

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

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

25. September 2018

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 Bachelor-Arbeit 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 angefertigt 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 25. September 2018

Christian Rebischke

Sperrvermerk

Foobar Foobar Foobar

Clausthal-Zellerfeld, der 25. September 2018

Christian Rebischke

Inhaltsverzeichnis

Vorwort	6
Problemstellung	7
Technische Grundlagen	10
Netzwerkgrundlagen	10
Domain Name System (DNS)	12
Hypertext Transport Protocol (HTTP)	14
Hypertext Transport Protocol Secure (HTTPS)	16
Common Internet File System (CIFS)	17
Herleitung eines Lösungsansatzes	19
Anforderungsanalyse	19
Funktionale Anforderungen	20
Nicht-funktionale Anforderungen	21
Systemarchitektur	22
Datenmodell	23
Relationale Datenbanken	23
NoSQL-Datenbanken	24
Prometheus	27
Komponenten	27
PromQL	30
Grafana	31
Konfigurationsmanagement	32
Ansible	32
Funktionsweise	33
Organisationsstruktur	33
Dateistruktur	34
Sicherheit	36
Wahl der Hardware	37
Projektumsetzung	39
Erfüllung der Anforderungen	39
Test- und Entwicklungssystem	40

Entwicklungsprozess	43
Versionsverwaltung	43
Entwicklungsmethoden	45
Prototyp	45
Betriebstests	45
Fazit	46

Vorwort

Es gibt nun mehr als 20 Jahren das Internet 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 Disziplinen spielt das Internet und die damit verknüpfte Informationstechnologie eine Rolle. Sei es die Industrie 4.0 mit ihren cyberphysischen 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[13] 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.

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 Netzeintritte 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)

Abbildung 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 Netzes 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 das 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 aus erreichbar sind. Da außerdem nur ein binärer 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 schnelle Verbindungen angewiesen. Eine zu hohe Latenz zwischen einem Client und dem Dienst führt unweigerlich zu für den Nutzer spürbaren Konsequenzen (zum Beispiel zu verzögerten Seitenaufrufen beim Web-Browsing). Noch mehr ins Gewicht fallen Latenzen bei VoIP, dort sind Latenzen oder ein Jitter (die Varianz der Laufzeit der Datenpakete[4]) leicht auszumachen. Was fehlt, ist ein Netz aus verteilten Messpunkten, das es ermöglicht, die Erreichbarkeit einzelner Dienste periodisch und über einen längeren Zeitraum zu beobachten (siehe Abbildung 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. Weiterhin könnten im Fall eines Ausfalls die zuständigen Netzadministratoren benachrichtigt werden, im Idealfall durch gewohnte Kommunikationswege wie Email. 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. Insgesamt lassen sich daraus folgende Herausforderungen an diese Arbeit ableiten:

- Es muss eine Hardware gefunden werden, welche sich als verteilter Messpunkt eignet.
- Es muss eine Monitoring-Plattform gebaut oder eine vorhandene Monitoring-Plattform erweitert werden, so dass sie den Anforderungen 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 Rechnernetze kommunizieren.

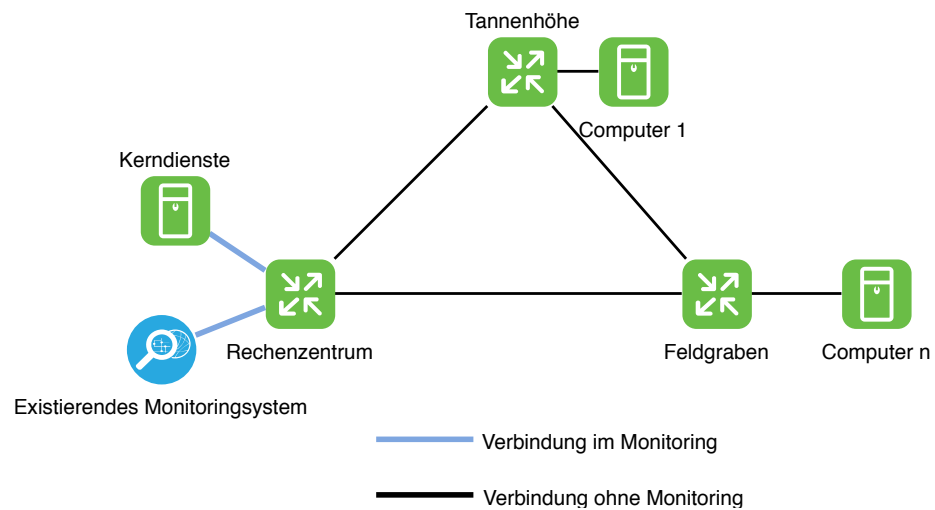


Abbildung 1: Veranschaulichung der Problemstellung

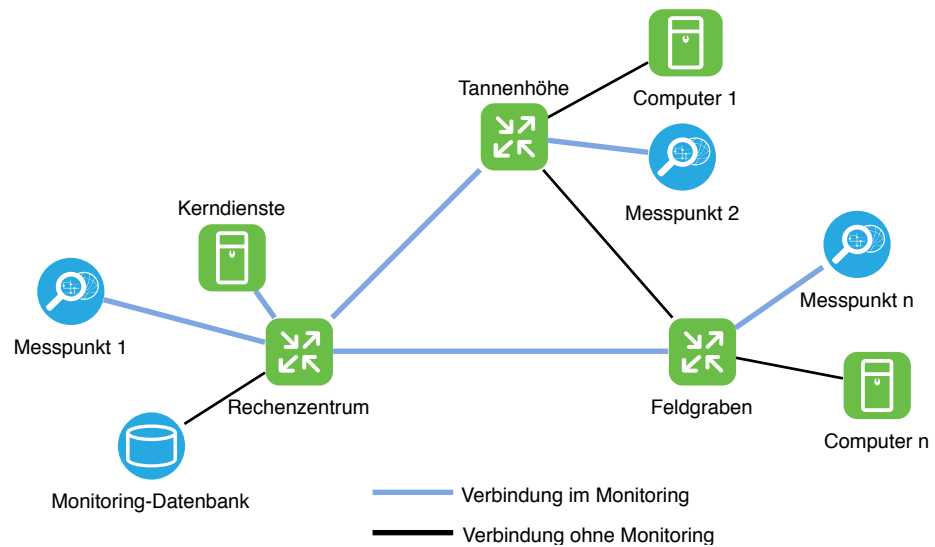


Abbildung 2: Veranschaulichung der Problemlösung

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. Darunter fallen einige wichtige Netzwerkprotokolle, sowie eine allgemeine Einführung in Computernetzwerke.

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[37, Siehe Abschnitt 4.1.2]. Diese *offenen Systemen* sind mit einem physischen Medium verbunden und bilden so ein Computernetzwerk. Abbildung 3 zeigt eine solche Verbindung mehrerer *offener Systeme*.

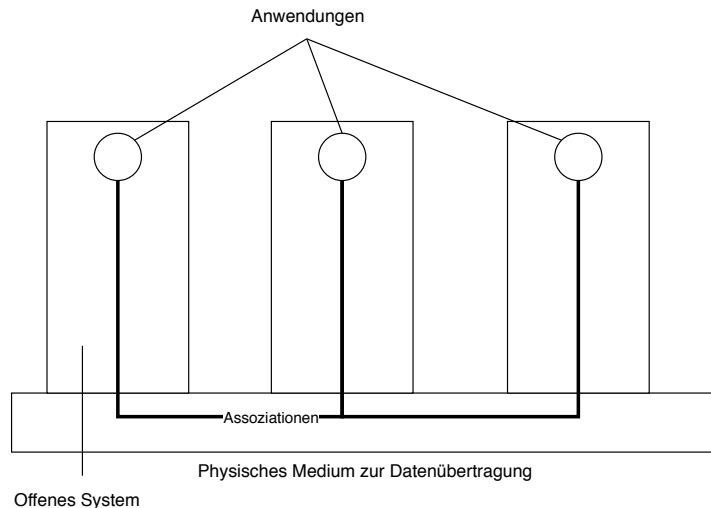


Abbildung 3: Veranschaulichung der Assoziationen verschiedener offener Systeme mit diversen Anwendungen über ein physisches Medium

Innerhalb eines dieser *offenen Systeme* sind sieben Netzwerkschichten oder auch *OSI-Schichten* definiert[37, Siehe Abschnitt 6.1.2]. Zur Kommunikation zwischen zwei *offenen Systemen* wird mindestens eine Schicht durchlaufen, in den meisten Fällen jedoch mehrere Schichten. Abbildung 4 listet alle sieben Schichten, deren Protokolle, Einheiten und Einordnung auf. Die Funktionsweise des OSI-Modell lässt sich am besten durch ein kurzes Beispiel erklären:

Es wird angenommen, dass mit einem Laptop eine Website aufgerufen wird. Nachfolgend wird nur der Netzverkehr zwischen dem Client und dem Server betrachtet. Wenn der Client eine HTTP-basierte Website aufrufen möchte, sendet dieser eine HTTP-Anfrage in Form von HTTP-Daten (Anwendungsschicht). Diese HTTP-Daten werden wiederum in ein Transport Control Protocol (TCP)-Segment eingebettet (Transportschicht), welches wiederum in ein Internet Protocol (IP)-Paket eingebettet ist (Vermittlungsschicht). Dieses IP-Paket enthält die Adresse des Empfängers und wird ebenfalls eingebettet in einen Ethernet-Frame (Sicherheitsschicht), welcher wiederum in Form von kodierten Bits über ein Netzkabel übertragen wird (Bitübertragungsschicht). Dieses Matrjoschka-ähnliche Gebilde wird über das physische Medium versendet und der Empfänger packt es anfangend 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

OSI-Schicht	Einordnung	Protokollbeispiele	Einheiten
Anwendungen	Ende-zu-Ende-Verbindungen über mehrere Hops	HTTP HTTPS SMTP XMPP DNS LDAP SSH NRPE NSCA	Daten
Darstellung			
Sitzung			
Transport			
Vermittlung			
Sicherung	Punkt-zu-Punkt-Verbindungen	Ethernet Token Ring FDDI MAC ARCNET	Frames
Bitübertragung			Bits

Abbildung 4: Das OSI-Schichtenmodell mit Protokollbeispielen und verwendeten Einheiten

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 20 Jahre alten Verzeichnisdienst, welcher über das gleichnamige Protokoll für Menschen merkbare Internetadressen auf IP-Adressen abbildet. DNS wurde erstmalig im Jahr 1983 in den beiden Request For Comments (RFC)s RFC 882[32] und RFC 883[33] 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[30]. 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 5 veranschaulicht die Gliederung dieses

FQDN in die einzelnen Bestandteile.

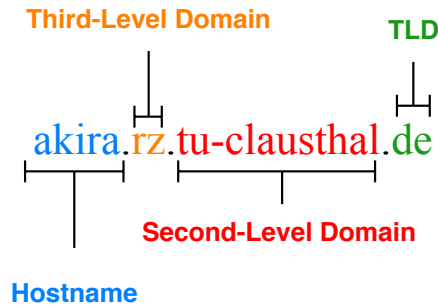


Abbildung 5: Bestandteile eines FQDN mit optionalem Hostname und Third-Level Domain

Eine Angabe des Hostname oder einer Third-Level Domain ist optional. Es können auch weitere Domains hinzugefügt werden. So ist auch eine Sixth-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[31, 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 sind nicht nur hierarchisch strukturiert, sondern unterliegen zusätzlich 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*[28]. Die Vergabe der TLDs verlief direkt über die Internet Corporation for Assigned Names and Numbers (ICANN). Der jeweilige Käufer ist Eigentümer der jeweiligen TLD. Ein Eigentümer 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*[30, siehe Section 2.4]. *Name Server* sind Server-Programme, welche Informationen zur Baumstruktur enthalten. Ergo halten *Name Server* Domain-IP-Tabellen für ihre hierarchische Ebene vor. Wenn kein Eintrag zu dem angefragten FQDN existiert, wird der nächsthöhere DNS-Server gefragt. An oberster Stelle stehen 13 *Root Nameserver* Server[56]. Die *Root Name Server* umfassen 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 User Datagram Protocol (UDP) an den Zielport 53[31, Siehe Section 4.2.1]. Es ist allerdings auch DNS über TCP spezifiziert[31, 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 DNS um Verschlüsselung erweitern[23] (DNS-over-HTTPS ist zum Zeitpunkt dieser Arbeit noch nicht endgültig spezifiziert). Dieses Bestreben, DNS sicherer zu machen, unterstreicht die Wichtigkeit des Protokolls. Das Protokoll ist heute maßgeblich für das Funktionieren des Internets. Ohne DNS wären diverse vernetzte Anwendungen nicht möglich. Der Webbrowser etwa löst permanent DNS-Anfragen aus. 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-Anfragen dauert der Aufbau einer neuen Seite 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[12]. Bezogen auf die Latenz von DNS-Anfragen hieße das, dass der Benutzer 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.

Hypertext Transport Protocol (HTTP)

Hypertext Transport Protocol (HTTP) ist ein auf TCP/IP aufbauendes Datenübertragungsprotokoll, welcher auf der Anwendungsschicht des OSI-Modell 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[8]. 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[7]. HTTP ist ein zustandsloses Protokoll, dementsprechend werden mehrere Anfragen getrennt voneinander und ohne Kontext zu einander bearbeitet. Es wird keine Session aufgebaut und keine Sitzungsinformationen verwaltet. Sitzungsinformationen können zusätzlich durch *Cookies* übertragen werden. *Cookies* ergänzen den HTTP-Header um Sitzungsinformationen, dadurch ist eine genaue Zuordnung möglich. Zur Interaktion mit dem Server besitzt HTTP mehrere Methoden. Die häufigsten sind:

GET für einfache Anfragen

POST um Informationen wie beispielsweise Logindaten an den Server zu senden.

HEAD ähnlich wie **GET** mit dem Unterschied, dass der Server bei einer Antwort keinen Inhalt mitliefert.

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 1 ist eine HTTP-Anfrage mit Antwort und Verbindungsaufbau an den Host `http://tu-clausthal.de` visualisiert. 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[18]. Ab Zeile 10 beginnt die Antwort des Servers. Dort werden Protokoll und die Version nochmal bestätigt und in diesem Beispiel ein Statuscode hinzugefügt (301 weist daraufhin, dass der Inhalt der Seite verschoben worden ist und an einem neuen Ort liegt (siehe Zeile 13)). Dazu wird auf einen neuen Uniform Resource Locator (URL) verwiesen. Die URL dient als Pfadangabe zur gewünschten Ressource oder Information. 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 1: Eine HTTP-Anfrage an http://tu-clausthal.de

```
1 * Rebuilt URL to: http://tu-clausthal.de/
2 * Trying 2001:638:605:20:1::2...
3 * TCP_NODELAY set
4 * Connected to tu-clausthal.de (2001:638:605:20:1::2)
   port 80 (#0)
5 > GET / HTTP/1.1
6 > Host: tu-clausthal.de
7 > User-Agent: curl/7.59.0
8 > Accept: */*
9 >
10 < HTTP/1.1 301 Moved Permanently
11 < Date: Fri, 11 May 2018 23:48:50 GMT
12 < Server: Apache/2.2.22 (Ubuntu)
13 < Location: http://www.tu-clausthal.de/
14 < Vary: Accept-Encoding
15 < Content-Length: 235
16 < Content-Type: text/html; charset=iso-8859-1
17 <
18 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
19 <html><head>
20 [...]
21 </body></html>
22 * Connection #0 to host tu-clausthal.de left intact
```

Bei der Verwendung von HTTP können folgende Fehlerfälle auftreten:

- Die Verbindung ist zu langsam und der Seitenaufbau erfolgt in einem für den Nutzer zu langsamen Tempo.
- Die Verbindung kommt nicht zu Stande, weil der Webserver nicht erreichbar ist.

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[51, Siehe Section 2.3]. Als URL-Präfix dient *https://*. Bei der Verschlüsselung und Herstellung der Datenintegrität bleibt die eigentliche HTTP-Syntax intakt, ergo handelt es sich um eine Verschlüsselung der einzelnen HTTP-Pakete. Dazu verschickt der Client beim Verbindungsaufbau über Port 433 ein *TLS ClientHello* an den Server, woraufhin der TLS-

Handshake initiiert wird. Dieser Handshake beinhaltet die Überprüfung der Integrität des Servers 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 dem Zertifikat einer Zertifizierungsstelle. 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 in so fern 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[57]. Beim RSA-Handshake wird vom Client ein symmetrischer Schlüssel erzeugt, dieser wird mit dem öffentlichen Schlüssel des Servers verschlüsselt und dem Server mitgeteilt. 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 mit dem symmetrischen Schlüssel 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. Ergo 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.

Common Internet File System (CIFS)

CIFS ist ein von der Firma Microsoft 1996 eingeführtes Datentransferprotokoll auf Basis von Server Message Block[6] und Network Basic Input Output System (NetBIOS) über TCP/IP. Das Protokoll ist nicht nur auf Dateitransfer 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*[54, S. 16]. Beim *Locking* wird der Zugriff auf eine Datei beschränkt, um deren Korruption zu vermeiden, welche passieren kann, wenn mehrere Nutzer auf die gleiche Datei schreiben. 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 Korruption geschützt. Die von der Internet Assigned Numbers Authority (IANA) für CIFS vergebene Portnummer ist: 445[54, 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 6 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[54, 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

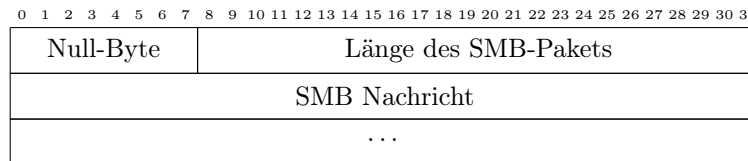


Abbildung 6: Aufbau eines SMB-Pakets

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.

Herleitung eines Lösungsansatzes

Anforderungsanalyse

Nachfolgend werden die ermittelten funktionalen und nichthypfunktionalen Anforderungen erläutert. Funktionale Anforderungen stellen das “eigentliche Systemverhalten und die jeweiligen Funktionen des zu erstellenden Produkts”[52, 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[17, S. 292]. Um die funktionalen und nichthypfunktionalen Anforderungen besser einordnen zu können, werden folgende Schlüsselwörter zum Kennzeichnen für Anforderungen nach RFC 2119[9] 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 man sie vermeiden kann.

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[19].

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 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 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 3: Funktionale Anforderung FA3

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

Tabelle 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 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 6: Funktionale Anforderung FA6

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

Tabelle 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 8: Funktionale Anforderung FA8

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:	
<ul style="list-style-type: none"> • Python • Bash • PHP • Javascript 	

Tabelle 9: Nicht-Funktionale Anforderung NFA1

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

Tabelle 10: 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 11: Nicht-Funktionale Anforderung NFA3

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

Tabelle 12: Nicht-Funktionale Anforderung NFA4

NFA5: Kontrollierbarkeit	Priorität: MUSS
Das zu entwickelnde System MUSS bei wachsendem Komplexitätsgrad und Anzahl von Messpunkten steuerbar und kontrollierbar bleiben.	

Tabelle 13: Nicht-Funktionale Anforderung NFA5

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.

Messensor 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. Die einzelnen Messsensoren senden ihre Messdaten zur Datenbank. Der Webserver mit der Visualisierungslösung stellt diese Daten für den Nutzer aufbereitet grafisch dar.

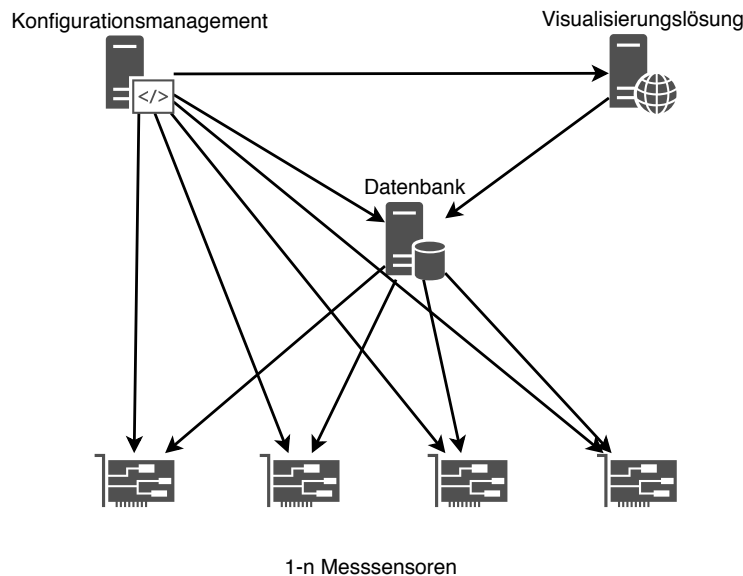


Abbildung 7: Zusammenspiel aller Systemkomponenten

Datenmodell

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 Database (TSDB)s. Dieses Kapitel versucht einen Überblick über alle Datenbanktypen zu geben und den passenden Datenbanktyp für das Projekt festzulegen.

Relationale Datenbanken

Bei relationalen Datenbanken handelt es sich um tabellenbasierte Datensätze. Als Grundlage dieser Datenbanken dienen Relationen. Relationen sind algebraische Strukturen und damit wohldefinierte Beschreibungen der Tabellen. Also in sich selbst abgeschlossene Tautologien. Für Schreib- und Leseoperationen auf diesen durch Relationen beschriebenen Tabellen wird eine formale Sprache verwendet: die relationale Algebra. Jede Tabelle wird beim Relationenmodell als eine Menge untergeordneter Tupel aufgefasst[29, 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 manipulieren 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 relationalen 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 2 zeigt das Hinzufügen neuer Daten, das Abrufen von Daten, das Manipulieren von Daten und das Erzeugen einer neuen Tabelle. Als Implementierung wurde *sqlite* verwendet. 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 2: 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

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 Tupeln mit einem Schlüssel und einem Wert. 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 zeitlichabhängiger Daten gespeichert werden müssen. TSDBs sammeln große Datenmengen und benutzen einen Zeitstempel als Index für diese Daten. Die gewonnenen Daten werden demnach einem festen Zeitpunkt zugeordnet. Dies macht eine TSDB 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 TSDB im Monitoring-System *Prometheus*. Trotz dieser Kategorisierungen haben NoSQL-Datenbanken mehrere gemeinsame Eigenschaften. So verbindet sie der Verzicht von Eigenschaften relationaler Datenbanken für höhere Skalierbarkeit[16, 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. Alle Daten werden in einer Tabelle gespeichert, welche sich beliebig horizontal durch Verwendung von mehr Rechenzeit skalieren lässt. Dies äußert sich in einer hohen Leistung, Skalierbarkeit und Flexibilität von NoSQL-Datenbanken[53].

Datenmodell

Es werden folgende Informationen als Daten erwartet:

- Ein sekundengenaue Zeitstempel.
- Der DNS-Eintrag oder die IP-Adresse des Messknotens
- Die Messdaten

Da die Informationen nicht sonderlich komplex sind, bietet sich eine NoSQL-Datenbank für die Speicherung an. Eine NoSQL-Datenbank hat den Vorteil, dass sich die Datenbank einfacher skalieren lässt und performanter ist. Durch die strikte Relation zwischen Messdaten und ihrem sekundengenauen Zeitstempel fällt die Wahl auf eine TSDB. Folgende TSDB-Implementierungen stehen zur näheren Auswahl:

- *InfluxDB*[25]
- *OpenTSDB*[38]
- *Prometheus*[42]

InfluxDB und *OpenTSDB* sind beides generische Vertreter von einer TDSB und vielseitig einsetzbar. Interessant für dieses Projekt ist jedoch *Prometheus*. *Prometheus* beschränkt sich auf Monitoringdaten, bietet jedoch diverse Vorzüge gegenüber den anderen beiden großen Vertretern von TSDBs (Details zum internen Aufbau und den Vorteilen von *Prometheus* sind im Kapitel *Prometheus* zu finden). Daten werden in *Prometheus* als Tupel aus 64-bit Fließkommazahlen und einem Millisekunden genauen Zeitstempel gespeichert[41]. Diese Tupel werden mit einem Identifikator versehen. Dieser Identifikator besteht aus einem eindeutigen Namen für die Metrik und einem Key-Value-Store (siehe Quelltext 3 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* gesammelt bis sie, zusammen mit Metadaten und einer Indexdatei, in einem Verzeichnis abgelegt werden. Metadaten sind Informationen über die gesammelten Daten, darunter fallen *Labels*. Die Indexdatei speichert die Relation zwischen den Metriknamen und den Metriken, die gestückelt abgelegt werden. Diese Stückelungen werden auch *Chunks* genannt[45]. Bevor diese Metriken gespeichert werden, legt *Prometheus* außerdem einen Write-Ahead-Log (WAL) an. Der WAL dient zur Wiederherstellung von Daten nach einem Crash. Gelöschte Daten werden nicht unwiderruflich gelöscht, sondern in *Tombstone*-Dateien abgelegt. Quelltext 4 zeigt eine solche Verzeichnisstruktur. Die *Prometheus*-Datenbank arbeitet demnach dateibasiert im Gegensatz zu BLOBs in SQL-basierten Datenbanken. *Prometheus*.

Quelltext 3: Prometheus Datenformat und Beispiel

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

```

1 ./data/01BKGv7JBM69T2G1BGBGM6KB12
2 ./data/01BKGv7JBM69T2G1BGBGM6KB12/meta.json
3 ./data/01BKGv7JBM69T2G1BGBGM6KB12/wal
4 ./data/01BKGv7JBM69T2G1BGBGM6KB12/wal/000002
5 ./data/01BKGv7JBM69T2G1BGBGM6KB12/wal/000001
6 ./data/01BKGTZQ1SYQJTR4PB43C8PD98
7 ./data/01BKGTZQ1SYQJTR4PB43C8PD98/meta.json
8 ./data/01BKGTZQ1SYQJTR4PB43C8PD98/index
9 ./data/01BKGTZQ1SYQJTR4PB43C8PD98/chunks
10 ./data/01BKGTZQ1SYQJTR4PB43C8PD98/chunks/000001
11 ./data/01BKGTZQ1SYQJTR4PB43C8PD98/tombstones
12 ./data/01BKGTZQ1HHWHV8FBjXW1Y3W0K
13 ./data/01BKGTZQ1HHWHV8FBjXW1Y3W0K/meta.json
14 ./data/01BKGTZQ1HHWHV8FBjXW1Y3W0K/wal
15 ./data/01BKGTZQ1HHWHV8FBjXW1Y3W0K/wal/000001
16 ./data/01BKGv7JC0RY8A6MACW02A2PJD
17 ./data/01BKGv7JC0RY8A6MACW02A2PJD/meta.json
18 ./data/01BKGv7JC0RY8A6MACW02A2PJD/index
19 ./data/01BKGv7JC0RY8A6MACW02A2PJD/chunks
20 ./data/01BKGv7JC0RY8A6MACW02A2PJD/chunks/000001
21 ./data/01BKGv7JC0RY8A6MACW02A2PJD/tombstones

```

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[46] und im Jahr 2015 als Open-source Software freigegeben. *Prometheus* entstand aus der Not heraus, hunderte von Mikroservices und tausende von Service-Instanzen in einem firmeninternen Container-Cluster zu überwachen[60]. Seitdem setzt sich *Prometheus* besonders im Cloud-Bereich durch und wurde 2016 sogar in die *Cloud Native Computing Foundation* aufgenommen[11], einer Organisation zur Förderung von Cloud-Software im Opensource-Bereich.

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 insbesondere im Cloud-Bereich durch[27]. *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). Die TDSB wurde bereits im Kapitel Datenmodell erläutert. Der HTTP-Server bietet eine grafische Oberfläche an, um Queries auf der TDSB zu testen, inklusive dem Anzeigen von Graphen. Abbildung 9 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 *Alertmanager* angedockt. Das *Pushgateway* bietet einen statischen Bezugspunkt und Puffer, von dem der *Prometheus Server* mit dem *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, die die Logik zum Sammeln von Messdaten besitzen und diese via einem HTTP-Server dem *Prometheus Server* zur Verfügung stellen. 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 über diesen Host. Außerdem werden diverse Services, die auf dem Host laufen, automatisch erkannt und Messdaten für diese Services zur Verfügung gestellt. Wenn beispielsweise ein HTTP-Server auf Port 80 dieses Hosts läuft, sammelt der sich auf dem Host befindliche *Node Exporter* automatisch Messdaten über den Service und bietet diese dem *Prometheus Server* über einen eigenen HTTP-Server an. Der *Blackbox Exporter* andererseits baut zwar ebenfalls einen HTTP-Server als Quelle für den *Prometheus Server* auf, aber Daten werden nicht über lokale Services gesammelt. Stattdessen werden von dem *Blackbox Exporter* aus diverse Tests durchgeführt, wie beispielsweise die Erreichbarkeit von Kerndiensten wie DNS. Die Abbildung 8 zeigt den internen Aufbau des *Prometheus Servers* und dessen Zusammenspiel mit den einzelnen Komponenten. Alle Verbindungen außerhalb des *Prometheus Servers* sind HTTP-Anfragen einer REST-API. Alle Komponenten sind statische Binärdateien und geschrieben in der Sprache *Go*.

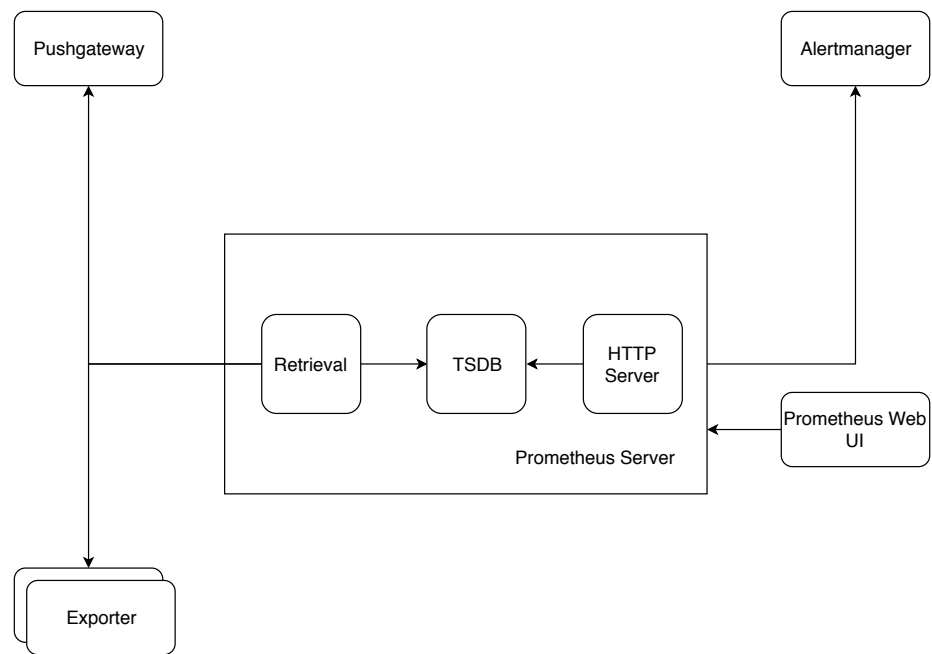


Abbildung 8: Interner Aufbau des Prometheus Server und dessen Komponenten

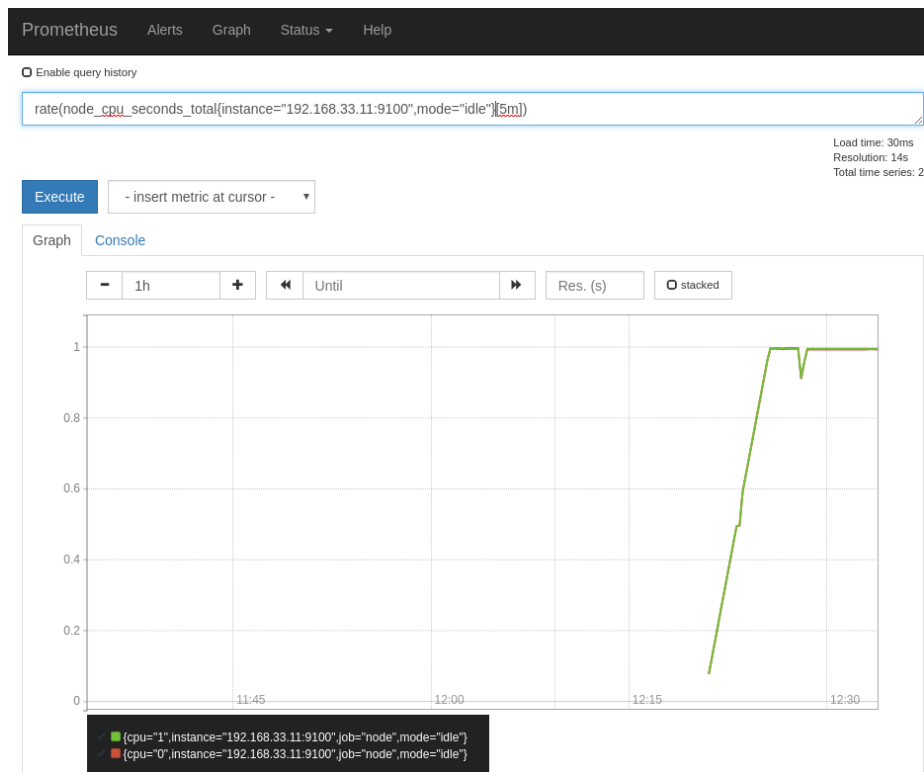


Abbildung 9: Prometheus Web UI Beispiel

PromQL

Prometheus besitzt zum Abfragen der Messdaten eine eigene Query-Language namens *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[44]. Dies ist insbesondere für die grafische Darstellung der Messdaten relevant. Dazu unterstützt die *PromQL* vier verschiedene Datentypen[44]:

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 unbenutzt).

Der *Instant Vector* besteht aus einem eindeutigen Namen, einer Metrik und einem Key-Value-Store (siehe auch Quelltext 3). Innerhalb eines *Instant Vectors*

lassen sich reguläre Ausdrücke (Regex) anwenden. Reguläre Ausdrücke beschreiben anhand von fest definierten Zeichen und syntaktischer Regeln eine Menge von Zeichenketten[5]. Für reguläre Ausdrücke verwendet *PromQL* die von der Firma *Google* entwickelte Bibliothek *RE2*[49]. 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[43].

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*[21]. *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 10 zeigt *Grafanas* Platz in der *Prometheus*-Landschaft).

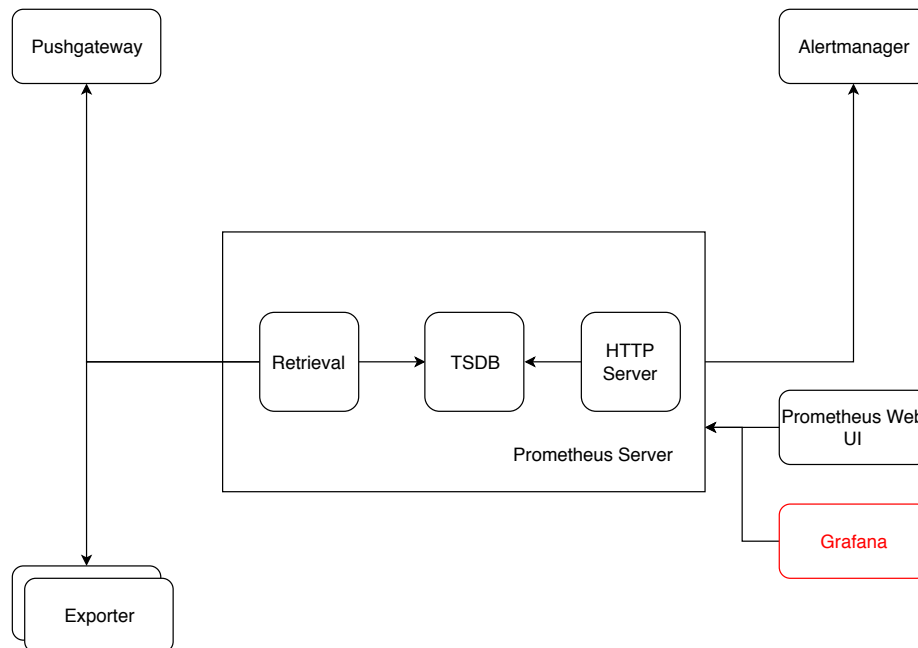


Abbildung 10: Eingliederung von Grafana in Prometheus

Ein weiteres Merkmal von *Grafana* sind *Grafanas* Provisionierungseigenschaften. Durch diverse Einstellungen in *Grafanas* Konfigurationsdateien und Umgebungsvariablen lässt sich *Grafana* gut in Cloud-Umgebungen ausrollen (Umgebungsvariablen sind temporäre oder permanente Variablen im Betriebssystem).

Konfigurationsmanagement

In der Vergangenheit wurden Rechnersysteme zunehmend vertikal skaliert, das heißt, die technischen Ressourcen des Rechnersystems wurden angehoben (beispielsweise der Read-Only Memory (RAM)). In den letzten Jahren wurde zunehmend festgestellt, dass diese Vorgehensweise nicht nur sehr kostenintensiv ist, sondern auch nicht besonders 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 (zum Beispiel sind heutige Cloud-Systeme de facto alle horizontal skaliert). Diese 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 zu steuern, also gewisse Aufgaben auf den jeweiligen Systemen anzustoßen. *Konfigurationsmanagement* dagegen umfasst die Installation und Konfiguration der Systeme, 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*.

Ansible

Die Entwicklung an *Ansible* startete im Februar 2012 unter Michael DeHaan als Opensource-Anwendung[15]. 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[14]. Im Jahr 2015 wurde die Firma durch den großen Linux-Dienstleister *Red Hat* übernommen[50]. Wenig später begann die Opensource-Gemeinschaft damit, *Ansible* um unzählige Module zu erweitern und so den Funktionsumfang zu vergrößern.

Funktionsweise

Ansible arbeitet mit Templates zur Orchestrierung und Konfiguration von Systemen. 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[61]. Außerdem besteht die Möglichkeit, in der Programmiersprache *Python* geschriebene Module mit in YAML aufgeschriebenen 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, sind in *Ansible* in Gruppen zu organisieren. Anhand vom Namen der Gruppe und des Zielsystems lassen sich unterschiedliche vom Administrator definierte Daten laden. Diese Daten werden in die mit YAML verfassten Aufgaben eingefügt. Auf diese Weise können komplexe Gebilde von Systemen abgebildet werden und Gemeinsamkeiten in der Orchestrierung und Konfiguration effektiv ausgenutzt werden. Bei einzelnen Konfigurationsdateien wird die auf *Python* basierte Template-Engine *Jinja2* benutzt[26]. Durch *Jinja2* lassen sich komplexe Konfigurationsdateien 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 gefüttert. Um die Orchestrierung oder Konfiguration zu starten, reicht es, wenn sich *Ansible* auf dem System, welches zur Pflege 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 in YAML-Dateien festgelegten Aufgaben ab. Dies geschieht durch die Verwendung der Sprache *Python* oder einfach durch Kommandos in der entfernten Kommandozeile auf den Zielsystemen. Durch diese Vorgehensweise arbeitet *Ansible agentless*, also ohne einem Agenten auf dem Zielsystem. Dies ermöglicht den Einsatz von *Ansible* auch auf Systemen mit eingeschränkter Erweiterbarkeit (beispielsweise Switches oder Router). Außerdem arbeitet *Ansible* idempotent, dadurch lassen sich Aufgaben beliebig oft ausführen und führen immer zum gleichen Ergebnis.

Organisationsstruktur

Ansible hat eine definierte Struktur zur Speicherung der Konfigurationsdateien, Aufgaben, Datenstrukturen für die Templates und Templates. Dabei bedient sich *Ansible* bei Wörtern aus der Schauspielerei, um den Einstieg in die Software zu erleichtern. *Playbooks* (Deutsch: Drehbücher) werden verwendet, um einzelnen Hosts *Roles* (Deutsch: Rollen) zu zuweisen. Die nötigen Daten zum Abspielen dieser Rollen befinden sich im *Inventory* (Deutsch: Inventar), zu dem die Hosts ebenfalls gehören. Quelltext 5 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 fest machen. 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. Dies ist eine YAML-Datei. YAML-Dateien werden häufig auch mit YML abgekürzt. Der Ordner *roles* beinhaltet die Rolle *hello_world* mit den Unterordnern *defaults*, *files*, *tasks* und *templates*. Im Ordner *defaults* befinden sich Standardbelegungen für Variablen zur Initialisierung. 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 5: Organisationsstruktur eines Ansible Projekts

```

1  .
2  |-- ansible.cfg
3  |-- group_vars
4  |-- hosts
5  |-- host_vars
6  |-- playbooks
7  |   '-- configure_hosts.yml
8  '-- roles
9      '-- hello_world
10         |-- defaults
11         |   '-- main.yml
12         |-- files
13         |   '-- static_file.txt
14         |-- tasks
15         |   '-- main.yml
16         '-- templates
17         '-- configuration.j2

```

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 *Jinja2* erreicht (If/Else, For-Schleifen, While-Schleifen etc). Dadurch ist *Jinja2* in der Theorie Turing-Vollständig. Quelltext 6 ist ein Beispiel

für die Dateistruktur eines *Ansible Tasks*. Der *Task* verteilt SSH-Schlüssel auf eine beliebige Anzahl von 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 Dateiondung *.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 6 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 7 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. Gefüttert wird dieses Template durch Quelltext 8. Quelltext 8 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 6: 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 7: 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 8: Daten für das Jinja2-Template Beispiel

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

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. Üblicherweise durch moderne kryptografisch sichere Verfahren wie RSA oder Elliptischen Kurven. Der öffentliche Schlüssel wird auf dem Zielsystem platziert, meistens 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].

Wahl der Hardware

Da das zu entwickelnde 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+* kommt mit folgenden Spezifikationen[48]:

- SoC: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- GPU: Broadcom Videocore-IV
- RAM: 1GB 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, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- Dimensionen: 82mm x 56mm x 19.5mm, 50g Gewicht
- Preis: 32.88 Britische Pfund[47] (36.45 € Stand: 2018-09-03)

Im Gegenzug dazu die *Odroid XU4* Spezifikationen[36]:

- SoC: Samsung Exynos5422 CortexTM-A15 2Ghz and CortexTM-A7 Octa core CPUs
- GPU: Mali-T628 MP6(OpenGL ES 3.1/2.0/1.1 and OpenCL 1.2 Full profile)
- RAM: 2Gbyte LPDDR3 RAM PoP stacked

- Netzwerk: Gigabit Ethernet, kein WLAN
- Bluetooth: nicht vorhanden
- Speicher: Micro-SD
- GPIO: 30-pin GPIO header[34]
- Anschlüsse: 2x USB 3.0, 1x USB 2.0, HDMI 1.4a
- Dimensionen: 83mm x 58mm x 20 mm
- Preis: 59 US Dollar[35] (50,80 € Stand: 2018-09-03)

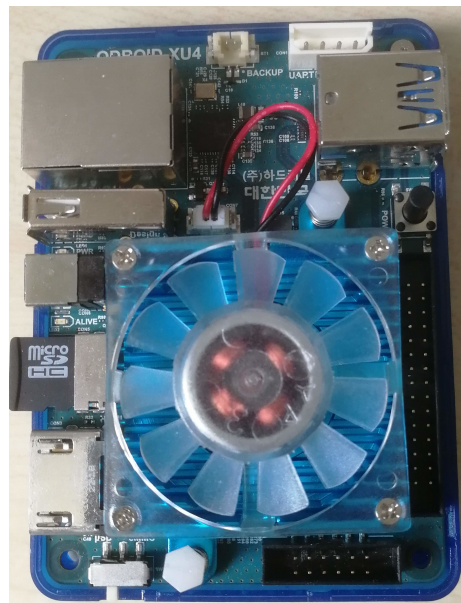


Abbildung 11: Darstellung eines Odroid XU4

Projektumsetzung

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

Erfüllung der Anforderungen

Nachfolgend sind die evaluierten Anforderungen und deren Umsetzung spezifiziert.

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

Tabelle 14: Umsetzung der Funktionalen Anforderungen

Anforderung	Umsetzung
NFA1	Das für das Deployment verwendete Projekt <i>Ansible</i> benutzt <i>Python</i> . Für die Erweiterung um <i>CIFS</i> -Support ist nur die Verwendung der Programmiersprache <i>Golang</i> möglich.
NFA2	Die niedrigen Beschaffungskosten sind durch die Wahl der <i>Odroid XU4</i> Plattform gegeben.
NFA3	Die für das Deployment benötigten Passwörter werden mit gängigen Verschlüsselungsverfahren in <i>Ansible-Vault</i> hinterlegt. Die Verbindungen zu den Zielsystemen finden verschlüsselt und mit einer Authentizitätsprüfung statt.
NFA4	Das System ist horizontal skalierbar durch den Einsatz von <i>Prometheus</i> als Cluster einsetzbare Datenbank und <i>Ansible</i> als Automatisierungswerkzeug zum Deployment.
NFA5	Das System bleibt kontrollierbar durch den Einsatz von <i>Ansible</i> .

Tabelle 15: Umsetzung der Nicht-Funktionalen Anforderungen

Test- und Entwicklungssystem

Da die benötigte Hardware zu Projektanfang noch nicht zur Verfügung stand, galt es ein Test- und Entwicklungssystem aufzubauen. Als Grundlage für dieses Test- und Entwicklungssystem dient die Software *Vagrant*[59] 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[22]. 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*. *Vagrant* selbst ist eine statisch gebaute aus *Golang* kompilierte Binärdatei, dies erleichtert das Deployment auf allen gängigen Systemen. Außerdem zeichnet sich *Vagrant* durch eine enge Verzahnung mit Automatisierungswerkzeugen aus, darunter auch *Ansible*[58]. *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 dessen Orchestrierung organisiert wird. Diese *Ruby* datei wird von *Vagrant* eingelesen und *Vagrant* dockt über die APIs an den jeweiligen *Provider* an und spawned 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). 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 9 zeigt die *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*, dem selben Betriebssystemabbild und der IP-Adresse **192.168.33.11**. Die virtuellen Maschinen liegen beide im selben Netzwerk damit sie sich gegenseitig erreichen können. Der Wirt-Host auf dem die virtuellen Maschinen laufen ist ebenfalls durch ein Interface zu den beiden Maschinen 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 dem Standard-Provider *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 das Testsystem später auf jeden anderen Hypervisor übertragen und die virtuellen Maschinen lassen sich über den Befehl *vagrant snapshot* leicht zu einem vorher definierten Zustand zurücksetzen.

Quelltext 9: 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 10 zeigt eine SSH-Konfigurationsdatei. Die erste Zeile vergibt einen Alias für die *glsip*-Adresse. Auf diese Weise ist es nicht nötig einen Eintrag im DNS vorzunehmen für ein Testsystem. 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 eine Gerätedatei innerhalb des *Linux*-Betriebssystems zum Verwerfen von Daten[24]. Die *Fingerprints* der *Host-Zertifikate* werden für die Testsysteme verworfen, da bei jedem neu Erstellen die Authentizitätsprüfung von SSH sonst einen Fehler werfen 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 10: 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
```

Entwicklungsprozess

Versionsverwaltung

Für den Entwicklungsprozess, des Konfigurationsmanagement und der 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 Daten[20]. 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[55]. Die speziellen Vorteile von *Git* sind:

- Ein Dezentraler Ansatz.
- 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*, den 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[39]. 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*) sich gegenseitig in die Quere kommen. Ist die Entwicklung an der Funktion abgeschlossen wird der Entwicklungszweig wieder mit dem ursprünglichen Hauptzweig (auch *Master* genannt) vereinigt. Quelltext 11 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[10]. 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 erfasst mit zusätzlichen Metadaten. Diese Metadaten beinhalten: die eindeutige Nummer der Änderung, der Entwicklungszweig, der Name des Autors und seine Email-Adresse, ein Zeitstempel und ein Kommentar zu der Änderung.

Quelltext 11: 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 [..]
```

Entwicklungsmethoden

Prototyp

Betriebstests

Fazit

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 13.05.2018).
- [2] *Ansible-Vault Dokumentation.* URL: https://docs.ansible.com/ansible/2.6/user_guide/vault.html (besucht am 28.08.2018).
- [3] *Begriffserklärung von DNS in der Wikipedia.* URL: https://de.wikipedia.org/wiki/Domain_Name_System (besucht am 05.05.2018).
- [4] *Begriffserklärung von Jitter in der Wikipedia.* URL: <https://de.wikipedia.org/wiki/Jitter> (besucht am 14.04.2018).
- [5] *Begriffserklärung von Regulären Ausdrücken in der Wikipedia.* URL: https://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck (besucht am 03.08.2018).
- [6] *Begriffserklärung von SMB in der Wikipedia.* URL: https://de.wikipedia.org/wiki/Server_Message_Block (besucht am 26.05.2018).
- [7] 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 11.05.2018).
- [8] 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 11.05.2018).
- [9] 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 22.04.2018).
- [10] S. Chacon und B. Straub. *Pro Git*. The expert's voice. Apress, 2014. ISBN: 9781484200766. URL: <https://books.google.de/books?id=jVYnCGAAQBAJ>.
- [11] *Cloud Native Computing Foundation.* URL: <https://www.cncf.io> (besucht am 31.07.2018).

- [12] 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 20.07.2018).
- [13] Human Rights Council. *The promotion, protection and enjoyment of human rights on the Internet*. United Nations. Juni 2016. URL: https://www.article19.org/data/files/Internet_Statement_Adopted.pdf (besucht am 11.04.2018).
- [14] Michael DeHaan. *Introducing AnsibleWorks*. 4. März 2013. URL: <https://www.ansible.com/blog/2013/03/04/introducing-ansibleworks> (besucht am 12.08.2018).
- [15] Michael DeHaan. *The Origins of Ansible*. 8. Dez. 2013. URL: <https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible> (besucht am 12.08.2018).
- [16] Ted Dunning und Ellen Friedman. *Time Series Databases: New Ways to Store and Access Data*. O'Reilly, Sep. 2014. ISBN: 9781491917022.
- [17] 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 21.04.2018).
- [18] 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 12.05.2018).
- [19] 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 23.04.2018).
- [20] *Git - Wikipedia Eintrag*. URL: <https://de.wikipedia.org/wiki/Git> (besucht am 14.09.2018).
- [21] *Grafana Projektwebseite*. URL: <https://grafana.com/> (besucht am 03.08.2018).
- [22] *Hashicorp Wikipedia Artikel*. URL: <https://en.wikipedia.org/wiki/HashiCorp> (besucht am 10.09.2018).
- [23] Z. Hu u. a. *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. RFC Editor, Mai 2016. (Besucht am 19.07.2018).
- [24] 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 14.09.2018).
- [25] *InfluxDB Projektwebseite*. URL: <https://www.influxdata.com> (besucht am 27.07.2018).
- [26] *Jinja2 Projektwebseite*. URL: <http://jinja.pocoo.org/> (besucht am 18.08.2018).

- [27] 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 31.07.2018).
- [28] *Liste aller Internet Top-Level Domains in der Wikipedia*. URL: https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains (besucht am 10.05.2018).
- [29] Andreas Meier. *Relationale Datenbanken: Eine Einführung für die Praxis*. Springer Berlin Heidelberg, 2013. ISBN: 9783662097410. URL: <https://books.google.de/books?id=DI0eBwAAQBAJ> (besucht am 22.07.2018).
- [30] 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 05.05.2018).
- [31] 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 05.05.2018).
- [32] 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 05.05.2018).
- [33] 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 05.05.2018).
- [34] *Odroid XU4 GPIO specifications*. URL: https://wiki.odroid.com/odroid-xu4/hardware/expansion_connectors (besucht am 03.09.2018).
- [35] *Odroid XU4 Preis*. URL: https://www.hardkernel.com/main/shop/good_list.php?lang=en (besucht am 03.09.2018).
- [36] *Odroid XU4 specifications*. URL: https://www.hardkernel.com/main/products/prdt_info.php (besucht am 03.09.2018).
- [37] *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 02.06.2018).
- [38] *OpenTSDB Projektwebseite*. URL: <http://opentsdb.net/> (besucht am 27.07.2018).
- [39] Stefan Otte. “Version control systems”. In: *Computer Systems and Telematics* (2009).
- [40] *Prometheus compared to other solutions*. URL: <https://prometheus.io/docs/introduction/comparison/> (besucht am 27.07.2018).

- [41] *Prometheus Data Model Layout*. URL: https://prometheus.io/docs/concepts/data_model/ (besucht am 27.07.2018).
- [42] *Prometheus Projektwebseite*. URL: <https://prometheus.io/> (besucht am 27.07.2018).
- [43] *Prometheus PromQL Functions Overview*. URL: <https://prometheus.io/docs/prometheus/latest/querying/functions/> (besucht am 03.08.2018).
- [44] *Prometheus Querying Language*. URL: <https://prometheus.io/docs/prometheus/latest/querying/basics/> (besucht am 01.08.2018).
- [45] *Prometheus Storage Layout*. URL: <https://prometheus.io/docs/prometheus/latest/storage/> (besucht am 27.07.2018).
- [46] *Prometheus Übersicht*. URL: <https://prometheus.io/docs/introduction/overview/> (besucht am 31.07.2018).
- [47] *Raspberry Pi 3B+ Preis*. URL: <https://uk.rs-online.com/web/p/processor-microcontroller-development-kits/1720555> (besucht am 03.09.2018).
- [48] *Raspberry Pi 3B+ specifications*. URL: <https://www.raspberrypi.org/magpi/raspberry-pi-specs-benchmarks/> (besucht am 03.09.2018).
- [49] *RE2 Projektwebseite*. URL: <https://github.com/google/re2> (besucht am 03.08.2018).
- [50] *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 12.08.2018).
- [51] 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 13.05.2018).
- [52] 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=M90jBAAQBAJ> (besucht am 21.04.2018).
- [53] Ben Scofield. *NoSQL – Death to Relational Databases(?)* Jan. 2010. URL: <https://www.slideshare.net/bscofield/nosql-codemash-2010> (besucht am 22.07.2018).
- [54] *Server Message Block (SMB) Protocol*. 47.0. Microsoft Corporation. Dez. 2017. URL: [https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SMB2/\[MS-SMB2\].pdf](https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SMB2/[MS-SMB2].pdf) (besucht am 26.05.2018).
- [55] 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>.

- [56] *Status Website der DNS Root Name Server*. URL: <http://root-servers.org/> (besucht am 11.05.2018).
- [57] *Transport Layer Security (TLS)*. URL: <https://hpbn.co/transport-layer-security-tls/> (besucht am 13.05.2018).
- [58] *Vagrant Dokumentation*. URL: <https://www.vagrantup.com/docs/index.html> (besucht am 10.09.2018).
- [59] *Vagrant Projektwebseite*. URL: <https://www.vagrantup.com/> (besucht am 10.09.2018).
- [60] Julius Volz. *Monitoring, the Prometheus Way*. Youtube. 8. Mai 2017. URL: <https://www.youtube.com/watch?v=PDxcEzu62jk> (besucht am 31.07.2018).
- [61] *YAML in der deutschen Wikipedia*. URL: <https://de.wikipedia.org/wiki/YAML> (besucht am 18.08.2018).

Quelltextverzeichnis

1	Eine HTTP-Anfrage an http://tu-clausthal.de	16
2	Verwendung von Structured Query Language	24
3	Prometheus Datenformat und Beispiel	26
4	Beispiel für das Prometheus Datenmodell	27
5	Organisationsstruktur eines Ansible Projekts	34
6	Beispiel eines Ansible Tasks	36
7	Beispiel eines Jinja2-Templates	36
8	Daten für das Jinja2-Template Beispiel	36
9	Vagrantfile der Test- und Entwicklungsumgebung	42
10	Beispiel einer SSH Konfigurationsdatei	43
11	Beispiele für die Verwendung des Git-Clients	45

Tabellenverzeichnis

1	Funktionale Anforderung FA1	20
2	Funktionale Anforderung FA2	20
3	Funktionale Anforderung FA3	20
4	Funktionale Anforderung FA4	20
5	Funktionale Anforderung FA5	20
6	Funktionale Anforderung FA6	20
7	Funktionale Anforderung FA7	21
8	Funktionale Anforderung FA8	21
9	Nicht-Funktionale Anforderung NFA1	21
10	Nicht-Funktionale Anforderung NFA2	21
11	Nicht-Funktionale Anforderung NFA3	22
12	Nicht-Funktionale Anforderung NFA4	22
13	Nicht-Funktionale Anforderung NFA5	22
14	Umsetzung der Funktionalen Anforderungen	39
15	Umsetzung der Nicht-Funktionalen Anforderungen	40

Abbildungsverzeichnis

1	Veranschaulichung der Problemstellung	9
2	Veranschaulichung der Problemlösung	9
3	Veranschaulichung der Assoziationen verschiedenener offener Sys- teme mit diversen Anwendungen über ein physisches Medium . .	11
4	Das OSI-Schichtenmodell mit Protokollbeispielen und verwende- ten Einheiten	12
5	Bestandteile eines FQDN mit optionalem Hostname und Third- Level Domain	13
6	Aufbau eines SMB-Pakets	18
7	Zusammenspiel aller Systemkomponenten	23
8	Interner Aufbau des Prometheus Server und dessen Komponenten	29
9	Prometheus Web UI Beispiel	30
10	Eingliederung von Grafana in Prometheus	31
11	Darstellung eines Odroid XU4	38

Glossar

AES Advanced Encryption Standard. Vom National Institute of Standards and Technology standardisiertes Verschlüsselungsverfahren.. 37

ARPANET Advanced Research Projects Agency Network. Der Vorgänger des Internets und ehemaliges Projekt der US-Luftwaffe.. 12

BLOB Binary Large Object. Besonders große Binärdateien.. 25, 26, 44

CIFS Common Internet File System. Ein offenes Protokoll zum Filetransfer und diversen anderen Dienst.. 4, 7, 17, 18, 20, 43

DNS Domain Name System. Ist für die Namensauflösung im Internet zuständig.. 4, 7, 8, 11–14, 20, 25, 28, 42

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.. 12, 13, 54

FTP File Transfer Protocol. Datentransferprotokoll auf der Anwendungsschicht des OSI-Modells.. 12

HTTP Hypertext Transport Protocol. Das im Internet übliche Protokoll zur Übermittlung von Webseiten.. 4, 7, 11, 14–18, 20, 28

HTTPS Hypertext Transport Protocol Secure. Eine Erweiterung für HTTP, welche HTTP mit TLS versieht.. 4, 7, 14, 16, 17, 20

IANA Internet Assigned Numbers Authority. Abteilung der ICANN. Ist für die Vergabe von Nummern und Namen im Internet zuständig.. 18

ICANN Internet Corporation for Assigned Names and Numbers. Internationale Aufsichtsbehörde für das Internet. ICANN koordiniert die Vergabe von einmaligen IP-Adressen und DNS Hostnamen.. 13

- IP** Internet Protocol. Das Standardprotokoll auf der Vermittlungsschicht des OSI-Modells.. 11–14, 17, 25
- NetBIOS** Network Basic Input Output System. Programmierschnittstelle zur Kommunikation zwischen zwei Programmen.. 17
- NIC** Network Information Center. Verwaltung einer oder mehrerer Top-Level Domains im Internet.. 12, 13
- OSI-Modell** Open Systems Interconnection Model. Referenzmodell für Verbindungen im Internet.. 10, 11, 14
- PFS** Perfect Forward Secrecy. Bezeichnung für ein Verfahren, welches sicherstellt, dass im Falle eines Schlüsselverlusts bereits gesendete Daten nicht mehr entschlüsselt werden können.. 17
- RAM** Read-Only Memory. Flüchtiger aber sehr schneller Speicher in Rechner-systemen.. 32
- REST-API** Representational State Transfer Application Programming Interface. Auf HTTP basierende Schnittstelle zur Interaktion mit anderen Programmen.. 15, 28, 31
- RFC** Requests for Comments (deutsch: Bitte um Kommentare). Eine Reihe von technischer Standards, welche sich mit dem Internet befassen.. 12–14, 19
- RSA** Asymmetrisches kryptographisches Verfahren.. 17, 36
- SMB** Server Message Block. Protokoll auf der Anwendungsschicht des OSI-Modells zur Übermittlung von Daten und anderen Diensten in Rechnernetzen.. 17, 18, 54
- SQL** Structured Query Language. Eine Sprache zur Beschreibung von relationalen Datenbanken und dessen Operationen auf eben diesen.. 23, 24, 26, 30
- SSH** Secure Shell. Netzwerkprotokoll, welches Verschlüsselung bietet. Wird zum entfernten Aufruf einer Kommandozeile verwendet.. 33, 35, 36, 41, 42
- SSL** Secure Sockets Layer. Die veraltete Bezeichnung für TLS.. 16
- TCP** Transport Control Protocol. Ein Protokoll der Transportschicht des OSI-Modells. Im Gegensatz zu UDP ist TCP auf Datenintegrität und eine verlässliche Verbindung ausgelegt.. 11, 14, 17, 18
- TDSB** Time Series Database. Eine per Zeit indexierte Datenbank. Optimiert auf große Mengen an Daten, die eine strikte Relation zu der Zeit besitzen.. 23, 25–28

- TLD** Top-Level Domain. Höchste Stufe der DNS Auflösung im Internet.. 13
- TLS** Transport Layer Security. Zusätzliche Schicht für diverse Protokolle für Verschlüsselung, Entschlüsselung und Authentifikation von Daten.. 14, 16, 17
- TU Clausthal** Technische Universität Clausthal. 1775 gegründete Technische Universität im Oberharz (Niedersachsen).. 1, 6, 7, 21, 32
- UDP** User Datagramm Protocol. Einfaches Protokoll auf der Transportschicht zum senden von Daten. UDP besitzt keine Mechanismen zur Sicherstellung von Datenintegrität.. 14
- URL** Uniform Resource Locator. Vollwertiger Bezeichner einer Internetadresse mit vorangestelltem Protokoll und nachgestellten Pfad zur angeforderten Datei.. 15, 16
- VoIP** Voice over Internet Protocol. Die Übermittlung von Sprache über das Internet Protocol.. 7, 8
- WAL** Write-Ahead-Log. Ein Mechanismus in Prometheus zur Wiederherstellung von Metriken nach einem Crash.. 26
- YAML** Yet Another Markup Language. Vereinfachte Auszeichnungssprache für Datenstrukturen.. 33–35