbar und kontrollierbar bleiben. Dies betrifft auch die Komponenten Datenbank und Visualisierungslösung.	

Tabelle 4.9: Funktionale Anforderung FA9

4.1.2 Nicht-funktionale Anforderungen

NFA1: Wahl der Programmiersprache	**Priorität: SOLL**
Das System **SOLL** in einer dem Rechenzentrum der TU Clausthal gängigen Programmiersprache entwickelt werden. Folgende Programmiersprachen werden im Rechenzentrum der TU Clausthal täglich benutzt: • Python • Bash • PHP • Javascript	

Tabelle 4.10: Nicht-Funktionale Anforderung NFA1

NFA2: Niedrige Beschaffungskosten	**Priorität: SOLL**
Die Hardware der Messpunkte **SOLL** möglichst günstig in der Beschaffung sein.	

Tabelle 4.11: Nicht-Funktionale Anforderung NFA2

NFA3: Sicherheit	**Priorität: SOLL**
Das System **SOLL** sicher konzipiert sein. Alle Übertragungen, welche sensible Daten übertragen, **SOLLEN** mit gängigen als sicher eingestuften Algorithmen verschlüsselt sein.	

Tabelle 4.12: Nicht-Funktionale Anforderung NFA3

NFA4: Horizontale Skalierbarkeit	**Priorität: MUSS**
Das zu entwickelnde System **MUSS** horizontal skalierbar sein.	

Tabelle 4.13: Nicht-Funktionale Anforderung NFA4

4.2 Systemarchitektur

Auf Grundlage der Problemstellung und der Anforderungen ergibt sich die folgende Aufstellung von Komponenten für die Systemarchitektur:

Datenbank Eine Datenbank zur Speicherung der gewonnenen Messdaten.

Konfigurationsmanagement Ein Subsystem zur Verwaltung, Kontrolle und Konfiguration der einzelnen Messsensoren, der Datenbank und der Visualisierungslösung.

Visualisierungslösung Das Subsystem zur Visualisierung der, von den Messsensoren, gewonnenen Messdaten.

Messsensor Ein verteilter Messpunkt, um die in den Anforderungen spezifizierten Daten zu gewinnen.

Aus der Komponentenaufstellung ergeben sich die Assoziationen zwischen den einzelnen Komponenten. Das Konfigurationsmanagement umspannt alle Komponenten und sorgt für deren Konfiguration, Verwaltung und Kontrolle. Dadurch ist es möglich die Mean Time To Recover (MTTR) so kurz wie möglich zu halten. Die MTTR bezeichnet die mittlere Zeit die benötigt wird um ein

System im Fehlerfall wiederherzustellen[12]. Nur durch den Einsatz von Konfigurationsmanagement auf allen Systemkomponenten ist es möglich diese, im Falle eines Systemausfalls, innerhalb wenigen Minuten automatisiert aus den Konfigurationsdateien wieder herzustellen. Außerdem ergibt sich durch die Verwendung eines Konfigurationsmanagement mit Versionsverwaltung eine Historie der Veränderungen an dieser Komponente. Die Konfiguration ist nachvollziehbar und identisch reproduzierbar. Die einzelnen Messensoren senden ihre Messdaten zur Datenbank. Der Webserver mit der Visualisierungslösung stellt diese Daten für den Nutzer aufbereitet grafisch dar.

Abbildung 4.1: Erster stark vereinfachter Entwurf einer möglichen Lösung

4.3 Datenbanken

Zur Speicherung der Messdaten steht ein breites Spektrum von Datenbanken zur Verfügung. Das Spektrum reicht von traditionellen, relationalen Datenbanken, wie zum Beispiel MySQL, bis zu modernen NoSQL-Datenbanken, wie Key-Value-Stores und Time Series Databases (TSDBs). Dieses Kapitel versucht einen Überblick über alle Datenbanktypen zu geben und den passenden Datenbanktyp für das Projekt festzulegen.

4.3.1 Relationale Datenbanken

Bei relationalen Datenbanken handelt es sich um tabellenbasierte Datensätze. Als Grundlage dieser Datenbanken dienen Relationen. Relationen sind algebraische Strukturen in Form von n-Tupel. Also in sich selbst abgeschlossene Tautologien. Für Schreib- und Leseoperationen auf diesen durch Relationen beschrieben Tabellen wird eine formale Sprache verwendet: die relationale Algebra. Jede Tabelle wird beim Relationenmodell als eine Menge untergeordneter Tupel aufgefasst[48, S. 4]. Ein weiteres Merkmal von relationalen Datenbanken ist die Verwendung von Primärschlüsseln (Englisch: primary key). Durch Primärschlüsseln ist jeder Eintrag in einer relationalen Datenbank eindeutig identifizierbar. Um Datenbankeinträge abzufragen, zu verändern und in die Datenbank zu transferieren, wird die international standardisierte Datenbankbeschreibungssprache Structured Query Language (SQL) eingesetzt. Die nachfolgende Tabelle namens *Studenten* beschreibt einen Datenbankeintrag in einer relationen Datenbank:

Matrikelnummer	Vorname	Nachname
1	Alice	Alcatraz
2	Bob	Bounty
3	Charlie	Echo

In der oben abgebildeten Datenbank ist nur eine Tabelle namens *Studenten* vorhanden. Auf dieser Tabelle können mit SQL diverse Operationen durchgeführt werden. Quelltext 4.1 zeigt das Hinzufügen neuer Daten, das Abrufen von Daten, das Manipulieren von Daten und das Erzeugen einer neuen Tabelle. Als Implementierung für das Beispiel Quelltext 4.1 wurde *SQLite* verwendet. *SQLite* ist eine quelloffene und äußerst leichtgewichtige Datenbank[78]. In der ersten Zeile wird durch das *INSERT* Statement ein neuer Student zur Tabelle *Studenten* mit der Matrikelnummer 4, dem Vornamen Dana und dem Nachnamen Foxtrott hinzugefügt. In der zweiten Zeile werden alle Vornamen aus der Tabelle *Studenten* ausgegeben und in der siebten Zeile wird der Nachname vom Studenten namens Charlie zu Alcatraz geändert. Zeile 8 zeigt das Erzeugen einer neuen Tabelle *Professoren* mit der MitarbeiterID als aufsteigenden Primärschlüssel und den Einträgen zu Vorname und Nachname.

Quelltext 4.1: Verwendung von Structured Query Language

```
1 sqlite> INSERT INTO Studenten VALUES (4,'Dana','Foxtrott
    ');
2 sqlite> select Vorname from Studenten;
3 Alice
4 Bob
5 Charlie
6 Dana
7 sqlite> UPDATE Studenten SET Nachname = "Alcatraz" WHERE
    Matrikelnummer = 3;
8 sqlite> CREATE TABLE Professoren(MitarbeiterID INTEGER
    PRIMARY KEY ASC, Vorname, Nachname);
```

Bekannte Implementierungen für relationale Datenbanken sind:

- MySQL
- MariaDB
- SQLite
- Microsoft SQL Server

4.3.2 NoSQL-Datenbanken

NoSQL-Datenbanken sind zunächst ein grober Sammelbegriff für alle Datenbanken die kein relationales Konzept verfolgen und somit nicht durch die Sprache SQL beschrieben werden können. In den meisten Fällen handelt es sich bei NoSQL-Datenbanken um einfache Key-Value-Stores. Key-Value-Stores sind Ansammlungen von Daten in n-Tupeln mit einem Schlüssel und einem oder mehrerer Werte. Sie ähneln demnach also einem Wörterbuch. Meist sind diese Ansammlungen von Tupeln in großen Binärdateien (Binary Large Object (BLOB)) oder sogar in einfachen Textdateien gespeichert. Ein Beispiel für eine Key-Value-Datenbank ist *Dynamo* von der Firma *Amazon*. Doch NoSQL-Datenbanken umfassen nicht nur Key-Value-Stores. Es existieren auch NoSQL-Datenbanken zum Speichern von Graphen, großen Mengen an Dokumenten, wie *IBM Notes*, Spaltenorientierten Datenbanken wie *Apache Cassandra* und der Time Series Database (TSDB). TSDBs kommen bei jedem Gebiet zum Einsatz, wo große Mengen zeitveränderlicher Daten gespeichert werden müssen. TSDBs sammeln große Datenmengen und benutzen einen Zeitstempel als Index für diese Daten. Jedes Datum wird demnach einem festen Zeitpunkt zugeordnet. Dies macht eine TDSB besonders beliebt als Datenbank für Wetterdaten, Aktienhandelsdaten, Messdaten oder sonstige Daten die abhängig von der Zeit und in besonders großen Mengen gespeichert werden sollen. Beispiele für TSDBs sind *InfluxDB* und die TDSB im Monitoring-System *Prometheus*. Trotz dieser Kategorisierungen haben NoSQL-Datenbanken mehrere gemeinsame Eigenschaften.

Beispielsweise verzichten NoSQL-Datenbanken auf Eigenschaften von relationalen Datenbanken, um die Skalierbarkeit zu erhöhen[21, S. 13]. Dadurch sind NoSQL-Datenbanken simpler aufgebaut und haben weniger Ansprüche an die gespeicherten Daten. Es müssen keine Datentypen eingehalten oder sonstige Relationen zwischen Daten gewährleistet werden, insbesondere gibt es auch keine ACID-Eigenschaften. Die Kurzform ACID bedeutet[40][69]:

Atomarität Entweder komplette Ausführung der Transaktion oder keine Ausführung.

Konsistenzerhaltung Wenn die Datenbank vor der Transaktion in sich in einem konsistenten Zustand befand, soll sich die Datenbank nach der Transaktion immer noch in einem konsistenten Zustand befinden.

Isolation Transaktionen sind voneinander isoliert. Dies bedeutet, dass laufende Transaktionen sich nicht gegenseitig beeinflussen können.

Dauerhaftigkeit Abgeschlossene Transaktionen sind garantiert dauerhaft in der Datenbank gespeichert.

Stattdessen kommt in NoSQL-Datenbanken das BASE-Paradigma zum Einsatz. BASE ist die Kurzform für *basically available, soft state, eventually consistent* (Deutsch: Grundsätzlich verfügbar, weicher Zustand, eventuell konsistent). Im Gegensatz zu pessimistischen SQL-Datenbanken verfolgen NoSQL-Datenbanken ein optimistisches Paradigma[59, Siehe Seite 51] und akzeptieren einen möglichen Zustand der Inkonsistenz. Außerdem sind NoSQL-Datenbanken im wesentlichen schemafrei[40]. Durch diesen Verzicht von ACID-Eigenschaften erreichen NoSQL-Datenbanken eine hohe Leistung, Skalierbarkeit und Flexibilität[3, Siehe Seite 11].

Erwartete Daten und Wahl der geeigneten Datenbank

Es werden folgende Informationen als Daten erwartet:

- Ein sekundengenauer Zeitstempel.
- Der FQDN oder die IP-Adresse des Messknotens
- Die Messdaten. Dies können sowohl boolesche Werte sein, als aber auch diverse Metriken wie Latenz oder CPU-Auslastung.

Durch die strikte Abhängigkeit zwischen Messdaten und ihrem sekundengenauen Zeitstempel fällt die Wahl auf eine TDSB. TSDBs gehören zur Gruppe der NoSQL-Datenbanken. Folgende TDSB-Implementierungen stehen zur näheren Auswahl:

- *InfluxDB*[37]
- *OpenTSDB*[57]

- *Prometheus*[62]

InfluxDB und *OpenTSDB* sind Vertreter von einer TDSB und vielseitig einsetzbar. Interessant für dieses Projekt ist jedoch *Prometheus*. *Prometheus* beschränkt sich auf Monitoringdaten und erweitert eine TDSB um ein ganzes Ökosystem zum Überwachen von Servern, Container und Netzwerken. Darunter fallen diverse Werkzeuge zum Sammeln von Daten, als auch Software für das *Alerting* von Administratoren im Fehlerfall. Beispiele für Firmen die *Prometheus* nutzen sind: *Soundcloud* (eine große Musikplattform für Künstler), *Docker Inc.* (die Firma hinter der Container-Technologie Docker), *DigitalOcean* (ein weltweit erfolgreicher Hoster von virtuellen Maschinen und Container), *Jodel* (eine unter Studenten beliebte anonyme Microblogging-Plattform)[62].

4.4 Prothemeus

Bei *Prometheus* handelt es sich um keine reine TDSB. *Prometheus* ist vielmehr eine auf Monitoringdaten spezialisierte und mit diversen Komponenten erweiterbare Monitoringanwendung. In diesem Kapitel wird der interne Aufbau von *Prometheus*, dessen Komponenten und deren Zusammenspiel näher erläutert. *Prometheus* wurde ursprünglich im Jahr 2012 von Mitarbeitern des Online-Musikdiensts *Soundcloud* entwickelt[67] und im Jahr 2015 als Opensource Software freigegeben. *Prometheus* entstand aus der Not heraus, hunderte von Mikroservices und tausende von Service-Instanzen in einem firmeninternen Container-Cluster zu überwachen[86]. Dieser Cluster war anfangs eine komplette Eigenentwicklung[86], wurde jedoch später gegen den Cluster-Scheduler Kubernetes ersetzt[34], welcher gängige Container-Technologien wie beispielsweise Docker unterstützt[87]. Seitdem setzt sich *Prometheus* besonders im Cloud-Bereich durch und wurde 2016 sogar in die *Cloud Native Computing Foundation* aufgenommen[15], einer Organisation zur Förderung von Cloud-Software im Opensource-Bereich. Bekannte Mitglieder der *Cloud Native Computing Foundation* sind[44]:

- Amazon AWS (Platinum Member)
- Microsoft Azure (Platinum Member)
- Alibaba Cloud (Platinum Member)
- Google Cloud (Platinum Member)
- Openstack (Opensource Member)
- RedHat (Platinum Member)

4.4.1 Datenspeicherung in Prometheus

Daten werden in *Prometheus* als Tupel aus 64-bit Fließkommazahlen und einem Millisekunden genauen Zeitstempel gespeichert[61], im *Prometheus*-Um-

feld auch Metrik genannt. Diese Tupel werden mit einem Identifikator versehen. Dieser Identifikator besteht aus einem eindeutigen Namen für die Metrik (zum Beispiel die Latenz in Millisekunden) und einem Key-Value-Store (siehe Quelltext 4.2 Zeile 1; für ein Beispiel siehe Zeile 2). Die Keys werden auch *Labels* genannt. Anhand dieser *Labels* ist es möglich, Daten einer Metrik zu filtern. Zwei Stunden werden diese Metriken von *Prometheus* standardmäßig gesammelt bis sie, zusammen mit Metadaten und einer Indexdatei, in Dateien in einem Verzeichnis abgelegt werden. Metadaten sind Informationen über die gesammelten Daten, darunter fallen die *Labels*, welche beispielsweise Informationen zur Herkunft der gesammelten Metrik beinhalten können. Die Indexdatei speichert die Relation zwischen den Metriknamen und den Metriken, die gestückelt abgelegt werden. Diese Stückelungen werden auch *Chunks* genannt[66] und ermöglichen eine Isolation einzelner Transaktionen, vergleichbar mit dem ACID- Paradigma bei relationalen Datenbanken. Bevor Metriken gespeichert werden, legt *Prometheus* außerdem einen Write-Ahead-Log (WAL) an. Der WAL dient zur Wiederherstellung von Daten nach einem Crash, beispielsweise ein Stromausfall. In diesem Fall werden die zwischengespeicherten und noch nicht umgesetzten Transaktionen auf Konsistenz geprüft und in die TDSB übertragen. Gelöschte Daten werden nicht unwiderruflich gelöscht, sondern in *Tombstone*-Dateien abgelegt. Quelltext 4.3 zeigt eine solche Verzeichnisstruktur, mit einzelnen *Chunks*, WAL-Elementen, Metadaten und *Tombstone*-Dateien. Die *Prometheus*-Datenbank arbeitet demnach dateibasiert.

Quelltext 4.2: Prometheus Datenformat und Beispiel

```
1 <metric name>{<label name>=<label value>, ...}
2 http_requests_total{service="service", server="www.tu-
    clausthal.de", env="production"}
```

Quelltext 4.3: Beispiel für die Datenspeicherung in Prometheus

```
./data/01BKGV7JBM69T2G1BGBGM6KB12
./data/01BKGV7JBM69T2G1BGBGM6KB12/meta.json
./data/01BKGV7JBM69T2G1BGBGM6KB12/wal
./data/01BKGV7JBM69T2G1BGBGM6KB12/wal/000002
./data/01BKGV7JBM69T2G1BGBGM6KB12/wal/000001
./data/01BKGTZQ1SYQJTR4PB43C8PD98
./data/01BKGTZQ1SYQJTR4PB43C8PD98/meta.json
./data/01BKGTZQ1SYQJTR4PB43C8PD98/index
./data/01BKGTZQ1SYQJTR4PB43C8PD98/chunks
./data/01BKGTZQ1SYQJTR4PB43C8PD98/chunks/000001
./data/01BKGTZQ1SYQJTR4PB43C8PD98/tombstones
./data/01BKGTZQ1HHWHV8FBJXW1Y3W0K
./data/01BKGTZQ1HHWHV8FBJXW1Y3W0K/meta.json
./data/01BKGTZQ1HHWHV8FBJXW1Y3W0K/wal
./data/01BKGTZQ1HHWHV8FBJXW1Y3W0K/wal/000001
./data/01BKGV7JC0RY8A6MACW02A2PJD
./data/01BKGV7JC0RY8A6MACW02A2PJD/meta.json
./data/01BKGV7JC0RY8A6MACW02A2PJD/index
./data/01BKGV7JC0RY8A6MACW02A2PJD/chunks
./data/01BKGV7JC0RY8A6MACW02A2PJD/chunks/000001
./data/01BKGV7JC0RY8A6MACW02A2PJD/tombstones
```

4.4.2 Komponenten

Die meisten *Prometheus*-Komponenten sind in der Sprache *Go* (auch bekannt als *Golang*) geschrieben. *Go* ist eine von der Firma *Google* entwickelte, kompilierbare Programmiersprache, welche sich vorallem dadurch auszeichnet, dass sie sicherer als die Programmiersprache *C* ist und eine gute Portierbarkeit aufgrund von statischen Binärdateien besitzt. Durch diese Eigenschaften setzt sich *Go* immer stärker, inbesondere im Cloud-Bereich und im *Kubernetes*-Umfeld, durch[41]. So schreibt der *Go*-Blog, dass *Go* mittlerweile von allen global agierenden Cloud-Firmen eingesetzt wird, darunter *Amazon AWS*, *Microsoft Azure* und *Google Cloud*[22]. Das Branchenanalyseunternehmen *RedMonk* listet eine Vielzahl von *Go*-basierten Softwareprodukten darunter[10]:

- Docker
- Kubernetes (Der *Amazon AWS* Dienst *Amazon EKS* nutzt beispielsweise Kubernetes und Docker)
- gRPC
- fluentd
- Grafana

- Prometheus

Alle diese Softwareprodukte werden gefördert durch die *Cloud Native Computing Foundation*[44]. Abbildung 4.2 zeigt eine Übersicht über die wichtigsten *Prometheus* Komponenten. *Prometheus*' Kernkomponente ist der *Prometheus Server*. Der *Prometheus Server* beinhaltet eine TDSB, einen HTTP-Server und ein Modul zum Anfordern von Messdaten (im weiteren Verlauf *Retrieval* genannt). Der HTTP-Server bietet eine grafische Oberfläche via HTML an, um Queries auf der TDSB zu testen und Graphen anzuzeigen. Abbildung 4.3 zeigt die grafische Oberfläche mit einem Beispiel-Query und Graphen. Außerdem bietet der HTTP-Server eine REST-API an, an der die restlichen Komponenten andocken können. Weitere Komponenten sind der *Alertmanager*, das *Pushgateway*, die *Exporter* und das bereits erläuterte und im *Prometheus Server* integrierte *Prometheus Web UI*. Der *Alertmanager* ist verantwortlich für das Alarmieren der Benutzer, wenn Messdaten vom Benutzer bestimmte Richtwerte überschreiten. Unterstützt werden vom *Alertmanager* direkt keine Kommunikationswege. Viel mehr wird Fremdsoftware, beispielsweise für den Mailtransfer, an den *Alertmanger* angedockt. Das *Pushgateway* bietet einen statischen Bezugspunkt und Puffer, von dem der *Prometheus Server* mit dem *Prometheus Retrieval*-Modul Messdaten abholen kann. Dies ist nützlich, wenn der *Prometheus Server*, statt die Messdaten abzuholen, die Messdaten zugeschoben bekommen soll. Diese Messdaten werden zum *Pushgateway* geschoben und *Prometheus Server* holt die Messdaten dort ab. Weitere Komponenten sind die *Exporter*. *Exporter* sind Applikationen, welche Messdaten vom Host über offene Schnittstellen sammeln und diese via einem HTTP-Server dem *Prometheus Server* zur Verfügung stellen. Die *Exporter* exportieren demnach die gewonnenen Messdaten an das *Prometheus Retrieval*-Modul, welches die Daten in die TDSB importiert. Es gibt eine Vielzahl von *Exportern* und die Zahl ist stetig steigend. Prominente Beispiele für *Exporter* sind:

- *Node Exporter*
- *Blackbox Exporter*

Node Exporter werden auf Hosts platziert und liefern allgemeine Messdaten von diesem Host. Außerdem werden diverse Services, die auf dem Host laufen, automatisch erkannt (auch bezeichnet als *Service Discovery*). Für das sammeln der Performance- oder Messdaten greift der *Node Exporter* auf Schnittstellen im Kernel des Betriebssystems zurück. Der Linux Kernel beispielsweise bietet Statistiken und Performance-Daten via Dateien in den Ordnern `/sys/` oder `/proc/sys/` an. Der *Node Exporter* liest aus diesen Dateien und exportiert die gewonnenen Messdaten über den eigenen HTTP-Server an das *Prometheus Retrieval*-Modul. Der *Blackbox Exporter* baut ebenfalls einen HTTP-Server als Quelle für den *Prometheus Server* auf, aber Daten werden nicht über das lokale System gesammelt. Stattdessen werden von dem *Blackbox Exporter* aus Tests an entfernten Servern unternommen, wie beispielsweise die Erreichbarkeit von DNS-Servern oder SMTP-Servern. Dazu sendet der *Blackbox Exporter* zum

Beispiel DNS-Pakete an einen DNS-Server und überprüft so, ob der DNS-Server vom *Blackbox Exporter* aus erreichbar ist. Alle Verbindungen zwischen *Prometheus* Komponenten, sind HTTP-Anfragen einer REST-API. Alle Komponenten sind statische ausführbare Binärdateien (*Executables*). Es existieren eine Vielzahl von anderen *Exportern* auf die hier nicht näher eingegangen wird. Eine vollständige Liste findet sich auf der *Prometheus* Webseite. Darunter sind beispielsweise auch *Exporter* um automatisch Performance-Daten aus der *Amazon AWS* Cloud zu exportieren[47]:

- AWS ECS Exporter[28]
- AWS Health Exporter[29]
- AWS SQS Exporter[30]

Dieses Beispiel unterstreicht die Einsatzfähigkeit und Adoption von *Prometheus* im Cloud-Bereich.

Abbildung 4.2: Interner Aufbau des Prometheus Server und dessen externe Komponenten

Abbildung 4.3: Prometheus Web UI Beispiel, welches via Node Exporter gesammelte Messdaten in Form einer Zeitreihe darstellt

4.4.3 PromQL

Prometheus besitzt zum Abfragen der Messdaten eine eigene Query-Language names *PromQL*. *PromQL* ist nicht vergleichbar mit SQL und besitzt keinen relationalen Ansatz. Stattdessen handelt es sich um eine Art Filtersprache, welche mit booleschen Operatoren und diversen Funktionen ergänzt worden ist. Anhand von *PromQL* lassen sich einzelne Messdatensätze auswählen und weiter auswerten oder filtern[65]. Dies ist insbesondere für die grafische Darstellung der Messdaten relevant. Dazu unterstützt die *PromQL* vier verschiedene Datentypen[65]:

Instant Vector ein Set von Messdaten mit demselben Zeitstempel.

Range Vector eine Auswahl von Messdaten über einen größeren Zeitraum.

Scalar ein Fließkommawert.

String ein Array von Chars (zurzeit von *Prometheus* unbenutzt).

Der *Instant Vector* besteht aus einem eindeutigen Namen, einer Metrik und einem Key-Value-Store (siehe auch Quelltext 4.2). Innerhalb eines *Instant Vectors* lassen sich reguläre Ausdrücke (Regex) anwenden. Ein regulärer Ausdruck ist eine Zeichenkette, die der Beschreibung von Mengen von anderen Zeichenketten mithilfe syntaktischer Regeln dient[7]. Für reguläre Ausdrücke verwendet *PromQL* die von der Firma *Google* entwickelte Bibliothek *RE2*[72]. Wird dieser *Instant Vector* um einen *Range Selector* erweitert, handelt es sich um einen *Range Vector*. Ein *Range Selector* ist eine in eckigen Klammern geschriebene Zeitangabe für einen *Instant Vector*. Der *Range Selector* wird dem *Instant Vector* nachträglich angefügt. Zusätzlich lassen sich diese Vektoren durch weitere Funktionen und Operatoren beeinflussen. So bietet *PromQL* eine Vielzahl von weiteren Funktionen, um beispielsweise den Durchschnitt über einen *Range Vector* zu bilden[63]. *PromQL* ist eine harte Voraussetzung um die *Prometheus GUI* zu bedienen, da spätere Queries zur Visualisierung der Messdaten alle in *PromQL* geschrieben werden.

4.5 Grafana

Da die *Prometheus Web UI* eingeschränkt mit Graphen und Visualisierung der Daten umgehen kann und eher eine Testplattform für *PromQL-Anfragen* darstellt, wird für den Zweck der Visualisierung eine weitere Plattform verwendet: *Grafana*[32]. *Grafana* zeichnet sich durch eine hohe Erweiterbarkeit, vollen *Prometheus*-Support und einer Vielzahl an Konfigurationsmöglichkeiten aus. Der Transport der Daten von *Prometheus* zu *Grafana* findet via REST-API und der, von *Prometheus* unterstützten Query-Language *PromQL* statt (Abbildung 4.4 zeigt *Grafanas* Platz in der *Prometheus*-Landschaft).

Abbildung 4.4: Eingliederung von Grafana in Prometheus

Ein weiteres Merkmal von *Grafana* sind *Grafanas* Provisionierungseigenschaften. Durch diverse Einstellungen in *Grafanas* Konfigurationsdateien und Umgebungsvariablen lässt sich *Grafana* gut in IaaS-Umgebungen (IaaS) ausrollen (Umgebungsvariablen sind temporäre oder permanente Variablen im Betriebssystem)[20, Beispiel zum Deployment von Grafana in Amazon AWS]. *Grafana* besitzt beispielsweise nativen Support für *Amazon AWS Cloudwatch*[31].

4.6 Konfigurationsmanagement

In der Vergangenheit wurden Rechnersysteme zunehmend vertikal skaliert, das heißt, die technischen Ressourcen des einzelnen Rechnersystems wurden angehoben (beispielsweise der Random Access Memory (RAM)). Es wurde jedoch zunehmends festgestellt, dass diese Vorgehensweise nicht nur sehr kostenintensiv ist, sondern auch nichts zur Ausfallsicherheit beiträgt. Die Antwort auf dieses Problem ist horizontale Skalierung. Bei horizontaler Skalierung werden weitere Instanzen von Rechnersystemen der Anwendung hinzugefügt. Dieser Ansatz ermöglicht ausfallsichere und äußerst performante Infrastrukturen, vorausgesetzt das zu berechnende Problem ist horizontal skalierbar. Dies ist der Fall bei voneinander unabhängigen Input. Horizontale Skalierung hat allerdings auch andere Ansprüche. So wächst die Anzahl der zu betreuenden Systemen rapide an. Damals wurden solche Systeme noch von Menschen gepflegt, konfiguriert und installiert. Durch die schiere Masse der neuen Systeme ist das nun unmöglich

und zu kostenintensiv geworden. Daraufhin wurden Anwendungen ins Leben gerufen, welche die Arbeiten an mehreren Systemen vereinfachen sollen. Die zwei Leitbegriffe für solche Anwendungen sind *Orchestrierung* und *Konfigurationsmanagement*. Bei der *Orchestrierung* geht es darum, eine hohe Anzahl von Systemen oder Services zu steuern, also gewisse Aufgaben auf den jeweiligen Systemen anzustoßen. *Konfigurationsmanagement* dagegen umfasst die Installation und Konfiguration von Software auf Systemen, im englischsprachigen Raum zusammengefasst *Deployment* genannt. Da im Rechenzentrum der TU Clausthal zur Zeit das auf der Programmiersprache *Python* basierende Orchestrierungs- und Konfigurationsmanagementwerkzeug *Ansible* dominiert, beschränkt sich diese Arbeit auf die Verwendung von *Ansible*. Es gibt jedoch andere populäre Werkzeuge. Darunter fallen beispielsweise *Puppet*, *Saltstack* und *Chef*.

4.7 Ansible

Die Entwicklung an *Ansible* startete im Februar 2012 unter Michael DeHaan als Opensource-Anwendung[19]. Ein Jahr später ging aus dem Projekt die Firma *AnsibleWorks* hervor, welche neben der Entwicklung von Ansible professionellen Support für die Software anbietet und auch an einer webbasierten Benutzerschnittstelle namens *Tower* arbeitet[18]. Im Jahr 2015 wurde die Firma durch den großen Linux-Dienstleister *Red Hat* übernommen[73]. Wenig später begann die Opensource-Gemeinschaft damit, *Ansible* um unzählige Module zu erweitern und so den Funktionsumfang zu vergrößern.

4.7.1 Funktionsweise

Ansible arbeitet mit Templates zur Orchestrierung und Konfiguration von Programmpaketen auf Rechnern und Rechner -Konfigurationsdateien. Die Auszeichnungssprache für die Datenstrukturen, welche die Templates befüllen, heißt Yet Another Markup Language (YAML) (oft auch nur mit YML abgekürzt). YAML zeichnet sich durch für Menschen einfache Lesbarkeit und die Möglichkeit, Kommentare einzufügen, aus[88]. Außerdem besteht die Möglichkeit, in der Programmiersprache *Python* geschriebene Module mit in YAML aufgeschrieben Anweisungen zu nutzen. Diese Module sind quelloffen und werden von einer Community um *Ansible* und der Firma *Red Hat* gepflegt. Die zu pflegenden Systeme, nachfolgend nur noch als Zielsysteme bezeichnet, können in *Ansible* in Gruppen organisiert werden. Als Eingabe für *Ansible* dienen speziell auf *Ansible* zugeschnittene YAML-Dateien. In diesen YAML-Dateien werden Konfigurationsparameter als YAML-Datenstrukturen erfasst. *Ansible* nimmt anschließend diese YAML-Datenstrukturen als Eingabe und führt *Python* oder Kommandos auf der Kommandozeile auf dem Zielsystem aus, um die spezifizierten Konfigurationsparameter auf dem Zielsystem einzupflegen. Je nach Gruppe oder Name des Zielsystems lassen sich unterschiedliche Konfigurationsparameter in *Ansible* laden. Auf diese Weise können komplexe Gebilde von Software-Systemen und Hardware-Konfigurationsparametern abgebildet und Gemeinsamkeiten in der

Orchestrierung und Konfiguration effektiv ausgenutzt werden. Um Konfigurationsdateien für gängige Software wie beispielsweise *OpenSSH* oder Webserver wie *Apache2* mit *Ansible* dynamisch zu generieren wird die auf *Python* basierende Template-Engine *Jinja2* benutzt[39]. Durch *Jinja2* lassen sich komplexe Konfigurationsdateien von Software auf den Zielsystemen in Form von Templates ausdrücken. Das Ziel ist, ein Template für mehrere Konfigurationsvarianten einer Konfigurationsdatei zu haben. Diese Templates werden mit Datenstrukturen in YAML erweitert. Um die Orchestrierung oder Konfiguration zu starten, reicht es, wenn sich *Ansible* auf dem System, welches zur Pflege anderer Rechner und deren Software eingesetzt werden soll, befindet. Nachfolgend wird dieses System als Managementsystem bezeichnet. Das Managementsystem verbindet sich zu den einzelnen Zielsystemen via dem Protokoll Secure Shell (SSH) und arbeitet die auf dem Managementsystem befindlichen Aufgaben für das Zielsystem in Form von YAML-Dateien ab. Dies geschieht durch die Verwendung der Sprache *Python* oder einfach durch Kommandos der Kommandozeile der Zielsysteme. Durch diese Vorgehensweise arbeitet *Ansible agentless*, also ohne einen Agenten auf dem Zielsystem. Dies ermöglicht den Einsatz von *Ansible* auch auf Systemen mit eingeschränkter Leistungsfähigkeit (beispielsweise Switches oder Router). Außerdem arbeitet *Ansible* idempotent, dadurch lassen sich Aufgaben beliebig oft ausführen und führen immer zum gleichen Ergebnis. Dies setzt allerdings voraus, das *Ansible* korrekt konfiguriert wurde.

4.7.2 Organisationsstruktur

Ansible hat eine definierte Struktur zur Speicherung der Konfigurationsdateien, Aufgaben, Datenstrukturen für die Templates und Templates. Dabei bedient sich *Ansible* bei Synonymen aus der Filmbranche, um den Einstieg in das Werkzeug zu erleichtern. *Playbooks* (Deutsch: Drehbücher) werden verwendet, um einzelnen Hosts *Roles* (Deutsch: Rollen) zuzuweisen. Die nötigen Daten zum Abspielen dieser Rollen befinden sich im *Inventory* (Deutsch: Inventar), zu dem die Hosts ebenfalls gehören. Quelltext 4.4 zeigt die Organisationsstruktur eines solchen *Ansible*-Projekts. Die Datei *ansible.cfg* ist die Konfigurationsdatei für *Ansible* und definiert den Pfad zum Inventar, den Rollen und den *Playbooks*. Die Datei *hosts* beinhaltet eine Auflistung aller Zielsysteme mit Eingliederung in Gruppen und Subgruppen. Auf diese Weise lassen sich komplexe Installationen aus Zielsystemen hierarchisch gliedern und Gemeinsamkeiten festmachen. In den Dateien *group_vars* und *host_vars* werden die eigentlichen Daten für die Gruppen und einzelnen Hosts vermerkt. Der Ordner *playbooks* umfasst alle *Playbooks*. Im Beispiel ist das *Playbook configure_hosts* abgebildet. Es handelt sich um eine YAML-Datei. Der Ordner *roles* beinhaltet die Rolle *hello_world* mit den Unterordnern *defaults*, *files*, *tasks* und *templates*. Im Ordner *defaults* befinden sich Standardbelegungen, mit denen Variablen initialisiert werden. Der Ordner *files* umfasst statische Dateien, welche auf den Zielsystemen installiert werden sollen. Die letzten beiden Ordner *tasks* und *templates* umfassen die eigentliche Logik zur Konfiguration (die abzuarbeitenden Aufgabenschritte) und *Jinja2*-Templates. Um eine Konfiguration zu starten, würde der Administrator

das *Playbook* configure_hosts.yml mit dem Kommando *ansible-playbook playbooks/configure_hosts.yml* aufrufen. *Ansible* würde daraufhin das zu den Hosts gehörende Inventar heraussuchen und die in den Rollen und Templates vorhandenen Variablen durch Werte aus den *host_vars* oder *group_vars* ersetzen.

Quelltext 4.4: Organisationsstruktur eines Ansible Projekts

```
.
|-- ansible.cfg
|-- group_vars
|-- hosts
|-- host_vars
|-- playbooks
|   `-- configure_hosts.yml
`-- roles
    `-- hello_world
        |-- defaults
        |   `-- main.yml
        |-- files
        |   `-- static_file.txt
        |-- tasks
        |   `-- main.yml
        `-- templates
            `-- configuration.j2
```

4.7.3 Dateistruktur

Ansible nutzt als zentrale Dateistruktur die Auszeichnungssprache YAML und ergänzt diese durch Elemente der Python-Template-Engine *Jinja2*. YAML unterstützt gängige Datenstrukturen wie in etwa Listen und Wörterbücher (Dictionaries) und Kombinationen davon sowie Kommentare. Höhere Logikebenen werden durch YAML und *Jinja2* erreicht (If/Else, For-Schleifen, While-Schleifen etc). Dadurch ist *Jinja2* in der Theorie Turing-vollständig. Quelltext 4.5 ist ein Beispiel für die Dateistruktur eines *Ansible Tasks*. Der *Task* verteilt SSH-Schlüssel auf eine beliebige Anzahl an Zielsystemen. YAML-Dateien beginnen stets mit drei Querstrichen und enden mit drei Punkten. Mittlerweile ist dies jedoch optional und *Ansible* erkennt YAML-Dateien ohne diese Markierungen (meistens an der Dateiendung *.yml*). *Ansible Tasks* bestehen aus einer Liste von Dictionaries. Jeder Listeneintrag stellt eine Aufgabe dar, welche *Ansible* abzuarbeiten hat. Initialisiert wird ein solcher Eintrag mit dem Namen der Aufgabe und einer Abfolge von weiteren Schlüssel-Wert-Paaren. Bei den Paaren handelt es sich um fest definierte und in Python verfasste *Ansible Module*. In dem Beispiel Quelltext 4.5 sind dies die Module *file* und *template*. Das *file*-Modul dient zum Erstellen von Ordnern, Platzieren von statischen Dateien oder Anlegen von Dateien mit statischen Inhalt. Diesem Modul werden eine Anzahl von Parametern übergeben. In diesem Fall ein Pfad auf einem Linux-

System, der gewollte Zustand *directory* sowie ein Besitzer, eine Gruppe und eine Konfiguration der Dateiberechtigung. *0700* bedeutet volle Lese-, Schreib- und Ausführrechte für den Besitzer der Datei oder des Ordners. Dem zur Folge erstellt diese Zeile einen Ordner mit den im vorherigen Satz erwähnten Dateirechten und dem Besitzer *root*. Das *template*-Modul hingegen liest ein lokales *Jinja2*-Template ein, wertet dieses aus und hinterlegt es auf dem Zielsystem mit den Dateirechten *0600* (Schreib- und Leserechte) für den Benutzer *root* in dem Pfad `/root/.ssh/authorized\_keys`. Quelltext 4.6 zeigt das zum *Task* gehörende *Jinja2*-Template. Die erste Zeile im Template konfiguriert *Jinja2* so, dass alle Leerzeichen oder Tabs am Anfang einer Zeile gelöscht werden. In Zeile 2 ist der Anfang einer For-Schleife in *Jinja2*-Syntax abgebildet. In dieser For-Schleife wird über die Liste *root_ssh_keys* iteriert und jede Iteration wird der Variable *user* als Wert zugewiesen. Der vertikale Strich startet einen Filter. In diesem Fall der Filter *sort*. Die Liste wird durch diesen Filter vor dem Schleifendurchlauf sortiert. Der horizontale Strich am Ende der Anweisung weist die Schleife an, alle Leerzeichen oder Tabs am Ende der generierten Zeile zu entfernen. Innerhalb der For-Schleife wird die Funktion *lookup* aufgerufen, welche eine Information auf dem lokalen Dateisystem einliest. Als Argument für die Funktion wird angegeben, dass die Information aus einer Datei bezogen werden soll und der Pfad zu dieser Datei wird aus der *user*-Schleifenvariable und dem String *../pubkeys/* zusammengebaut. Die letzte Zeile beendet die Schleife. Die Eingabe für dieses Template ist in Quelltext 4.7 abgebildet. Quelltext 4.7 wird dazu in den Ordnern *group_vars* oder *host_vars* abgelegt und definiert die Liste *root_ssh_keys* mit den beiden Einträgen *chris.pub* und *test.pub*

Quelltext 4.5: Beispiel eines Ansible Tasks

```
1 ---
2
3 - name: ensure /root/.ssh exists
4   file: path=/root/.ssh state=directory owner=root group=
      root mode=0700
5
6 - name: add authorized keys for root
7   template: src=authorized_keys.j2 dest=/root/.ssh/
      authorized_keys owner=root group=root mode=0600
8
9 ...
```

Quelltext 4.6: Beispiel eines Jinja2-Templates

```
1 #jinja2: lstrip_blocks: True
2 {% for user in root_ssh_keys | sort -%}
3     {{ lookup('file', '../pubkeys/' + user) }}
4 {% endfor %}
```

Quelltext 4.7: Daten für das Jinja2-Template Beispiel

```
1 ---
2
3 root_ssh_keys:
4   - chris.pub
5   - test.pub
6
7 ...
```

4.7.4 Sicherheit

Ansible nutzt für alle Verbindungen das Fernadministrationsprotokoll Secure Shell (SSH). Dadurch ist die Integrität, Sicherheit und Authentizität der übertragenen Daten sichergestellt. SSH verwendet dazu ein asynchrones Verschlüsselungsverfahren. Auf dem Client wird ein Schlüsselpaar generiert. Dies erfolgt durch moderne kryptografisch sichere Verfahren wie RSA oder Elliptischen Kurven. Der öffentliche Schlüssel wird auf dem Zielsystem platziert, meist geschieht dies durch vorheriges Einloggen auf dem Zielsystem mit einem Passwort. Bei der Verbindung vom Client zum Zielsystem wird durch das asynchrone Verfahren die Authentizität des Servers und des Clients bestätigt und der Zugriff gewährt. Um bestimmte Daten auch auf dem lokalen Dateisystem des Managementsystems zu verschlüsseln wird *Ansible-Vault* verwendet. *Ansible-Vault* ist ein Mechanismus, um Dateien mit einem symmetrischen Schlüssel zu verschlüsseln. Als Algorithmus wird das Verschlüsselungsverfahren Advanced Encryption Standard (AES) verwendet[2].

4.8 Wahl der Hardware

Da das zu entwickelnde Software-System horizontal skalierbar sein soll (siehe Nicht-Funktionale Anforderung 4 (NFA4)), soll das System auch günstig in der Anschaffung sein (NFA2). Dem kommt zu Gute, dass die Software auf den Messpunkten nicht viele Hardware-Ressourcen benötigt. Es wird lediglich genug Speicher für ein kleines Linux mit einigen Kernkomponenten wie *OpenSSH* und der eigentlichen Software für die Messpunkte benötigt. Durch die niedrigen Beschaffungskosten fallen somit bereits größere Computersysteme in Form von Server-Racks oder Tower-PC-Einheiten weg. Übrig bleibt die Klasse der Einplatinenrechner. Einplatinenrechner erfreuen sich in den letzten Jahren immer größerer Beliebtheit und liegen in einem Budget-Level von 10 € bis 100 €. Nachfolgend werden die zwei Einplatinenrechner *Raspberry Pi 3B+* und *Odroid XU4* verglichen. Der *Raspberry Pi 3B+* hat die folgenden Spezifikationen[71]:

- CPU: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- GPU: Broadcom Videocore-IV

- RAM: 1 GB LPDDR2 SDRAM
- Netzwerk: Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
- Bluetooth: Bluetooth 4.2, Bluetooth Low Energy (BLE)
- Speicher: Micro-SD
- GPIO: 40-pin GPIO header
- Anschlüsse: HDMI, 3.5 mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- Dimensionen: 82 mm x 56 mm x 19.5 mm, 50g Gewicht
- Preis: 32.88 britische Pfund[70] (36.45 € Stand: 2018-09-03)

Im Gegenzug dazu die *Odroid XU4* Spezifikationen[55]:

- CPU: Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa core CPUs
- GPU: Mali-T628 MP6(OpenGL ES 3.1/2.0/1.1 and OpenCL 1.2 Full profile)
- RAM: 2 GB LPDDR3 RAM PoP stacked
- Netzwerk: Gigabit Ethernet, kein WLAN
- Bluetooth: nicht vorhanden
- Speicher: Micro-SD
- GPIO: 30-pin GPIO header[53]
- Anschlüsse: 2x USB 3.0, 1x USB 2.0, HDMI 1.4a
- Dimensionen: 83 mm x 58 mm x 20 mm
- Preis: 59 US-Dollar[54] (50,80 € Stand: 2018-09-03)

Der *Raspberry Pi 3B+* überzeugt zwar durch einen niedrigeren Preis, jedoch ist es für den *Raspberry Pi 3B+* unmöglich die theoretische Leistung von 1 Gbit/s zu erreichen, da der Prozessor nur mit einer USB 2.0 Schnittstelle zu der Gigabit Schnittstelle verbunden ist. Deshalb limitiert sich die praktische Geschwindigkeit auf maximal 300 Mbit/s[71]. Da das Projekt jedoch eine native 1 Gigabit Schnitstelle voraussetzt (siehe Tabelle 4.8), fällt die Wahl auf den *Odroid XU4*.

Abbildung 4.5: Darstellung eines Odroid XU4

Kapitel 5

Projektumsetzung

Die folgenden Unterkapitel umfassen die Umsetzung eines ersten Prototyps. Dazu gehören das Erfüllen der Anforderungen, das Aufsetzen eines ersten Test- und Entwicklungssystem, die Versionsverwaltung und der erste Prototype.

5.1 Erfüllung der Anforderungen

Nachfolgend sind die evaluierten Anforderungen und deren Umsetzung beschrieben.

Anforderung	Umsetzung
FA1	Die Überwachung von *DNS* erfolgt durch den *Prometheus Blackbox-Exporter*.
FA2	Die Überwachung von *HTTP* erfolgt durch den *Prometheus Blackbox-Exporter*.
FA3	Die Überwachung von *HTTPS* erfolgt durch den *Prometheus Blackbox-Exporter*.
FA4	*Prometheus* ist zum Stand vom 01.09.2018 nicht in der Lage *CIFS* zu überwachen. Um die Anforderung zu erfüllen, wird der *Prometheus Node-Exporter* benutzt und die darunter liegende Bibliothek *ProcFS* um Funktionalität erweitert.
FA5	Die Performance-Daten werden in einer *TSDB* in *Prometheus* gespeichert.
FA6	Die Grafische Aufbereitung erfolgt durch die Plattform *Grafana*.
FA7	Die Bandbreitenmessung erfolgt durch den *Prometheus Node-Exporter*.
FA8	Wird erfüllt über die *Odroid XU4* Plattform.
FA9	Das System bleibt kontrollierbar durch den Einsatz von *Ansible*.

Tabelle 5.1: Umsetzung der funktionalen Anforderungen

Anforderung	Umsetzung
NFA1	Das für das Deployment verwendete Projekt *Ansible* benutzt *Python*. Für die Erweiterung um *CIFS*-Support ist nur die Verwendung der Programmiersprache *Golang* möglich.
NFA2	Die niedrigen Beschaffungskosten sind durch die Wahl der *Odroid XU4* Plattform gegeben.
NFA3	Die für das Deployment benötigten Passwörter werden mit gängigen Verschlüsselungsverfahren in *Ansible-Vault* hinterlegt. Die Verbindungen zu den Zielsystemen finden verschlüsselt und mit einer Authentizitätsprüfung statt.
NFA4	Das System ist horizontal skalierbar durch den Einsatz von *Prometheus* als Cluster einsetzbare Datenbank und *Ansible* als Automatisierungswerkzeug zum Deployment.

Tabelle 5.2: Umsetzung der nicht-funktionalen Anforderungen

5.2 Test- und Entwicklungsumgebung

Da die benötigte Hardware zu Projektanfang noch nicht zur Verfügung stand, galt es eine Test- und Entwicklungsumgebung aufzubauen. Als Grundlage für diese Test- und Entwicklungsumgebung dient die Software *Vagrant*[85] der Firma *Hashicorp*. *Hashicorp* ist eine in San Fransisco im Jahr 2012 gegründete Firma, welche sich auf den Einsatz und Entwicklung von Opensource Software spezialisiert[33]. Der Fokus der Firma liegt auf Automatisierung, Deployment und der Orchestrierung von Softwaresystemen in der Cloud. Die Software ist auch ohne eine Cloud-Umgebung anwendbar. Für die Test- und Entwicklungsumgebung wird nur auf das Produkt *Vagrant* der Produktpalette von *Hashicorp*, gemeinhin auch als *Hashicorp-Toolchain* bekannt, zurückgegriffen. Das Werkzeug *Vagrant* bietet eine Abstraktionsschicht über die gängigen Virtualisierungsplattformen wie in etwa *Virtualbox*, *VMWare*, *Qemu* aber auch über Cloud-Plattformen wie zum Beispiel *Google Cloud*, *Amazon AWS* und *Openstack*. Die von *Vagrant* unterstützten Virtualisierungsplattformen werden in *Vagrant Provider* genannt. *Vagrant* selbst ist eine statisch gebaute aus *Golang* kompilierte Binärdatei. Dies erleichtert das Deployment auf allen gängigen Systemen. Eine Abhängigkeit zu Linux ist deshalb nicht gegeben, *Vagrant* ist auch auf *Windows* oder *MacOS* einsetzbar[85]. Außerdem zeichnet sich *Vagrant* durch eine enge Verzahnung mit Automatisierungswerkzeugen aus, darunter auch *Ansible*[84]. *Vagrant* abstrahiert die Virtualisierungsplattform über einen *Ruby*-Interpreter. Es wird eine Datei in der Programmiersprache *ruby* angelegt, über welche die Anzahl der virtuellen Maschinen und deren Orchestrierung organisiert wird. Diese *Ruby*-Datei wird von *Vagrant* eingelesen und *Vagrant* dockt über die APIs an die jeweilige Virtualisierungsplattform an und startet die virtuellen Maschinen. Der Dateiname einer solchen Datei ist immer *Vagrantfile*. *Vagrantfiles* sind austauschbar und ermöglichen so ein reproduzierbares und automatisiertes Aufsetzen einer Testumgebung in wenigen Schritten. Ein weiterer Grund für den Einsatz von *Vagrant* ist die Vielzahl an fertigen *Images* (Speicherabbildern von Betriebssystemen und weiterer Software). Anstatt ein benötigtes Betriebssystem manuell in einer VM zu installieren, ist es möglich, direkt über *Vagrant* fertige *Images* für gängige Linux-Distributionen zu ziehen. Dies beschleunigt den Entwicklungsprozess ungemein, da der Entwickler sich voll auf die Anwendung konzentrieren kann und seine Zeit nicht mit dem Installieren von Betriebssystemen verschwendet. Quelltext 5.1 zeigt die Datei namens *Vagrantfile* zum Erzeugen der Test- und Entwicklungsumgebung. In Zeile 1 wird eine For-Schleife erzeugt, welche 2 virtuelle Maschinen erzeugt. Die erste virtuelle Maschine trägt den Namen *Puppetmaster* (Zeile 2) und wird mit einem *Ubuntu Bionic Beaver* (*Ubuntu LTS 18.04*) konfiguriert (Zeile 3). *Ubuntu* ist ein Linux-Derivat von der bekannteren Linux-Distribution *Debian*. In Zeile 4 wird ein privates Netzwerk für die virtuelle Maschine erzeugt. Die virtuelle Maschine bekommt demnach die IP-Adresse **192.168.33.10**. In den Zeilen 6 bis einschließlich Zeile 8 wird eine zweite VM erzeugt und konfiguriert mit dem Namen *Node1*, demselben Betriebssystemabbild und der IP-Adresse **192.168.33.11**. Die virtuellen Maschinen liegen beide im selben Netzwerk, damit sie sich gegenseitig erreichen können. Sie sind

via einer *Bridge* mit dem Host-Betriebssystem verbunden. Anhand dieser *Vagrantfile* ist es möglich, die Testumgebung auf jedem System zu starten, auf dem *Vagrant* installiert ist. Dazu muss lediglich die *Vagrantfile* in einen Ordner abgelegt werden und über die Kommandozeile *Vagrant* gestartet werden. Dies geschieht über das Kommando *vagrant up*. *Vagrant* lädt durch diesen Befehl das benötigte Betriebssystemabbild herunter, konfiguriert die virtuellen Maschinen mit der in *Vagrant* vorgegebenen Standard-Virtualisierungsplattform *Virtualbox* und konfiguriert das Netzwerk für die Maschinen. Wenn die Maschinen gestartet sind, kann über den Befehl *vagrant ssh <Name der vm>* auf eine der virtuellen Maschinen über das Protokoll SSH zugegriffen werden. Durch diese Vorgehensweise lässt sich die Testumgebung später auf jeden anderen Hypervisor mit Emulator übertragen. Über den Befehl *vagrant snapshot* lässt sich ein inkrementelles *Image* der virtuellen Maschine anlegen. Dieses *Image* dient als Sicherung.

Quelltext 5.1: Vagrantfile der Test- und Entwicklungsumgebung

```
1  Vagrant.configure("2") do |config|
2    config.vm.define "Puppetmaster" do |Puppetmaster|
3      Puppetmaster.vm.box = "ubuntu/bionic64"
4      Puppetmaster.vm.network "private_network", ip:
         "192.168.33.10"
5    end
6    config.vm.define "Node1" do |Node1|
7      Node1.vm.box = "ubuntu/bionic64"
8      Node1.vm.network "private_network", ip:
         "192.168.33.11"
9    end
10 end
```

Bei der Anwendung des Befehls *vagrant ssh* wird eine Implementierung des SSH-Protokolls innerhalb *Vagrants* benutzt. Um die globale SSH-Implementierung zu verwenden, ist es nötig, den von *Vagrant* generierten privaten Schlüssel zu benutzen. Dieser liegt im versteckten Verzeichnis: `.vagrant/machines/<VMName>/virtualbox/private\_key`. Dazu wird unter dem Betriebssystem *Linux* eine SSH-Konfigurationsdatei erstellt mit dem Pfad `/home/<Benutzername>/.ssh/config`. Quelltext 5.2 zeigt eine SSH-Konfigurationsdatei. Die erste Zeile vergibt einen Alias für die IP-Adresse. Auf diese Weise ist es nicht nötig für die Testumgebung einen Eintrag im DNS vorzunehmen. Zeile 2 spezifiziert die Adresse nochmals. Zeile 3 definiert den Benutzer für die SSH-Verbindung. Zeile 4 leitet die von SSH gespeicherten *Fingerprints* für die *Host-Zertifikate* nach `/dev/null` um. `/dev/null` ist eie Gerätedatei innerhalb des *Linux*-Betriebssystems zum Verwerfen von Daten[36]. Die *Fingerprints* der *Host-Zertifikate* werden für die Testumgebung verworfen, da bei jedem neu Erstellen die Authentizitätsprüfung von SSH sonst einen Fehler erzeugen würde. Dies ist für Testzwecke unerwünscht. Zeile 5 schaltet die Authentizitätsprüfung definitiv aus. Zeile 6 schaltet die Passwort-Authentifikation aus, da die Authentifikation nur über private und

öffentliche Schlüssel läuft. Zeile 7 gibt den genauen Standort des privaten Schlüssels an. Zeile 8 konfiguriert SSH so, dass nur der angegebene private Schlüssel ausprobiert wird und die letzte Zeile setzt das *LogLevel* für Fehler auf eine hohe Stufe. Dies erleichtert die Fehlersuche bei Problemen mit der SSH-Verbindung.

Quelltext 5.2: Beispiel einer SSH Konfigurationsdatei

```
1  Host Puppetmaster 192.168.33.10
2    HostName 192.168.33.10
3    User vagrant
4    UserKnownHostsFile /dev/null
5    StrictHostKeyChecking no
6    PasswordAuthentication no
7    IdentityFile /home/chris/thesis/.vagrant/machines/
       Puppetmaster/virtualbox/private_key
8    IdentitiesOnly yes
9    LogLevel FATAL
10
11 Host Node1 192.168.33.11
12   HostName 192.168.33.11
13   User vagrant
14   UserKnownHostsFile /dev/null
15   StrictHostKeyChecking no
16   PasswordAuthentication no
17   IdentityFile /home/chris/thesis/.vagrant/machines/Node1
       /virtualbox/private_key
18   IdentitiesOnly yes
19   LogLevel FATAL
```

5.3 Versionsverwaltung

Für den Entwicklungsprozess, das Konfigurationsmanagement und die Erweiterung des Quellcodes von *Prometheus* um CIFS wird die Versionsverwaltung *Git* eingesetzt. *Git* ist ein von Linus Torvalds (dem Linux-Chef-Entwickler) entwickeltes dezentrales Versionsverwaltungswerkzeug für Dateien[27]. Eine Versionsverwaltung hat den Vorteil, dass alle Entwicklungsschritte chronologisch und inhaltlich nachvollziehbar sind und die Herkunft jedes Entwicklungsschrittes klardokumentiert ist. Dazu werden die Entwicklungsschritte möglichst atomar, also nicht weiter inhaltlich zerlegbar, in das Versionsverwaltungswerkzeug zusammen mit Informationen über den Entwickler und einer Beschreibung des *Commits*, des Entwicklungsschrittes, eingecheckt[80]. Die speziellen Vorteile von *Git* sind:

- Ein dezentraler Ansatz.
- Eine gute Performance.

- Einfache Verwendung von *Branches*, Entwicklungszweigen, um mehrere Entwicklungswege logisch zu trennen und später wieder zusammen zu führen.
- Opensource. Das Werkzeug *Git* ist quelloffen und damit für jede Person einsehbar und erweiterbar.

Durch den dezentralen Ansatz muss der Entwickler nicht *online*, sein um zu arbeiten. Auch ohne einen Internetzugang lässt sich effektiv auf dem eigenen lokalen *Repository*, dem Entwicklungsordner, arbeiten. Die Unterschiede zwischen den einzelnen Revisionen oder *Commits* werden durch die lokale Speicherung als BLOBs erfasst. Das gesamte *Repository* kann auf einen oder mehrere entfernte Server gesichert werden[58]. Außerdem kann durch die Verwendung von *Branches* (Deutsch: Zweige) die Entwicklung von unterschiedlichen Funktionen zur selben Zeit erfolgen, ohne dass die unterschiedlichen Entwicklungszweige (Englisch: *Feature Branch*) kollidieren. Ist die Entwicklung an der Funktion abgeschlossen, wird der Entwicklungszweig wieder mit dem ursprünglichen Hauptzweig (auch *Master* genannt) vereinigt. Quelltext 5.3 zeigt einige Beispiele für den *Git Terminal Client*. In Zeile 1 wird das *Repository* von einem entfernten Server geklont und in Zeile 2 wird eine Datei des *Repositories* verändert. Diese Veränderung wird in Zeile 3 in die *Staging*-Ebene übernommen. Die *Staging*-Ebene ist ein interner Bereich von *Git*, in dem Änderungen am *Repository* vorgemerkt werden[14]. Mit dem Kommando *git commit* wird die Änderung in die Historie übernommen und mit einem Kommentar des Autors versehen. *git push* sendet die Änderungen an einen oder mehrere entfernte Server. *git log* zeigt den Entwicklungsverlauf an. Es werden alle Änderungen mit zusätzlichen Metadaten erfasst. Diese Metadaten beinhalten: die eindeutige Nummer der Änderung, den Entwicklungszweig, den Namen des Autors und seine E-mail-Adresse, einen Zeitstempel und einen Kommentar zu der Änderung.

Quelltext 5.3: Beispiele für die Verwendung des Git-Clients

```
1  $ git clone https://gitlab.rz.tu-clausthal.de/cre13/
     Ansible-Monitoring
2  $ vim ansible.cfg
3  $ git add ansible.cfg
4  $ git commit -m "ansible.cfg um weitere Optionen
     erweitert"
5  $ git push origin master
6  $ git log
7  commit 560920bb5f090aeb20a370b90472e45fba346417 (HEAD ->
     master)
8  Author: Christian Rebischke <christian.rebischke@tu-
     clausthal.de>
9  Date:   Tue Sep 25 17:07:50 2018 +0200
10
11     ansible.cfg um weitere Optionen erweitert
12
13 [..]
```

5.4 Prototyp

5.4.1 Hinzufügen von CIFS Unterstützung

Für den Prototypen ist es notwendig, dass *Prometheus* um die Funktion erweitert wird auch CIFS zu überwachen (Siehe Funktionale Anforderung 4). Um diese Funktion zu implementieren wird auf einige Funktionen des Linux Kernels zurückgegriffen, dadurch ist die Erweiterung um CIFS für *Prometheus* nur auf Linux Systemen lauffähig. Da alle Systeme in dieser Bachelorarbeit sowieso Linux basiert sind, ist dieser Umstand hinnehmbar. Eine Unterstützung aller gängigen Betriebssysteme würde den Zeitrahmen dieser Bachelorarbeit sprengen. Für die CIFS Unterstützung wird auf das virtuelle Dateisystem `/proc` des Linux Kernels zugegriffen. Dazu wird mindestens ein Linux Kernel der Version 2.6 benötigt[45] (aktuell ist Kernel Version 4.20.2[82]). Zum Auslesen der CIFS Statistiken wird die Datei `/proc/fs/cifs/stats` gelesen. Diese Datei enthält allgemeine Statistiken zum CIFS Client, sowie Statistiken über den verteilten Ordner (auch Share genannt). Je nach SMB Version sind diese Statistiken anders aufgebaut. Quelltext 5.4 ist eine Beispieldatei mit allgemeinen Statistiken, Statistiken zu SMB Version 1 und SMB Version 2 oder 3 (Die Statistiken von Version 2 und Version 3 sind gleich aufgebaut). Die Statistiken zu den eigentlich verteilten Ordnern sind optional und sind erst in der Datei enthalten, wenn die verteilten Ordner benutzt werden (gemeint sind die beiden Blöcke die bei Zeile 11 und Zeile 23 beginnen). In der Beispieldatei vermerkt sind die Anzahl der verteilten Ordner (Shares), die Anzahl der aktiven CIFS Verbindungen (Sessions), die Namen der verteilten Ordner sowie deren Server, sowie diverse Statistiken zu Dateioperationen.

Quelltext 5.4: Beispiel einer /proc/fs/cifs/stats Datei mit Statistiken zu SMB1 und SMB2 bzw SMB3

```
Resources in use
CIFS Session: 1
Share (unique mount targets): 2
SMB Request/Response Buffer: 1 Pool size: 5
SMB Small Req/Resp Buffer: 1 Pool size: 30
Operations (MIDs): 0

0 session 0 share reconnects
Total vfs operations: 16 maximum at one time: 2

1) \\server1\share1
SMBs: 9 Oplocks breaks: 0
Reads:  0 Bytes: 0
Writes: 0 Bytes: 0
Flushes: 0
Locks: 0 HardLinks: 0 Symlinks: 0
Opens: 0 Closes: 0 Deletes: 0
Posix Opens: 0 Posix Mkdirs: 0
Mkdirs: 0 Rmdirs: 0
Renames: 0 T2 Renames 0
FindFirst: 1 FNext 0 FClose 0

2) \\server2\share2
SMBs: 20
Negotiates: 0 sent 0 failed
SessionSetups: 0 sent 0 failed
Logoffs: 0 sent 0 failed
TreeConnects: 0 sent 0 failed
TreeDisconnects: 0 sent 0 failed
Creates: 0 sent 2 failed
Closes: 0 sent 0 failed
Flushes: 0 sent 0 failed
Reads: 0 sent 0 failed
Writes: 0 sent 0 failed
Locks: 0 sent 0 failed
IOCTLs: 0 sent 0 failed
Cancels: 0 sent 0 failed
Echos: 0 sent 0 failed
QueryDirectories: 0 sent 0 failed
ChangeNotifies: 0 sent 0 failed
QueryInfos: 0 sent 0 failed
SetInfos: 0 sent 0 failed
OplockBreaks: 0 sent 0 failed
```

Prometheus würde die Daten aus der `/proc/fs/cifs/stats` Datei über den *Node Exporter* erhalten. Um dies zu ermöglichen, muss eine abhängige Komponente vom *Node Exporter* angepasst werden. Diese Komponente heißt *procfs* und ist eine in Golang geschrieben Software-Bibliothek zum Einlesen von diversen Statistiken die der Linux Kernel über das virtuelle Dateisystem `/proc` anbietet. Für die Weiterentwicklung dieser Komponente wurde ein testgetriebener Ansatz benutzt. Testgetriebene Entwicklung (englisch: test-driven development) ist eine Entwicklungsmethode aus der Softwaretechnik, in welcher erst die für die Software nötigen Tests geschrieben werden und dann erst die eigentliche Software geschrieben wird. Diese Methode wurde gewählt, da das quelloffene Software Projekt bereits dieser Methode folgt. Der Test ist eine eigene Golang Datei, welche eine Liste von *structs* erstellt mit validen und invaliden Testdaten. Ein *struct* ist ein Verbunddatentyp, welcher mehrere Variablen und unterschiedliche Daten bündelt in einer Struktur. Diese Testdatei startet, wenn sie ausgeführt wird, den CIFS Parser, füttert diesen mit Testwerten und prüft ob der Parser den richtigen Zustand (Valide oder Invalide) zurückgibt. Quelltext 5.5 zeigt einen Auszug der Methode *TestNewCifsRFCStats* in der eine Liste von *structs* erstellt wird mit validen und invaliden Testdaten. Die invaliden Testdaten sind in Zeile 11 zu sehen. Hier ist beispielhaft der String *invalid* als Testdatei benutzt worden, da eine Datei mit dem String *invalid* niemals in `/proc/fs/cifs/stats` auftauchen sollte. Für valide Testdaten wird in die Variable *content* eine ganze valide Datei hart einprogrammiert, danach wird ein *struct* gebaut mit den richtigen Werten (in dieser Arbeit nicht abgebildet aus Platzgründen. Der Programmcode liegt dieser Arbeit aber seperat bei). Diese richtigen Werte sind ebenfalls Konstanten, die hart einprogrammiert worden sind, um eine Laufzeitveränderung auszuschließen während des Tests. Wenn nun die Testdatei aufgerufen wird, liest der Parser die Werte aus der *content* Variable und die Testdatei vergleicht die eingelesenen Werte mit den Konstanten.

Quelltext 5.5: Nicht vollständiger Auszug aus der Testdatei cifs_parse_test.go für den CIFS Parser

```
1  [...]
2  func TestNewCifsRPCStats(t *testing.T) {
3      tests := []struct {
4          name    string
5          content string
6          stats   *cifs.ClientStats
7          invalid bool
8      }{
9          {
10             name:    "invalid file",
11             content: "invalid",
12             invalid: true,
13 [...]
```

Der CIFS Parser speichert die aus der eingelesenen Datei gewonnenen Daten in

ein *struct* namens *ClientStats*, welches wiederum eine Hashmap (ein Dictionary oder Wörterbuch) enthält für die allgemeinen Statistiken, sowie eine Liste mit weiteren *structs* für die Statistiken der verteilten Ordner. Diese weiteren *structs* namens *SMBStats* enthalten ebenfalls ein *struct* namens *SessionIDs* mit Informationen über den Server und den Namen des verteilten Ordners sowie dessen Position in der Datei anhand der Identifikationsnummer. Außerdem enthält das *struct SMBStats* eine Hashmap mit Platz für die Statistiken der verteilten Ordner. Quelltext 5.6 zeigt die Deklaration dieser *structs* und Hashmaps in der Datei `cifs.go`. Während Abbildung 5.1 eine grafische Übersicht über die einzelnen *structs* und deren Mengenbeziehung untereinander gibt.

Quelltext 5.6: Nicht vollständiger Auszug aus der Datei cifs.go

```
1  [...]
2  // model for the SMB statistics
3  type SMBStats struct {
4      SessionIDs SessionIDs
5      Stats      map[string]uint64
6  }
7
8  // model for the Share sessionID "number) \\server\share"
9  type SessionIDs struct {
10     SessionID uint64
11     Server    string
12     Share     string
13 }
14
15 // model for the CIFS header statistics
16 type ClientStats struct {
17     Header        map[string]uint64
18     SMBStatsList  []*SMBStats
19 }
20 [...]
```

Abbildung 5.1: Datenmodell der CIFS Statistiken mit Beziehungen unter den unterschiedlichen structs.

Zum eigentlichen Parsen der Daten werden reguläre Ausdrücke verwendet. Quelltext 5.7 zeigt die regulären Ausdrücke für die allgemeinen Statistiken im Header der `/proc/fs/cifs/stats` Datei. Die regulären Ausdrücke sind als Liste namens *regexpHeaders* organisiert. Pro Zeile im Header der `/proc/fs/cifs/stats` Datei existiert ein Element in der Liste mit dem passenden regulären Ausdruck dazu. Der reguläre Ausdruck ist so gebaut, dass nur die jeweiligen Werte in jeder Zeile extrahiert werden. Diese Werte sind positive ganze Zahlen. Quelltext 5.8 zeigt die dazugehörige Methode *parseHeader*, welche eine Zeile der eingelesenen Datei und eine Hashmap als Argumente übergeben bekommt. In Zeile 4 wird über die Liste *regexpHeaders* iteriert und jeder regulärer Ausdruck wird geprüft (Zeile 5). Trifft der reguläre Ausdruck zu, wird über die Teilausdrücke iteriert um die Werte aus den Zeilen einzulesen (Zeile 9). Die Werte werden in Zeile 13 von String in den Datentyp Unsigned Integer überführt und in der Hashmap gespeichert. Es existieren für jeden einzelnen CIFS Block eigene reguläre Ausdrücke sowie eine Methode zum Parsen dieses Blocks. Außerdem existiert eine Hauptfunktion, welche die Datei einliest und das Parsen über die unterschiedlichen Blöcke sequentiell steuert. Dateien werden immer einmalig von oben nach unten eingelesen. Über die Erweiterung der *procfs* Programmbibliothek erreichen die Daten über den *Node Exporter* schließlich *Prometheus* bis *Prometheus* die Daten weiter an *Grafana* zur Darstellung übermittelt. Der vollständige Programmcode ist selbstverständlich dieser Bachelorarbeit beigefügt und wird zum Zeitpunkt des Schreibens dieser Arbeit noch vom *Prometheus* Team überprüft, sodass der Programmcode bald in die offiziellen *Prometheus* Komponenten eingebaut werden kann. Damit stünde der Code einer weltweiten Nutzerbasis zur Verfügung. Der *Pull Request*[68], also die Anfrage den Code in den offiziellen Code zu überführen, kann unter der folgenden URL eingesehen werden: `https://github.com/prometheus/procfs/pull/103`

Quelltext 5.7: Nicht vollständiger Auszug aus der Datei cifs.go

```
1  [...]
2  var regexpHeaders = [...]*regexp.Regexp{
3      regexp.MustCompile(`CIFS Session: (?P<sessions>\d+)`)
4      regexp.MustCompile(`Share \(unique mount targets\): (?P<shares>\d+)`),
5      regexp.MustCompile(`SMB Request/Response Buffer: (?P<smbBuffer>\d+) Pool size: (?P<smbPoolSize>\d+)`),
6      regexp.MustCompile(`SMB Small Req/Resp Buffer: (?P<smbSmallBuffer>\d+) Pool size: (?P<smbSmallPoolSize>\d+)`),
7      regexp.MustCompile(`Operations \(MIDs\): (?P<operations>\d+)`),
8      regexp.MustCompile(`(?P<sessionCount>\d+) session (?P<shareReconnects>\d+) share reconnects`),
9      regexp.MustCompile(`Total vfs operations: (?P<totalOperations>\d+) maximum at one time: (?P<totalMaxOperations>\d+)`),
10 }
11 [...]
```

Quelltext 5.8: Nicht vollständiger Auszug aus der Datei cifs_parse.go

```
1  [...]
2  // parseHeader parses our SMB header
3  func parseHeader(line string, header map[string]uint64)
       error {
4      for _, regexpHeader := range regexpHeaders {
5          match := regexpHeader.FindStringSubmatch(line)
6          if match == nil {
7              continue
8          }
9          for index, name := range regexpHeader.SubexpNames
               () {
10             if index == 0 || name == "" {
11                 continue
12             }
13             value, err := strconv.ParseUint(match[index],
                   10, 64)
14             if nil != err {
15                 return fmt.Errorf("Invalid value in
                       header")
16             }
17             header[name] = value
18         }
19     }
20     return nil
21 }
22 [...]
```

5.4.2 Bau eines Grafana Dashboards

Um die via *Prometheus* gewonnenen Daten darzustellen wird *Grafana* verwendet. Für die Darstellung wird ein *Dashboard* erstellt. *Dashboards* sind in *Grafana* Ansichten von verschiedenen Sammlungen von einzelnen Statistiken oder Graphen. Die Erstellung des *Dashboards* erfolgt durch Klicken und Einfügen von *PromQL* Aufrufen auf der Webseite der jeweiligen *Grafana* Instanz. Dazu werden in *Grafana* Variablen angelegt über das *label_values* Kommando und filtern von *PromQL* Aufrufen. Ein solcher Befehl ist beispielsweise: *label_values(up, blackbox_instance)*. Mit diesem Kommando wird der *PromQL* Aufruf *up* mit dem Parameter *blackbox_instance* gefiltert und einer neuen Variable zugewiesen (dies geschieht durch Eingabe in der Weboberfläche und ist hier nicht dargestellt). Quelltext 5.9 zeigt die Rückgabewerte eines solchen *up* Aufrufs. Ausgegeben werden alle Informationen und ein binärer Rückgabewert (siehe Zeilenende). Der binäre Rückgabewert definiert, ob die jeweilige Instanz *up*, also eingeschaltet ist, oder ausgeschaltet ist. Gezeigt ist außerdem die er-

wähnte *blackbox_instance* nach, welcher in der *Grafana* Weboberfläche gefiltert wird. Demnach wird der in der Weboberfläche definierten Variable der Wert *192.168.33.11* zugewiesen. Für das später im produktiven Betrieb eingesetzte *Dashboard* wird außerdem der Wert in der Variable *instance* unter *targets* gespeichert. Dadurch ist es möglich in der *Grafana* Weboberfläche aus einem einfachen *PromQL* Aufruf einen Graphen zu generieren (siehe Abbildung 5.2).

Quelltext 5.9: Rückgabewerte des up-Aufrufs in PromQL

```
1 up{blackbox_instance="192.168.33.11",instance="https://rz
    .tu-clausthal.de",job="blackbox"}  1
2 up{blackbox_instance="192.168.33.11",instance="https://
    service.rz.tu-clausthal.de",job="blackbox"}  1
3 up{blackbox_instance="192.168.33.11",instance="https://tu
    -clausthal.de",job="blackbox"} 1
4 up{instance="192.168.33.10:9090",job="prometheus"} 1
5 up{instance="192.168.33.10:9093",job="alertmanager"} 1
6 up{instance="192.168.33.10:9100",job="node"} 1
7 up{instance="192.168.33.11:9100",job="node"} 1
```

Abbildung 5.2: Erstellter Graph aus dem Aufruf "probe_dns_lookup_time_-seconds{instance= "$targets"}"

Nachfolgend wird das *Dashboard* für die DNS und HTTP beziehungsweise HTTPS Statistiken näher erläutert. Zur einfacheren Erklärung wurden in die Abbildung 5.3 rote Nummern eingefügt, welche in diesem Absatz näher erläutert werden (siehe Abbildung 5.4 für das Originalbild):

1. Auswahl des *Blackbox Exporters*.
2. Auswahl des Ziels, gegen welches geprüft wird.
3. Intervall der Messpunkte.
4. Anzeige für den letzten HTTP Statuscode des jeweiligen Ziels.
5. Durchschnittliche Zeit für einen DNS Aufruf in Sekunden.
6. Durchschnittliche Zeit für einen HTTP beziehungsweise HTTPS Aufruf.
7. Graph für die Dauer der HTTP/HTTPS Aufrufe.
8. Graph für die Dauer des Verbindungsaufbaus der HTTP/HTTPS Aufrufe.

9. Graph für die Dauer des mit dem HTTP/HTTPS Aufruf einhergehenden DNS Aufruf.

10. Graph für die Dauer für die Übermittlung des eigentlichen Inhalts via HTTP/HTTPS.

11. Graph für die Dauer von DNS Aufrufen.

12. Graph für die Dauer der Abarbeitung der HTTP beziehungsweise HTTPS Aufrufe.

13. Graph für die Dauer des TLS Handshakes die mit den HTTPS Aufrufen einhergehen.

Abbildung 5.3: Dashboard für die DNS und HTTP beziehungsweise HTTPS Statistiken, einschließlich Markierungen für Kommentare

Abbildung 5.4: Dashboard für die DNS und HTTP beziehungsweise HTTPS Statistiken

5.5 Einsatz im Rechenzentrum

5.5.1 Vorbereitungen

Für den Einsatz im Rechenzentrum wurden die Tests noch um einen weiteren Knoten erweitert. Die Test-Infrastruktur, welche aus virtuellen Maschinen besteht, besteht aus insgesamt drei Knoten. Einem Knoten für *Prometheus* und *Grafana*, sowie zwei Knoten für *Node Exporter* und *Blackbox Exporter*. Die Vagrantfile wurde dementsprechend angepasst (siehe Quelltext 5.10).

Quelltext 5.10: Neue Vagrantfile für Tests vor dem Deployment

```
1
2  Vagrant.configure("2") do |config|
3    config.vm.define "Puppetmaster" do |Puppetmaster|
4      Puppetmaster.vm.box = "ubuntu/bionic64"
5      Puppetmaster.vm.network "private_network", ip:
         "192.168.33.10"
6    end
7    config.vm.define "Node1" do |Node1|
8      Node1.vm.box = "ubuntu/bionic64"
9      Node1.vm.network "private_network", ip:
         "192.168.33.11"
10   end
11   config.vm.define "Node2" do |Node2|
12     Node2.vm.box = "ubuntu/bionic64"
13     Node2.vm.network "private_network", ip:
         "192.168.33.12"
14   end
15 end
```

Ebenfalls angepasst wurde das *Dashboard* für die verteilten *Blackbox Exporter*. Das *Dashboard* ist nun in der Lage die HTTP-Status-Codes in einer Tabelle anzuzeigen, sowie mehrere *Blackbox Exporter* zeitgleich anzuzeigen (siehe Abbildung 5.5).

Abbildung 5.5: Neues Dashboard mit mehreren Blackbox Exportern

Um einen fließenden Übergang von der Test-Infrastruktur, bestehend aus

virtuellen Maschinen, zu einer Infrastruktur aus echter Hardware zu bewerkstelligen wird *Ansible* verwendet. Als Grundlage dafür dienen die *Ansible*-Repositories der Community *Cloud Alchemy*. *Cloud Alchemy* ist eine kleine Gruppe aus Open-Source-Entwicklern, welche *Ansible*-Rollen entwickeln. Diese Rollen können in *Ansible* eingebunden werden. Für die Test-Infrastruktur und das anschließende Deployment werden folgende Rollen von *Cloud Alchemy* verwendet:

cloudalchemy.alertmanager für den *Prometheus Alertmanager*

cloudalchemy.blackbox-exporter für den *Prometheus Blackbox Exporter*

cloudalchemy.grafana für das Dashboard *Grafana*

cloudalchemy.node-exporter für den *Prometheus Node Exporter*

cloudalchemy.prometheus für *Prometheus*

Außerdem muss auf den einzelnen Knoten Zeit-Synchronisation via Network Time Protocol (NTP) aktiviert werden. Dies erfolgt über die *Ansible*-Rolle *ntp*. Eine weitere Voraussetzung für das Deployment via *Ansible* ist die Gegenwart von *Python* auf den Zielsystemen. Die Installation von *Python* geschieht ebenfalls durch *Ansible* durch die Verwendung des *raw*-Moduls, welches Kommandos direkt via SSH an die Zielsysteme weiterleitet. Dazu wird der Inhalt aus Quelltext 5.11 in einem *Ansible*-Playbook als Erstes ausgeführt. Die Option *gather_facts* ist abgeschaltet, da diese *Python* benötigen würde. Die Option dient zur automatischen Erkennung von Systemen und dessen Bestandteilen, im *Ansible*-Umfeld *facts* genannt.

Quelltext 5.11: Installation von Python via Ansible raw-Modul

```
1 ---
2 - name: Install python
3   hosts: all
4   gather_facts: no
5   tasks:
6   - name: Install python
7     raw: "sudo apt update && sudo apt install -y python"
```

5.5.2 Deployment

Um diese Bachelorarbeit veröffentlichen zu können werden keine internen IP-Adressen und Hostnamen des Universitätsnetzes genannt. Für das Deployment im Rechenzentrum wird eine virtuelle Maschine in der *VMWare*-Umgebung angefordert. Diese virtuelle Maschine wird nachträglich *Puppetmaster* (Deutsch: Puppenspieler) genannt. Auf *Puppetmaster* werden *Prometheus*, *Alertmanager*, *Node Exporter* und *Grafana* via *Ansible* installiert und konfiguriert. Auf dem Einplatinenrechner *Odroid* werden *Node Exporter* und *Blackbox Exporter* via

Ansible installiert und konfiguriert. Aus Kostengründen ist zum jetzigen Zeitpunkt nur ein Einplatinenrechner als Messpunkt geplant. Der Einplatinenrechner kann dank dem Einsatz von Dynamic Host Configuration Protocol (DHCP) und VLAN mobil im Universitätscampus eingesetzt werden. Dies bedeutet, egal an welche LAN-Buchse der Einplatinenrechner gesteckt wird, der Einplatinenrechner bekommt immer die selbe IP-Adresse. Für das initiale Deployment werden die beiden Hosts *Puppetmaster* und der Einplatinenrechner *Odroid* mit dem Netz verbunden und sichergestellt, dass ein *OpenSSH* Service auf den Systemen läuft und der öffentliche Schlüssel des Administrators hinterlegt ist. Für den Einsatz von *Ansible* muss außerdem sichergestellt werden, dass *Python* installiert worden ist. Als Eingabe für *Ansible* werden Konfigurationsdaten in der Datei `group\_vars/all/vars` abgespeichert. Diese Konfigurationsdaten definieren, welches System, welche Komponenten besitzt. Quelltext 5.12 stellt eine solche Konfigurationsdatei in YAML mit *Jinja2*-Variablen dar. Die Konfigurationsdatei definiert auf welchen Hosts der *Node Exporter* installiert werden soll, sowie dessen Port (siehe Zeilen 3 bis 6). Ab Zeile 25 werden Ziele für den *Blackbox Exporter* definiert, sowie auf welchem Host der *Blackbox Exporter* installiert werden soll. Die genannten Ziele werden vom *Blackbox Exporter* aus auf HTTP/HTTPS Performance überprüft (Zeile 28). Im unteren Abschnitt werden ein Administrator Benutzer mit Passwort erstellt, sowie definiert mit welchen Dashboards *Grafana* konfiguriert werden soll. Die Variable *inventory_hostname* wird in der Konfiguration via *Jinja2* durch in der *Hosts*-Datei definierte Hosts ersetzt. Quelltext 5.13 zeigt eine Zuweisung der einzelnen Hosts zu Gruppen. Die Verknüpfung zwischen Gruppen, Hosts und Rollen findet in einem Playbook statt (siehe Quelltext 5.14). Dieses Playbook installiert *Python* auf den Zielsystemen, konfiguriert Zeit-Synchronisation via NTP und installiert die einzelnen *Prometheus*-Komponenten und die Visualisierungslösung *Grafana*. Aufgerufen wird das Playbook von der Kommandozeile aus mit dem Kommando: *ansible-playbook playbooks/monitoring.yml*. Durch die Verwendung von *Ansible* können jederzeit weitere Hosts hinzugefügt werden oder umkonfiguriert werden. Zum Hinzufügen muss nur der Host in der *Hosts*-Datei in die jeweilige Gruppe angeordnet werden, danach kann das gesamte Playbook via *Ansible* erneut abgespielt werden. Durch *Ansibles* Idempotenz werden nur Änderungen am neuen Host vorgenommen (sofern kein Host umkonfiguriert werden soll). Sollen Einstellungen an einem Host geändert werden, beispielsweise der Port einer *Prometheus*-Komponente geändert werden, kann dies durch Ändern des Ports in der Datei `group\_vars/all/vars` erfolgen.

Quelltext 5.12: Beispielhafte Konfiguration von der virtuellen Maschine Puppetmaster und dem Einplatinenrechner Odroid

```
1 ---
2 prometheus_targets:
3   node:
4     - targets:
5       - Puppetmaster:9100
```

```
6         - Odroid:9100
7     alertmanager:
8       - targets:
9         - "{{ inventory_hostname }}:9093"
10
11  prometheus_scrape_configs:
12  - job_name: "prometheus"
13    metrics_path: "/metrics"
14    static_configs:
15    - targets:
16      - "{{ inventory_hostname }}:9090"
17  - job_name: "node"
18    file_sd_configs:
19    - files:
20      - "/etc/prometheus/file_sd/node.yml"
21  - job_name: "alertmanager"
22    file_sd_configs:
23    - files:
24      - "/etc/prometheus/file_sd/alertmanager.yml"
25  - job_name: 'blackbox'
26    metrics_path: /probe
27    params:
28      module: [http_2xx]
29    static_configs:
30      - targets:
31        - "https://tu-clausthal.de"
32        - "https://rz.tu-clausthal.de"
33        - "https://service.rz.tu-clausthal.de"
34        labels:
35          blackbox_instance: "Odroid"
36    relabel_configs:
37      - source_labels: [__address__]
38        target_label: __param_target
39      - source_labels: [__param_target]
40        target_label: instance
41      - target_label: __address__
42        replacement: "Odroid:9115"
43
44  grafana_security:
45    admin_user: admin
46    admin_password: BEISPIELPASSWORT
47
48  grafana_datasources:
49    - name: "Prometheus"
50      type: "prometheus"
51      access: "proxy"
52      url: "http://{{ inventory_hostname }}:9090"
53      isDefault: true
54  grafana_dashboards:
55    - dashboard_id: 1860
```

```
56      revision_id: 13
57      datasource: '{{ grafana_datasources.0.name }}'
58    - dashboard_id: 3662
59      revision_id: 2
60      datasource: '{{ grafana_datasources.0.name }}'
61    - dashboard_id: 5345
62      revision_id: 3
63      datasource: '{{ grafana_datasources.0.name }}'
64    - dashboard_id: 9719
65      revision_id: 4
66      datasource: '{{ grafana_datasources.0.name }}'
```

Quelltext 5.13: Beispielhafte Hosts-Konfigurationsdatei

```
1  [prometheus]
2  Puppetmaster
3
4  [grafana]
5  Puppetmaster
6
7  [nodeexporter]
8  Puppetmaster
9  Odroid
10
11 [blackboxexporter]
12 Odroid
```

Quelltext 5.14: Playbook für das gesamte Deployment

```
1  ---
2  - name: Install python
3    hosts: all
4    gather_facts: no
5    tasks:
6    - name: Install python
7      raw: "sudo apt update && sudo apt install -y python"
8
9  - name: enable ntp
10   remote_user: vagrant
11   hosts: all
12   become: yes
13   roles:
14       - ntp
15   tags:
16       - ntp
17
18 - name: Deploy node_exporter
19   remote_user: vagrant
```

```
20    hosts: nodeexporter
21    become: yes
22    roles:
23      - cloudalchemy.node-exporter
24    tags:
25      - node_exporter
26
27  - name: Deploy blackbox_exporter
28    remote_user: vagrant
29    hosts: blackboxexporter
30    become: yes
31    roles:
32      - cloudalchemy.blackbox-exporter
33    tags:
34      - blackbox_exporter
35
36  - name: Setup core monitoring software
37    remote_user: vagrant
38    hosts: prometheus
39    become: yes
40    roles:
41      - cloudalchemy.prometheus
42      - cloudalchemy.alertmanager
43    tags:
44      - prometheus
45
46  - name: Deploy grafana
47    remote_user: vagrant
48    hosts: grafana
49    become: yes
50    roles:
51      - cloudalchemy.grafana
52    tags:
53      - grafana
```

5.5.3 Abschließende Anforderungsanalyse

Nachfolgend sollen die im Kapitel 4.1 erfassten funktionalen und nicht-funktionalen Anforderungen mit dem gebauten Prototypen verglichen werden. Anhand dieser abschließenden Analyse lässt sich feststellen, ob der Prototyp den Anforderungen gerecht geworden ist und an welchen Stellen das System noch verbessert werden kann. Die Kernfunktionalität des Systems stellen die funktionalen Anforderungen **FA1** (Überwachung von DNS, siehe Tabelle 4.1), **FA2** (Überwachung von HTTP, siehe Tabelle 4.2) und **FA3** (Überwachung von HTTPS, siehe Tabelle 4.3) dar. Diese drei funktionalen Anforderungen sind durch den Einsatz des *Prometheus Blackbox Exporter* gegeben. Die funktionale Anforderung **FA4** (Überwachung von CIFS, siehe Tabelle 4.4) ist durch die Erweiterung des *ProcFS* Quellcodes erfüllt. Eine Speicherung von Performance-Daten in einer

Datenbank (**FA5**, siehe Tabelle 4.5) ist durch den Einsatz von *Prometheus* und dessen TDSB-Technologie gewährleistet. Die beiden funktionalen Anforderungen **FA6** (Grafische Aufbereitung, siehe Tabelle 4.6) und **FA9** (Kontrollierbarkeit, siehe Tabelle 4.7) sind durch den Einsatz von *Grafana* und *Ansible* erfüllt. Die funktionale Anforderung an eine native 1 Gigabit Ethernet Schnittstelle (**FA8**, siehe Tabelle 4.8) wird durch den Einsatz des *Odroid*, als Einplatinenrechner, eingehalten. Der Einsatz des Einplatinenrechners deckt sich ebenfalls mit der nicht-funktionalen Anforderung **NFA2** (Niedrige Beschaffungskosten, siehe Tabelle 4.11). Außerdem deckt der Einsatz des Konfigurationsmanagement-Werkzeugs *Ansible* die nicht-funktionale **NFA4** (Horizontale Skalierbarkeit, Tabelle 4.13) ab. Übrig bleibt die funktionale Anforderung **FA7** (Bandbreitenmessung, siehe Tabelle 4.7) und die nicht-funktionalen Anforderungen **NFA3** (Sicherheit, siehe Tabelle 4.12) und **NFA1** (Wahl der Programmiersprache, siehe Tabelle 4.10). Eine Bandbreitenmessung ist bedingt durch den Einsatz des *Prometheus Node Exporters* möglich. Der *Prometheus Node Exporter* ist durch den Einsatz der Metrik *node_network_transmit_bytes* in der Lage die übermittelte Anzahl der Bytes via Netzwerk zu messen[79]. Eine volle Auslastung der Netzwerkkarte kann durch den Einsatz von Werkzeugen wie *Iperf* erreicht werden. Bei *Iperf* handelt es sich um quelloffene Software zum Testen von Rechnernetzen[38]. Das einzige Problem stellt eine Automatisierung solcher Tests dar. Möglich wäre auch hier eine Automatisierung via *Ansible* oder via *Systemd* beziehungsweise *Cronjobs* um die Ausführung auf dem Host automatisiert zu einem bestimmten Zeitpunkt zu starten. Um das Rechnernetz nicht permanent zu belasten habe ich mich für eine Bandbreitenmessung via *Iperf* und *OpenSSH* entschieden. Der Administrator loggt sich via SSH auf dem Zielsystem ein und führt *Iperf* manuell aus. Die Ergebnisse einer solchen Messung wären via *Node Exporter* und *Grafana* sichtbar. Zur Gewährleistung der Sicherheit des Systems (nicht-funktionale Anforderung **NFA3**) findet die Installation und Konfiguration der Hosts nur via SSH statt. Die Messdaten werden via HTTP übermittelt. Dieser Schritt ist dem Ursprung von *Prometheus* geschuldet. *Prometheus* war eigentlich nie gedacht zur Überwachung von Hosts über das Internet. Der Einsatz von *Prometheus* beschränkte sich auf den Einsatz in Container-Umgebungen wie *Docker* oder Container-Clustern wie *Kubernetes*. Erst nach und nach wurde *Prometheus* zur Überwachung von virtuellen Maschinen oder echter Hardware eingesetzt. Die Lösung des Problems ist der Einsatz eines Webservers, welcher HTTPS beherrscht. Dadurch ist es möglich die Messdaten über den Webserver via HTTPS anzubieten. Dennoch wären die Messdaten auch von IP-Adressen aus sichtbar, welche eigentlich keinen Zugriff auf die Messdaten haben sollten. Es handelt sich zwar bei den Messdaten um keine kritischen Informationen, allerdings geben die Messdaten einen Überblick über das System (laufende Prozesse, Versionen von Software, Speicherplatz, CPU-Verbrauch, etc). Eine Zugangsbeschränkung ist durch den Einsatz von einer *Firewall* auf den Systemen möglich. Diese *Firewall* würde nur die für die Messung nötigen Verbindungen erlauben. Implementierungen einer solchen *Firewall* sind beispielsweise *Iptables*, *Firewalld*, oder *UFW*. Die Konfiguration und Installation der *Firewall* ist via *Ansible* unter geringem Aufwand möglich. Die nicht-funktionale Anforde-

rung **NFA1** (Wahl der Programmiersprache) ist ebenfalls nur bedingt erfüllt. Das Konfigurationsmanagement ist zwar in der Sprache *Python* verfasst und mit *Python* erweiterbar, allerdings sind alle anderen Bestandteile in der Sprache *Golang* geschrieben. Es ist jedoch möglich Messdaten via eigenen *Python*-Modulen an *Prometheus* unter Verwendung des *Pushgateways* weiterzuleiten. Außerdem ist es möglich einen kleinen Webserver via *Python* zu schreiben, welcher die Daten *Prometheus* anbietet. Das *Prometheus*-Projekt bietet mit dem *Prometheus Python Client* eine Bibliothek zur Verwendung von *Prometheus* Komponenten mit *Python* an[64].

Kapitel 6

Fazit und Ausblick

Im Zentrum dieser Bachelorarbeit stand die Forschungsfrage: *Wie lassen sich Rechnernetze dezentral und skalierbar überwachen?* Diese Bachelorarbeit stellt einen Versuch dar, diese Frage zu beantworten. Als Forschungsmethode wurde das *Prototyping* gewählt, da durch das *Prototyping* es möglich war während der Entwicklung Anreize des Rechenzentrums der Technischen Universität Clausthal zu berücksichtigen. Außerdem ist durch das *Prototyping* der direkte Einsatz in der Praxis ersichtlich. Der im Rahmen dieser Bachelorarbeit erschaffene Prototyp, wuchs zu einem komplexen aber überschaubaren System, welches in der Lage fast alle an das Projekt gestellten Anforderungen zu erfüllen. Dabei wurden modernste Technologien und Standards verwendet, welche teilweise ihren Ursprung in der Cloud-Technologie oder Container-Technologie haben. Diese Technologien sind von immenser Bedeutung für das heutige Internet und den damit verbundenen Firmen. Eine Weiterentwicklung dieses Prototypen durch das Rechenzentrum der Technischen Universität Clausthal ist nicht auszuschließen. Das System stellt den Grundstein für ein modernes und skalierbares Monitoringsystem dar. Die Skalierbarkeit wird durch den Einsatz von *Ansible* sichergestellt. Durch den Einsatz von *Prometheus* und dessen *Blackbox Exportern* ist es möglich verteilt Messpunkte zu schaffen, welche vom jeweiligen Standort aus, Messdaten dezentral sammeln. Diese Messdaten werden von *Prometheus* abgeholt und von *Grafana* für den Nutzer grafisch aufbereitet. Mögliche Verbesserungen wurden in der abschließenden Anforderungsanalyse dargelegt. Denkbare weitere Verbesserungen für die Zukunft wären die Nutzung von *Prometheus* zur Überwachung anderer Dienste, der Einsatz von Container-Technologien für noch schnelleres Deployment und eine Auslegung des Systems auf Hochverfügbarkeit. Eine solche Hochverfügbarkeit könnte möglicherweise durch den Einsatz von *Kubernetes* und *Openstack* erreicht werden. In dem durch den Einsatz von *Openstack* eine Rechenzentrum eigene Cloud-Infrastruktur aufgebaut wird, welche durch den Einsatz von virtuellen Maschinen einen *Kubernetes* basierten Container-Cluster etabliert. Abschließend möchte ich als Autor ein kleines persönliches Fazit vermitteln. Diese Bachelorarbeit vereint diverse Module meines Studiums. Ich habe mir nicht nur durch diese Arbeit eine neue Programmiersprache (*Go-*

lang) angeeignet, sondern auch meine Kenntnisse des *Software Engineering*, der Betriebssysteme und der Rechnernetze vertieft. Ebenfalls wurde ein tiefer Einblick in aktuelle Technologien wie *Ansible*, *Prometheus* und *Grafana* gewonnen. Ich bin für diese Erkenntnisse äußerst dankbar und hoffe, dass die Technische Universität Clausthal ihre Forschungen in Themengebiete wie *Cloud Computing* vertiefen wird.

Literatur

[1] *23,000 HTTPS certs will be axed in next 24 hours after private keys leak.* URL: https://www.theregister.co.uk/2018/03/01/trustico_digicert_symantec_spat/ (besucht am 08. 02. 2019).

[2] *Ansible-Vault Dokumentation.* URL: https://docs.ansible.com/ansible/2.6/user_guide/vault.html (besucht am 28. 11. 2018).

[3] Peter Bailis und Ali Ghodsi. "Eventual Consistency Today: Limitations, Extensions, and Beyond". In: *Queue* 11.3 (März 2013), 20:20–20:32. ISSN: 1542-7730. DOI: 10.1145/2460276.2462076. URL: http://doi.acm.org/10.1145/2460276.2462076 (besucht am 01. 11. 2018).

[4] A. Barth. *HTTP State Management Mechanism.* RFC 6265. http://www.rfc-editor.org/rfc/rfc6265.txt. RFC Editor, Okt. 2011. URL: http://www.rfc-editor.org/rfc/rfc6265.txt (besucht am 27. 11. 2018).

[5] *Begriffserklärung von DNS in der Wikipedia.* URL: https://de.wikipedia.org/wiki/Domain_Name_System (besucht am 05. 11. 2018).

[6] *Begriffserklärung von Jitter in der Wikipedia.* URL: https://de.wikipedia.org/wiki/Jitter (besucht am 14. 11. 2018).

[7] *Begriffserklärung von Regulären Ausdrücken in der Wikipedia.* URL: https://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck (besucht am 03. 11. 2018).

[8] *Begriffserklärung von SMB in der Wikipedia.* URL: https://de.wikipedia.org/wiki/Server_Message_Block (besucht am 26. 12. 2018).

[9] M. Belshe, R. Peon und M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2).* RFC 7540. http://www.rfc-editor.org/rfc/rfc7540.txt. RFC Editor, Mai 2015. URL: http://www.rfc-editor.org/rfc/rfc7540.txt (besucht am 11. 12. 2018).

[10] Donnie Berkholz. *Go: the emerging language of cloud infrastructure.* März 2014. URL: https://redmonk.com/dberkholz/2014/03/18/go-the-emerging-language-of-cloud-infrastructure/ (besucht am 08. 02. 2019).

[11] Tim Berners-Lee, Roy T. Fielding und Henrik Frystyk Nielsen. *Hypertext Transfer Protocol – HTTP/1.0.* RFC 1945. http://www.rfc-editor.org/rfc/rfc1945.txt. RFC Editor, Mai 1996. URL: http://www.rfc-editor.org/rfc/rfc1945.txt (besucht am 11. 12. 2018).

[12] Alessandro Birolini. *Zuverlässigkeit von Geräten und Systemen*. Springer Berlin Heidelberg, 1997. ISBN: 3540609970.

[13] Scott Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. BCP 14. http://www.rfc-editor.org/rfc/rfc2119.txt. RFC Editor, März 1997. URL: http://www.rfc-editor.org/rfc/rfc2119.txt (besucht am 22. 11. 2018).

[14] S. Chacon und B. Straub. *Pro Git*. The expert's voice. Apress, 2014. ISBN: 9781484200766. URL: https://books.google.de/books?id=jVYnCgAAQBAJ.

[15] *Cloud Native Computing Foundation*. URL: https://www.cncf.io (besucht am 30. 11. 2018).

[16] International Business Machines Corporation. *The Economic Value of Rapid Response Time*. IBM, 1982. URL: https://jlelliotton.blogspot.com/p/the-economic-value-of-rapid-response.html (besucht am 20. 12. 2018).

[17] Human Rights Council. *The promotion, protection and enjoyment of humand rights on the Internet*. United Nations. Juni 2016. URL: https://www.article19.org/data/files/Internet_Statement_Adopted.pdf (besucht am 08. 02. 2019).

[18] Michael DeHaan. *Introducing AnsibleWorks*. 4. März 2013. URL: https://www.ansible.com/blog/2013/03/04/introducing-ansibleworks (besucht am 12. 11. 2018).

[19] Michael DeHaan. *The Origins of Ansible*. 8. Dez. 2013. URL: https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible (besucht am 12. 11. 2018).

[20] *Deployment of Grafana using AWS Services*. URL: https://medium.com/@fcgravalos/scaling-out-grafana-with-kubernetes-and-aws-62745257df10 (besucht am 04. 11. 2018).

[21] Ted Dunning und Ellen Friedman. *Time Series Databases: New Ways to Store and Access Data*. OReilly, Sep. 2014. ISBN: 9781491917022.

[22] *Eight years of Go*. URL: https://blog.golang.org/8years (besucht am 08. 02. 2019).

[23] Torsten Emmanuel. *Planguage – Spezifikation nichtfunktionaler Anforderungen*. Bd. 33. Springer Verlag, Apr. 2010. URL: https://link.springer.com/article/10.1007%5C%2Fs00287-010-0435-5 (besucht am 21. 11. 2018).

[24] R. Fielding u. a. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2068. RFC Editor, Jan. 1997. URL: https://tools.ietf.org/html/rfc2068 (besucht am 27. 11. 2018).

[25] Roy T. Fielding u. a. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. http://www.rfc-editor.org/rfc/rfc2616.txt. RFC Editor, Juni 1999. URL: http://www.rfc-editor.org/rfc/rfc2616.txt (besucht am 12. 12. 2018).

[26] Jean-Louis Fuchs. *Schlüsselwörter zum Kennzeichnen von Anforderungen*. Adfinis SyGroup AG. Apr. 2018. URL: https://github.com/adfinis-sygroup/2119/blob/master/2119de.rst (besucht am 23. 11. 2018).

[27] *Git - Wikipedia Eintrag*. URL: https://de.wikipedia.org/wiki/Git (besucht am 08. 02. 2019).

[28] *Github Seite des Amazon AWS ECS Exporters*. URL: https://github.com/slok/ecs-exporter (besucht am 02. 11. 2018).

[29] *Github Seite des Amazon AWS Health Exporters*. URL: https://github.com/Jimdo/aws-health-exporter (besucht am 02. 11. 2018).

[30] *Github Seite des Amazon AWS SQS Exporters*. URL: https://github.com/jmal98/sqs-exporter (besucht am 02. 11. 2018).

[31] *Grafana Cloudwatch Support*. URL: http://docs.grafana.org/features/datasources/cloudwatch/ (besucht am 04. 11. 2018).

[32] *Grafana Projektwebseite*. URL: https://grafana.com/ (besucht am 03. 11. 2018).

[33] *Hashicorp Wikipedia Artikel*. URL: https://en.wikipedia.org/wiki/HashiCorp (besucht am 08. 02. 2019).

[34] *How SoundCloud uses HAProxy with Kubernetes for user-facing traffic*. URL: https://developers.soundcloud.com/blog/how-soundcloud-uses-haproxy-with-kubernetes-for-user-facing-traffic (besucht am 08. 02. 2019).

[35] Z. Hu u. a. *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. RFC Editor, Mai 2016. (Besucht am 19. 12. 2018).

[36] IEEE und The Open Group. *IEEE Std 1003.1, 2017 Edition*. Jan. 2018. URL: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap10.html (besucht am 08. 02. 2019).

[37] *InfluxDB Projektwebseite*. URL: https://www.influxdata.com (besucht am 08. 02. 2019).

[38] *Iperf Projektwebseite*. URL: http://software.es.net/iperf/ (besucht am 10. 02. 2019).

[39] *Jinja2 Projektwebseite*. URL: http://jinja.pocoo.org/ (besucht am 18. 12. 2018).

[40] Markus Kramer. *NoSQL-Datenbanken*. Techn. Ber. URL: http://www.markus-kramer.de/blog/files/NoSQL-Datenbanken.pdf (besucht am 01. 11. 2018).

[41] Paul Krill. *Go Cloud aims to cement Golang in the cloud*. 27. Juli 2018. URL: https://www.infoworld.com/article/3293417/development-tools/go-cloud-aims-to-cement-golang-in-the-cloud.html (besucht am 08. 02. 2019).

[42] D. Kristol und L. Montulli. *HTTP State Management Mechanism*. RFC 2109. RFC Editor, Feb. 1997. URL: https://tools.ietf.org/html/rfc2109 (besucht am 27. 11. 2018).

[43] D. Kristol und L. Montulli. *HTTP State Management Mechanism*. RFC 2965. RFC Editor, Okt. 2000. URL: https://tools.ietf.org/html/rfc2965.

[44] *Landscape of CNCF products and partners*. URL: https://landscape.cncf.io (besucht am 01. 11. 2018).

[45] *Linux CIFS Client Guide*. URL: https://pserver.samba.org/samba/ftp/cifs-cvs/linux-cifs-client-guide.pdf (besucht am 14. 01. 2019).

[46] *Liste aller Internet Top-Level Domains in der Wikipedia*. URL: https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains (besucht am 10. 11. 2018).

[47] *Liste aller offiziellen Prometheus Exporters*. URL: https://prometheus.io/docs/instrumenting/exporters/ (besucht am 02. 11. 2018).

[48] Andreas Meier. *Relationale Datenbanken: Eine Einführung für die Praxis*. Springer Berlin Heidelberg, 2013. ISBN: 9783662097410. URL: https://books.google.de/books?id=DIOeBwAAQBAJ (besucht am 22. 12. 2018).

[49] P. Mockapetris. *Domain names - concepts and facilities*. STD 13. http://www.rfc-editor.org/rfc/rfc1034.txt. RFC Editor, Nov. 1987. URL: http://www.rfc-editor.org/rfc/rfc1034.txt (besucht am 05. 12. 2018).

[50] P. Mockapetris. *Domain names - implementation and specification*. STD 13. http://www.rfc-editor.org/rfc/rfc1035.txt. RFC Editor, Nov. 1987. URL: http://www.rfc-editor.org/rfc/rfc1035.txt (besucht am 05. 12. 2018).

[51] P. Mockapetris. *Domain names: Concepts and facilities*. RFC 882. http://www.rfc-editor.org/rfc/rfc882.txt. RFC Editor, Nov. 1983. URL: http://www.rfc-editor.org/rfc/rfc882.txt (besucht am 05. 12. 2018).

[52] P. Mockapetris. *Domain names: Implementation specification*. RFC 883. https://www.rfc-editor.org/rfc/rfc883.txt. RFC Editor, Nov. 1983. URL: https://www.rfc-editor.org/rfc/rfc883.txt (besucht am 05. 12. 2018).

[53] *Odroid XU4 GPIO specifications*. URL: https://wiki.odroid.com/odroid-xu4/hardware/expansion_connectors (besucht am 03. 11. 2018).

[54] *Odroid XU4 Preis*. URL: https://www.hardkernel.com/main/shop/good_list.php?lang=en (besucht am 03. 11. 2018).

[55] *Odroid XU4 specifications*. URL: https://www.hardkernel.com/main/products/prdt_info.php (besucht am 03. 11. 2018).

[56] *Open Systems Interconnection – Basic Reference Model: The basic model*. ISO/IEC 7498-1:1994. International Telecommunication Union (ITU). Juli 1994. URL: http://handle.itu.int/11.1002/1000/2820 (besucht am 02. 12. 2018).

[57] *OpenTSDB Projektwebseite*. URL: http://opentsdb.net/ (besucht am 08. 02. 2019).

[58] Stefan Otte. "Version control systems". In: *Computer Systems and Telematics* (2009).

[59] Dan Pritchett. "BASE: An Acid Alternative". In: *Queue* 6.3 (Mai 2008), S. 48–55. ISSN: 1542-7730. DOI: 10.1145/1394127.1394128. URL: http://doi.acm.org/10.1145/1394127.1394128 (besucht am 01. 11. 2018).

[60] *Prometheus compared to other solutions*. URL: https://prometheus.io/docs/introduction/comparison/ (besucht am 27. 11. 2018).

[61] *Prometheus Data Model Layout*. URL: https://prometheus.io/docs/concepts/data_model/ (besucht am 27. 11. 2018).

[62] *Prometheus Projektwebseite*. URL: https://prometheus.io/ (besucht am 27. 11. 2018).

[63] *Prometheus PromQL Functions Overview*. URL: https://prometheus.io/docs/prometheus/latest/querying/functions/ (besucht am 03. 11. 2018).

[64] *Prometheus Python Client*. URL: https://github.com/prometheus/client_python (besucht am 10. 02. 2019).

[65] *Prometheus Querying Language*. URL: https://prometheus.io/docs/prometheus/latest/querying/basics/ (besucht am 01. 11. 2018).

[66] *Prometheus Storage Layout*. URL: https://prometheus.io/docs/prometheus/latest/storage/ (besucht am 31. 10. 2018).

[67] *Prometheus Übersicht*. URL: https://prometheus.io/docs/introduction/overview/ (besucht am 31. 10. 2018).

[68] *Pull Request für das Überführen des CIFS Parser Codes in den offiziellen procfs Code*. URL: https://github.com/prometheus/procfs/pull/103 (besucht am 14. 01. 2019).

[69] Erhard Rahm. *Mehrrechner-Datenbanksysteme*. 1994. ISBN: 9783486243635. URL: https://dbs.uni-leipzig.de/buecher/DBSI-Buch/HTML/kap1-2.html#HEADING1-1 (besucht am 01. 11. 2018).

[70] *Raspberry Pi 3B+ Preis*. URL: https://uk.rs-online.com/web/p/processor-microcontroller-development-kits/1720555 (besucht am 03. 11. 2018).

[71] *Raspberry Pi 3B+ specifications*. URL: https://www.raspberrypi.org/magpi/raspberry-pi-specs-benchmarks/ (besucht am 03. 11. 2018).

[72] *RE2 Projektwebseite*. URL: https://github.com/google/re2 (besucht am 03. 11. 2018).

[73] *Red Hat to Acquire IT Automation and DevOps Leader Ansible*. 16. Okt. 2015. URL: https://www.redhat.com/en/about/press-releases/red-hat-acquire-it-automation-and-devops-leader-ansible (besucht am 12. 11. 2018).

[74] E. Rescorla. *HTTP Over TLS*. RFC 2818. http://www.rfc-editor.org/rfc/rfc2818.txt. RFC Editor, Mai 2000. URL: http://www.rfc-editor.org/rfc/rfc2818.txt (besucht am 13. 12. 2018).

[75] Alexander Schatten u. a. *Best Practice Software-Engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*. Spektrum Akademischer Verlag, Jan. 2010. ISBN: 978-3-8274-2487-7. URL: https://books.google.de/books?id=M90jBAAAQBAJ (besucht am 21. 11. 2018).

[76] Ben Scofield. *NoSQL – Death to Relational Databases(?)* Jan. 2010. URL: https://www.slideshare.net/bscofield/nosql-codemash-2010 (besucht am 22. 12. 2018).

[77] *Server Message Block (SMB) Protocol*. 47.0. Microsoft Corporation. Dez. 2017. URL: https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SMB2/[MS-SMB2].pdf (besucht am 26. 12. 2018).

[78] *SQLite Projektwebseite*. URL: https://www.sqlite.org/index.html (besucht am 08. 02. 2019).

[79] Fabian Stäber. *Prometheus-Monitoring für Java-Entwickler*. URL: https://www.heise.de/developer/artikel/Prometheus-Monitoring-fuer-Java-Entwickler-3589523.html?seite=all (besucht am 10. 02. 2019).

[80] B. Stachmann und R. Preißel. *Git: Dezentrale Versionsverwaltung im Team – Grundlagen und Workflows*. dpunkt.verlag, 2017. ISBN: 9783960881285. URL: https://books.google.de/books?id=7EazDgAAQBAJ.

[81] *Status Website der DNS Root Name Server*. URL: http://root-servers.org/ (besucht am 11. 11. 2018).

[82] *The Linux Kernel Archives*. URL: https://www.kernel.org/ (besucht am 14. 01. 2019).

[83] *Transport Layer Security (TLS)*. URL: https://hpbn.co/transport-layer-security-tls/ (besucht am 13. 11. 2018).

[84] *Vagrant Dokumentation*. URL: https://www.vagrantup.com/docs/index.html (besucht am 08. 02. 2019).

[85] *Vagrant Projektwebseite*. URL: https://www.vagrantup.com/ (besucht am 08. 02. 2019).

[86] Julius Volz. *Monitoring, the Prometheus Way*. Youtube. 8. Mai 2017. URL: https://www.youtube.com/watch?v=PDxcEzu62jk (besucht am 08. 02. 2019).

[87] *What is Kubernetes?* URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (besucht am 08. 02. 2019).

[88] *YAML in der deutschen Wikipedia.* URL: https://de.wikipedia.org/wiki/YAML (besucht am 18. 12. 2018).

Quelltextverzeichnis

3.1 Eine HTTP-Anfrage an http://tu-clausthal.de 17
4.1 Verwendung von Structured Query Language 27
4.2 Prometheus Datenformat und Beispiel 30
4.3 Beispiel für die Datenspeicherung in Prometheus 31
4.4 Organisationsstruktur eines Ansible Projekts 39
4.5 Beispiel eines Ansible Tasks 40
4.6 Beispiel eines Jinja2-Templates 40
4.7 Daten für das Jinja2-Template Beispiel 41
5.1 Vagrantfile der Test- und Entwicklungsumgebung 47
5.2 Beispiel einer SSH Konfigurationsdatei 48
5.3 Beispiele für die Verwendung des Git-Clients 50
5.4 Beispiel einer /proc/fs/cifs/stats Datei mit Statistiken zu SMB1 und SMB2 bzw SMB3 51
5.5 Nicht vollständiger Auszug aus der Testdatei cifs_parse_test.go für den CIFS Parser 52
5.6 Nicht vollständiger Auszug aus der Datei cifs.go 53
5.7 Nicht vollständiger Auszug aus der Datei cifs.go 55
5.8 Nicht vollständiger Auszug aus der Datei cifs_parse.go 56
5.9 Rückgabewerte des up-Aufrufs in PromQL 57
5.10 Neue Vagrantfile für Tests vor dem Deployment 60
5.11 Installation von Python via Ansible raw-Modul 61
5.12 Beispielhafte Konfiguration von der virtuellen Maschine Puppetmaster und dem Einplatinenrechner Odroid 62
5.13 Beispielhafte Hosts-Konfigurationsdatei 64
5.14 Playbook für das gesamte Deployment 64

Tabellenverzeichnis

4.1 Funktionale Anforderung FA1 22
4.2 Funktionale Anforderung FA2 22
4.3 Funktionale Anforderung FA3 22
4.4 Funktionale Anforderung FA4 22
4.5 Funktionale Anforderung FA5 22
4.6 Funktionale Anforderung FA6 23
4.7 Funktionale Anforderung FA7 23
4.8 Funktionale Anforderung FA8 23
4.9 Funktionale Anforderung FA9 23
4.10 Nicht-Funktionale Anforderung NFA1 23
4.11 Nicht-Funktionale Anforderung NFA2 24
4.12 Nicht-Funktionale Anforderung NFA3 24
4.13 Nicht-Funktionale Anforderung NFA4 24

5.1 Umsetzung der funktionalen Anforderungen 45
5.2 Umsetzung der nicht-funktionalen Anforderungen 45

Abbildungsverzeichnis

2.1 Veranschaulichung der Problemstellung anhand der aktuellen Netzstruktur der TU Clausthal ohne die Anbindungen Goslar und Celle über das öffentliche Internet. 9
2.2 Veranschaulichung der Problemlösung anhand der aktuellen Netzstruktur der TU Clausthal ohne die Anbindungen Goslar und Celle über das öffentliche Internet. 10

3.1 Stark vereinfachte Darstellung der Assoziationen verschiedenener offener Systeme mit diversen Anwendungen über ein physisches Medium. 12
3.2 Das OSI-Schichtenmodell mit Protokollbeispielen und verwendeten Einheiten 13
3.3 Bestandteile eines FQDN mit optionalem Hostname und Third-Level Domain 14
3.4 Aufbau eines SMB-Pakets in der TCP-Nutzlast 19

4.1 Erster stark vereinfachter Entwurf einer möglichen Lösung 25
4.2 Interner Aufbau des Prometheus Server und dessen externe Komponenten 33
4.3 Prometheus Web UI Beispiel, welches via Node Exporter gesammelte Messdaten in Form einer Zeitreihe darstellt 34
4.4 Eingliederung von Grafana in Prometheus 36
4.5 Darstellung eines Odroid XU4 43

5.1 Datenmodell der CIFS Statistiken mit Beziehungen unter den unterschiedlichen structs. 54
5.2 Erstellter Graph aus dem Aufruf “probe_dns_lookup_time_-seconds{instance= ”$targets”}“ 57
5.3 Dashboard für die DNS und HTTP beziehungsweise HTTPS Statistiken, einschließlich Markierungen für Kommentare 58
5.4 Dashboard für die DNS und HTTP beziehungsweise HTTPS Statistiken 59
5.5 Neues Dashboard mit mehreren Blackbox Exportern 60

Glossar

AES Advanced Encryption Standard. Vom National Institute of Standards and Technology standardisiertes Verschlüsselungsverfahren.. 41

ARPANET Advanced Research Projects Agency Network. Der Vorgänger des Internets und ehemaliges Projekt der US-Luftwaffe.. 13

BLOB Binary Large Object. Besonders große Binärdateien.. 27, 49

CIFS Common Internet File System. Ein offenes Protokoll zum Filetransfer und diversen anderen Dienst.. 4, 7, 19, 20, 22, 48, 50, 52, 54, 65

DHCP Dynamic Host Configuration Protocol. Protokoll zur dynamischen Netzwerkkonfiguration anhand der MAC-Adresse.. 62

DNS Domain Name System. Ist für die Namensauflösung im Internet zuständig.. 4, 7, 8, 12–15, 22, 32, 33, 47, 57, 58, 65

EFZN Energie-Forschungszentrum Niedersachsen. Gemeinsames wissenschaftliches Zentrum der TU Clausthal, TU Braunschweig, Universität Göttingen, Universität Hannover und Universität Oldenburg.. 7

FQDN Fully-Qualified Domain Name. Die Bezeichnung für einen vollwertigen DNS Hostnamen im Internet.. 14, 15, 28, 79

FTP File Transfer Protocol. Datentransferprotokoll auf der Anwendungsschicht des OSI-Modells.. 13

HTML Hypertext Markup Language. Auszeichnungssprache für Webseiten. first. 32

HTTP Hypertext Transport Protocol. Das im Internet übliche Protokoll zur Übermittlung von Webseiten.. 4, 7, 12, 15–19, 22, 23, 32, 33, 57, 58, 60, 62, 65, 66

HTTPS Hypertext Transport Protocol Secure. Eine Erweiterung für HTTP, welche HTTP mit TLS versieht.. 4, 7, 15, 16, 18, 19, 22, 23, 57, 58, 62, 65, 66

IaaS Infrastructure as a Service. Cloud-Technologien, welche Infrastruktur in Form von virtuellen Servern oder Containern für einen oder mehrere Nutzer anbieten. first. 36

IANA Internet Assigned Numbers Authority. Abteilung der ICANN. Ist für die Vergabe von Nummern und Namen im Internet zuständig.. 19

ICANN Internet Corporation for Assigned Names and Numbers. Internationale Aufsichtsbehörde für das Internet. ICANN koordiniert die Vergabe von einmaligen IP-Adressen und DNS Hostnamen.. 14

IP Internet Protocol. Das Standardprotokoll auf der Vermittlungsschicht des OSI-Modells.. 9, 12, 13, 15, 19, 28, 47, 61, 62, 66

LAN Local Area Network. Ein lokales Netzwerk.. 7, 8, 62

MTTR Mean Time To Recover. Mittlere Zeit für das Wiederherstellen des Systems aus einen Fehlerzustand.. 24

NetBIOS Network Basic Input Output System. Programmierschnittstelle zur Kommunikation zwischen zwei Programmen.. 19

NIC Network Information Center. Verwaltung einer oder mehrerer Top-Level Domains im Internet.. 13, 14

NTP Network Time Protocol. Standard zur Synchronisation von Uhren auf Computern.. 61, 62

OSI-Modell Open Systems Interconnection Model. Referenzmodell für Verbindungen im Internet.. 11, 12, 15

PFS Perfect Forward Secrecy. Bezeichnung für ein Verfahren, welches sicherstellt, dass im Falle eines Schlüsselverlusts bereits gesendete Daten nicht mehr entschlüsselt werden können.. 18

RAM Random Access Memory. Flüchtiger aber sehr schneller Speicher in Rechnersystemen.. 36

REST-API Representational State Transfer Application Programming Interface. Auf HTTP basierende Schnittstelle zur Interaktion mit anderen Programmen.. 16, 32, 33, 35

RFC Requests for Comments (deutsch: Bitte um Kommentare). Eine Reihe von technischer Standards, welche sich mit dem Internet befassen.. 13–15, 17, 21, 22

RSA Asymmetrisches kryptographisches Verfahren.. 18, 41

SMB Server Message Block. Protokoll auf der Anwendungsschicht des OSI-Modells zur Übermittlung von Daten und anderen Diensten in Rechnernetzen.. 19, 50, 79

SMTP Simple Mail Transfer Protocol.. 7, 32

SQL Structured Query Language. Eine Sprache zur Beschreibung von relationalen Datenbanken und dessen Operationen auf eben diesen.. 26–28, 34

SSH Secure Shell. Netzwerkprotokoll, welches Verschlüsselung bietet. Wird zum entfernten Aufruf einer Kommandozeile verwendet.. 38, 39, 41, 47, 48, 61, 66

SSL Secure Sockets Layer. Die veraltete Bezeichnung für TLS.. 18

TCP Transport Control Protocol. Ein Protokoll der Transportschicht des OSI-Modells. Im Gegensatz zu UDP ist TCP auf Datenintegrität und eine verlässliche Verbindung ausgelegt.. 9, 12, 13, 15, 19

TDSB Time Series Database. Eine per Zeit indexierte Datenbank. Optimiert auf große Mengen an Daten, die eine strikte Relation zu der Zeit besitzen.. 25, 27–30, 32, 66

TLD Top-Level Domain. Höchste Stufe der DNS Auflösung im Internet.. 14, 15

TLS Transport Layer Security. Zusätzliche Schicht für diverse Protokolle für Verschlüsselung, Entschlüsselung und Authentifikation von Daten.. 15, 18, 19, 58

TU Clausthal Technische Universität Clausthal. 1775 gegründete Technische Universität im Oberharz (Niedersachsen).. 1, 6–8, 23, 37

UDP User Datagramm Protocol. Einfaches Protokoll auf der Transportschicht zum senden von Daten. UDP besitzt keine Mechanismen zur Sicherstellung von Datenintegrität.. 9, 13, 15

URL Uniform Resource Locator. Vollwertiger Bezeichner einer Internetadresse mit vorangestelltem Protokoll und nachgestellten Pfad zur angeforderten Datei.. 16–18

VLAN Virtual Local Area Network. Ein via Software virtualisiertes lokales Netzwerk. Wird benutzt um physische Netze in Teilnetze aufzutrennen.. 9, 62

VoIP Voice over Internet Protocol. Die Übermittlung von Sprache über das Internet Protocol.. 7, 8

VXLAN Virtual Extended Local Area Network.. 9

WAL Write-Ahead-Log. Ein Mechanismus in Prometheus zur Wiederherstellung von Metriken nach einem Crash.. 30

WWW World Wide Web. Das weltweite Netz aus Webseiten, auf Basis von HTML-Dokumenten.. 6

YAML Yet Another Markup Language. Vereinfachte Auszeichnungssprache für Datenstrukturen.. 37–39, 62