HAW HAMBURG

Bachelorarbeit

Hendrik Krause

Entwicklung eines robusten, kooperativen Monitoring-Konzepts im Umfeld eines Fernsehstudios

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Hendrik Krause

Entwicklung eines robusten, kooperativen Monitoring-Konzepts im Umfeld eines Fernsehstudios

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke
Zweitgutachter: Prof. Dr. Klaus-Peter Kossakowski

Eingereicht am: 25.08.2020

Hendrik Krause

Thema der Arbeit

Entwicklung eines robusten, kooperativen Monitoring-Konzepts im Umfeld eines Fernsehstudios

Stichworte

Norddeutscher Rundfunk, NDR, Monitoring, Verteilte Systeme, Netzwerk, Prometheus, Grafana, Virtual Studio Manager, VSM

Kurzzusammenfassung

Der Betrieb eines Fernsehstudios erfordert den Einsatz und Betrieb einer Vielzahl von Geräten. Um diese sinnvoll und übersichtlich steuern zu können, wird beim Norddeutschen Rundfunk die Software „Virtual Studio Manager“ eingesetzt. Gleichzeitig nutzt der NDR die Monitoringsoftware „Prometheus“ zur Überwachung von Computern und Serverschränken. In dieser Bachelorarbeit wird ein Konzept zur Kommunikation dieser beiden Systeme entwickelt. Der Schwerpunkt liegt dabei auf der Robustheit der Lösung unter Umsetzung mit Hilfe des Protokolls „Ember+“.

Hendrik Krause

Title of Thesis

Developing a robust and cooperative concept of monitoring in a televisions studio environment

Keywords

Norddeutscher Rundfunk, NDR, monitoring, distributed systems, network, Prometheus, Grafana, Virtual Studio Manager, VSM

Abstract

To operate a television studio the usage and control of a large number of devices is required. In order to be able to manage these devices in a meaningful and well-arranged way, the software „Virtual Studio Manager“ is used at the german broadcaster „Norddeutscher

Rundfunk“. At the same time, the NDR uses the monitoring software „Prometheus“ to monitor computers and server racks. In this bachelor thesis a concept is developed that allows these two systems to communicate with each other. The focus lies on the robustness of the solution with the implementation using the protocol „Ember+“.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

API	Application Programming Interface
ARD	Arbeitsgemeinschaft der öffentlich-rechtlichen Rundfunkanstalten der Bundesrepublik Deutschland
ASN.1	Abstract Syntax Notation 1
BER	Basic Encoding Rules
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HAW	Hochschule für Angewandte Wissenschaften
HTTP	Hypertext Transport Protocol
IP	Internet Protocol
ISA	Instruction Set Architecture
ISO	Interantional Organization for Standardization
ITU	International Telecommunication Union
KISS	Keep it simple, stupid
MIB	Management Information Base
MVP	Minimum Viable Product
NDR	Norddeutscher Rundfunk
NetEm	Network Emulator
OSI	Open Systems Interconnection

RFC	Request for Comment
RRD	Round Robin Database
SNMP	Simple Network Management Protocol
TC	Traffic Control
TCP	Transmission Control Protocol
TSDB	Time-Series Database
UDP	User Datagram Protocol
URL	Uniform Ressource Locator
VSM	Virtual Studio Manager
WSL	Windows Subsystem for Linux
ÖRR	öffentlich-rechtlicher Rundfunk

1 Einführung und Ziele

Der öffentlich-rechtliche Rundfunk (ÖRR) in Deutschland ist in landesweite und bundesweite Rundfunkanstalten aufgeteilt, welche die Versorgung der Bevölkerung mit Radio- und Fernsehprogrammen sicherstellen sollen. Während letztere ausschließlich auf bundesweite Ausstrahlung ausgerichtet sind, haben sich erstere zur Arbeitsgemeinschaft der öffentlich-rechtlichen Rundfunkanstalten der Bundesrepublik Deutschland (ARD) zusammengeschlossen. Auf diese Weise übernehmen sie neben regionalen und landesweiten Sendungen auch einen bundesweiten Programmauftrag. Eine dieser neun Landesrundfunkanstalten der ARD ist der Norddeutscher Rundfunk (NDR), der die Bundesländer Hamburg, Mecklenburg-Vorpommern, Niedersachen und Schleswig-Holstein abdeckt und auch eine der bekanntesten Formate des deutschen ÖRR, die Tagesschau, beherbergt. Ein professionelles Fernsehstudio zu betreiben, setzt allerdings den Betrieb, die Bedienung und die Wartung von einer Vielzahl von Geräten voraus: Kameras, Mikrofone, Bild- und Ton-Mischpulte, Übertragungsstrecken und Beleuchtungsanlagen, um nur wenige zu nennen. Um diese übersichtlich verwalten zu können wird im Sendebetrieb die Software VSM eingesetzt.
Gleichzeitig bedarf es auch einer leistungsfähigen digitalen Infrastruktur, mit der die Journalisten, die Techniker, die Mediengestalter und der Verwaltungsapparat ihren jeweiligen Aufgaben nachgehen können. Um Ausfälle von Serverhardware oder nötige Wartungsarbeit frühzeitig zu erkennen, setzt der NDR daher vermehrt die Monitoring-Lösung „Prometheus“ ein. Gegenstand dieser Arbeit ist es, eine Lösung zu entwickeln, die es erlaubt, beide Systeme kooperativ nebeneinander betreiben zu können, sodass wichtige Statusmeldungen miteinander ausgetauscht werden.

1.1 Motivation

Die Fortschritte in der Informationstechnologie haben ermöglicht, dass nahezu alles, was man in einem Fernsehstudio und in einer Senderegie findet, ferngesteuert werden kann.

Ein Bild-Ingenieur stellt die Blenden der Studiokameras aus der Ferne ein und Beleuchtungszenarien werden auf Knopfdruck automatisch abgefahren. Die meisten Geräte in der Regie sind im Grunde nur Fernbedienungen für Systeme, die sich an ganz anderen Stellen im Gebäude, häufig im Keller, befinden. Während Steuersignale früher noch manuell und später mit analogen Stromsignalen gegeben wurden, wird zunehmend auf Vernetzung und paketbasierte Übermittlung von Daten umgestellt. Viele solcher Systeme getrennt in diesem Netzwerk nebeneinander zu betreiben, bedeutet aber auch einen hohen organisatorischen Aufwand. Das kann unter Umständen dazu führen, dass Situationen nicht richtig eingeschätzt werden. Wenn beispielsweise ein Sendeweg ausfällt und der diensthabende Techniker nur unpassende Informationen zur Fehlerbehebung zur Hand hat, kann das Problem im schlimmsten Fall nicht gelöst werden. Aber selbst das beste Szenario wäre dann noch ein merklicher Sendungsausfall von „nur“ wenigen Sekunden und ist daher zu vermeiden. Aus diesem Grund sind viele Systeme und Verteilungswege redundant aufgebaut, was allerdings die Übersichtlichkeit senkt und den Wartungsaufwand steigert. Die beste Redundanz ist jedoch hinfällig, wenn das als Sicherung gedachte Gerät ausgefallen ist, ohne dass es bemerkt wird. Um dem entgegenzuwirken ist es wichtig, dass Mitarbeiter aktiv informiert werden und rechtzeitig Gegenmaßnahmen einleiten können, falls Equipment nicht wie erwartet funktioniert.

1.2 Aufgabenstellung

Bisher sind im NDR dazu mehrere voneinander getrennte Systeme in verschiedenen Bereichen im Einsatz, denn jede Hard- und Software ist dabei für die jeweilige Aufgabe passend zugeschnitten. Wie eingangs erwähnt, haben zwei dieser Systeme dabei eine besondere Bedeutung für einander und sollen sich ergänzen. Das Studioequipment ist nicht nur mit VSM zu steuern, sondern soll zukünftig auch überwacht und aufgezeichnet werden. Die Umsetzung soll über den Export von Kennwerten erfolgen, sodass diese von der Monitoringsoftware Prometheus erfasst werden können. Gleichzeitig muss es mit dem zu entwickelnden System möglich sein, den bedienenden Techniker über Ausfälle zu informieren und Hinweise zur Fehlereingrenzung und -Behebung mitzugeben. Dies ist über das Exportieren von Alarmsignalen und Kennwerten von Prometheus an VSM umzusetzen.

1.3 Struktur

Diese Arbeit orientiert sich grob am Ausarbeitungsleitfaden „ARC42“. [5] Im nachfolgenden Kapitel 2 werden die Grundlagen erläutert, was genau unter Monitoring zu verstehen ist und wie es im NDR eingesetzt wird, sowie die betreffenden Protokolle kurz angeschnitten und schließlich die Entwicklungsumgebung vorgestellt. Außerdem wird erklärt, was unter Robustheit zu verstehen ist und wie diese geprüft werden kann. Im dritten Kapitel wird das Problem tiefergehend analysiert und anschließend im vierten Kapitel der gewählte Lösungsansatz aufgezeigt, der im fünften Kapitel verschiedenen Tests unterzogen wird. Die Diskussion in Kapitel 6 dient dem Aufzeigen der Vor- und Nachteile, der technischen Schulden und Risiken und den Versäumnissen in dieser Arbeit. Abgeschlossen wird mit einer Zusammenfassung und einem Ausblick auf mögliche weitere Fragestellungen, die Stoff für zukünftige Arbeiten liefern können.

1.4 Verwandte Arbeiten

Grundlage für diese Arbeit bildet die Monografie des Prometheus-Kernentwicklers Brian Brazil [2] mit Ergänzungen durch die Lehrbücher von Andrew S. Tanenbaum [26, 27] und Maarten van Steen. [28] Es wird außerdem Bezug genommen auf die Definitionen und Eigenschaften von „Soft States“ nach Lui et al. [17] Die Entwicklungsumgebung bedient sich neben der erstgenannten Lehrbücher außerdem auch der Arbeit von Soltesz et al. [25] und dem empfehlenswerten Kurzvortrag von Liz Rice. [23]
Aufgrund der thematischen Nähe und dem ähnlichen Protokollaufbau sei im weiteren Sinne auch auf die Bachelorarbeit von A. Güldener [4] an der HAW Hamburg verwiesen.

2 Grundlagen

Um die Schritte und Entscheidungen bei der Entwicklung dieser Arbeit verständlich zu machen, werden in diesem Kapitel zunächst wichtige Begriffe und Hintergründe erklärt. Es wird im Detail gezeigt, was Monitoring überhaupt ist, welche Zwecke es verfolgt, welche Aufgaben sowie Vor- und Nachteile die verwendeten Programme haben und wie der Datenaustausch vorgesehen ist. Weiterhin wird die Zielvorgabe für eine robuste Kommunikation festgesetzt und schließlich die Rahmenbedingungen bei der Entwicklung erläutert.

2.1 Monitoring

Obwohl die deutsche Sprache mächtige Werkzeuge wie beispielsweise die Schaffung neuer Wörter durch Zusammensetzung (Komposita) bietet, fehlen doch gelegentlich einfache Vokabeln, die im englischen Sprachraum einfache Unterscheidungen ermöglichen. Diese Diskrepanz äußert sich vor allem im Informatik-Sektor, in dem viele Fachbegriffe englischen Ursprungs sind. So ist es im Fachjargon üblich, zwischen „safety and security“ einen großen Unterschied auszumachen, während im deutschen beide Wörter mit „Sicherheit“ übersetzt werden. Ähnlich verhält es sich mit „monitoring“, dass im Gegensatz zu „surveillance“ nicht die eher negativ konnotierte Überwachung als Form von Aufsicht, Observation, Spitzelei oder gar Spionage bedeutet. Viel eher soll es als Überwachung in Form fortlaufender Kontrolle von Geräten und Systemen oder der Aufzeichnung von Werten verstanden werden. Um diesem Missverständnis nicht Vorschub zu leisten, wird daher auch im Verlauf dieser Arbeit vordergründig das Wort Monitoring verwendet und nur der Leserlichkeit halber an wenigen Stellen davon abgewichen. Was genau alles unter Monitoring fällt, welche Werte dabei aufgezeichnet und welche Geräte kontrolliert werden, kann unterschiedliche Dimensionen annehmen. Eine besonders spannende Form wäre beispielsweise das Monitoring geologisch aktiver Vulkane. [18]

Brian Brazil, einer der Hauptentwickler von Prometheus, unterteilt die Zwecke des Monitorings wie folgt:

> „*Alerting* Zu wissen, wann etwas schief läuft, ist normalerweise der wichtigste Grund, sich Monitoring zu wünschen. Man will, dass das Überwachungssystem einen Menschen dazuzieht, der einen Blick darauf wirft.
>
> *Debugging* Nachdem ein Mensch hinzugezogen wurde, muss nachgeforscht werden, um die Grundursache zu ermitteln und letztendlich das Problem zu lösen.
>
> *Trending* Alarmierung und Fehlerbehebung erfolgen in der Regel auf Zeitskalen in der Größenordnung von Minuten bis Stunden. Weniger dringend, aber dennoch nützlich ist die Möglichkeit, zu sehen, wie die Systeme genutzt werden und sich im Laufe der Zeit verändern. Trends können in Design-Entscheidungen und Prozesse wie der Kapazitätsplanung einfließen.
>
> *Plumbing* Wenn man nur einen Hammer hat, beginnt alles, wie ein Nagel auszusehen. Am Ende des Tages sind alle Monitoringsysteme Datenverarbeitungspipelines. Manchmal ist es bequemer, einen Teil des Monitoringsystems für einen anderen Zweck einzusetzen, als eine maßgeschneiderte Lösung zu bauen. Das ist streng genommen kein Monitoring, aber in der Praxis üblich und darum hier eingechlossen.“ [2, S. 5]

Die Wichtigkeit von Alarmen lässt sich am englischen Synonym „pages“ fest machen, das daher rührt, dass bei Alarmen die zuständigen Personen mittels Pager nicht nur informiert, sondern wenn nötig auch aus dem Schlaf geholt wurden. [1, S. 15, 56] Im Großen und Ganzen bedeutet Monitoring, dass man jedes Ereignis und die dazugehörigen Eigenschaften, also den Kontext, in einem beliebigen System aufzeichnen könnte. Was genau ein Event ist, ist dabei eher vage definiert, sodass es sehr viel Verschiedenes sein kann, solange es mit dem System assoziiert wird. Gleichwohl sind die Eigenschaften, die jedes Event ausmachen, sehr verschieden. Ein Wechsel vom User Space in den Kernel Space des Betriebssystems hat vielleicht nur den Systemaufruf im Quellcode des Programms als Eigenschaft, tritt jedoch häufig auf. Eine einzelne TCP-Verbindung besteht dagegen mindestens aus einem Fünf-Tupel: Der Protokollbezeichnung, der Quelladresse, dem Quellport, der Zieladresse und dem Zielport. Dafür ändern sich diese Eigenschaften aber über mehrere Sekunden oder sogar Minuten nicht. Andere Ereignisse können noch mehr Eigenschaften besitzen. Wollte man nun alles aufzeichnen, ist die zu speichernde

Datenmenge also offensichtlich sehr groß. Als Folge muss beim Monitoring auf bestimmte Daten verzichtet werden, was auf verschiedene Arten geschehen kann.
Zum einen durch die Beschränkung auf „profiling“, bei dem zwar möglichst alle Eigenschaften eines Events aufgezeichnet werden, dies jedoch zeitlich eingeschränkt ist. Ein Beispiel hier wäre das Mitschneiden eines kompletten Netzwerkanschlusses mittels „Wireshark“.
Im Gegensatz dazu steht das „tracing“, bei dem nur jede hundertste oder tausendste Verbindung aufgezeichnet wird.
Aus der Seefahrt bereits bekannt ist das „logging“, also das Anfertigen eines Logbuchs oder einer Log-Datei. Es zeichnet sich dadurch aus, dass einzelne Eigenschaften mitgeschrieben werden und ist bei vielen Betriebssystemen von vornherein schon mitgeliefert und eingeschaltet. Logging lässt sich noch weiter unterteilen in Transaktionslogs, Requestlogs, Anwendungslogs und Debuglogs. [2, s. 7-9] Die vierte Form des Monitoring bilden Metriken.

2.1.1 Metriken

Eine Metrik ist dabei zuerst einmal nur eine Maßzahl, also ein Kennwert, der erhoben wird. Daraus folgt, dass der Kontext eines Vorgangs größtenteils ignoriert wird. Stattdessen wird gezählt, wie häufig ein bestimmter Programmpfad aufgerufen wurde, wie viele Anfragen ein Webserver erhalten hat oder wie lange die Bearbeitung der Anfragen insgesamt gedauert hat. Die Informationen sind dann nur in der Bezeichnung der Metrik und dem jeweiligen Wert zu finden, was die zu speichernde Menge sehr klein macht. Die Anzahl an zu speichernden Metriken kann dabei im Fall von Prometheus bis zu 10.000 Werte je Prozess erreichen. [2, s. 10] Was genau anhand dieser Zahlen zu Monitoren ist, ist Gegenstand aktueller Diskussionen und vom Einsatzzweck abhängig. Es haben sich zwei grundlegende Vorgehensweisen herausgebildet:
Die RED-Methode konzentriert sich auf die Anzahl der Anfragen pro Zeiteinheit (englisch: rate), auf die Gesamtzahl der Fehler, die sich dabei ergeben (englisch: errors) und die Dauer, die die überwachten Prozesse für die Bearbeitung der Anfragen benötigen (englisch: duration). Der Vorteil ist, dass sich Anstiege in der Latenzzeit schnell bemerkbar machen und auf Netzwerkprobleme hinweisen könnten. Da Wartezeiten auf Endkundenseite so schnell erfasst werden, lohnt sich diese Art des Monitorings vor allem bei Online-Diensten, die von Kunden genutzt werden.
Der als intuitiver geltende USE-Ansatz bezieht sich dagegen auf die Auslastung der

Hardware (englisch: utilization), auf die Menge der noch ausstehenden Vorgänge (englisch: saturation) und die Anzahl der Fehlerfälle (englisch: errors). Der Gedanke dabei ist, dass knapp werdende Ressourcen rechtzeitig aufgestockt oder nicht mehr reagierende Systeme abgeschaltet werden können. Anwendung findet dies zum Beispiel in Rechenzentren und bei Stapelverarbeitungssystemen Aber auch das Monitoring im NDR wird so gehandhabt. [2, s. 55]

2.1.2 Zeitserien-Datenbank

Die Bezeichnung einer Metrik setzt sich aus verschiedenen, einzelnen Bezeichnern, den Labels, zusammen. So gibt der Wert des speziellen Labels __name__ den Namen an, unter denen die Metriken geführt werden. Weitere Labels können andere Namen und Werte haben. Sie werden entweder von der Applikation selbst, oder von der Monitoringsoftware vergeben. Erstere werden dabei als Instrumentations-Label bezeichnet und enthalten beispielsweise den Bezeichner oder die Netzwerkadresse der Maschine, auf der der Prozess ausgeführt wird. Letztere heißen dagegen Target-Label und werden durch die Konfiguration der Monitoringsoftware vergeben. Mit diesen können die Metriken nach Teams, übergeordneten Geographien oder administrativen Strukturen geordnet werden. Die Metrik ist also eindeutig identifizierbar über die Zusammensetzung des __name__-Labels und aller Instrumentations- und Target-Labels. Jeder dieser eindeutigen Identifier, also jede Metrik, beschreibt dabei eine eigene Zeitserie. Wie so etwas beispielhaft aussehen kann, zeigt die Abbildung 2.1.

Element	Value
prometheus_tsdb_head_samples_appended_total{instance="localhost:9090",job="prometheus"}	2532

Abbildung 2.1: Beispiel für eine Zeitserie. In geschweiften Klammern finden sich die Labels mit einem Wert, davor der Inhalt des Namen-Labels als Bezeichner. Ganz rechts steht der zuletzt erfasste Wert.

Patel et al. definieren eine Zeitserie als eine geordnete Menge von reellen Werten. [21, S. 370–371] Im Falle von Prometheus handelt es sich dabei um Tupel, da sowohl der Wert in Form einer 64-bit Fließkommazahl, als auch der Zeitpunkt der Erfassung in Sekunden seit Unix-Zeitrechnung gespeichert werden. Dennoch wird für Prometheus angegeben, durch Datenkompression nur sehr wenig Speicher zu benötigen:

> „The compression algorithm used can achieve 1.3 bytes per sample on real-world data.“ [2, S. 14]

Alle Zeitserien zusammengenommen bilden die Zeitseriendatenkbank bzw. Time-Series Database (TSDB). Die Zeitserien müssen dabei nicht alle die gleiche Kardinalität aufweisen. Ein großer Vorteil der Speicherung in einer Datenbank ist die Abfrage mittels geeigneter Query-Language. Prometheus bringt dabei seine ganz eigene Sprache, die turingvollständige PromQL, mit. Mit ihrer Hilfe können die letzten Einträge von Zeitserien, oder auch ganze Abschnitte abgerufen und mit arithmetischen Operationen bearbeitet werden.

2.1.3 Prometheus

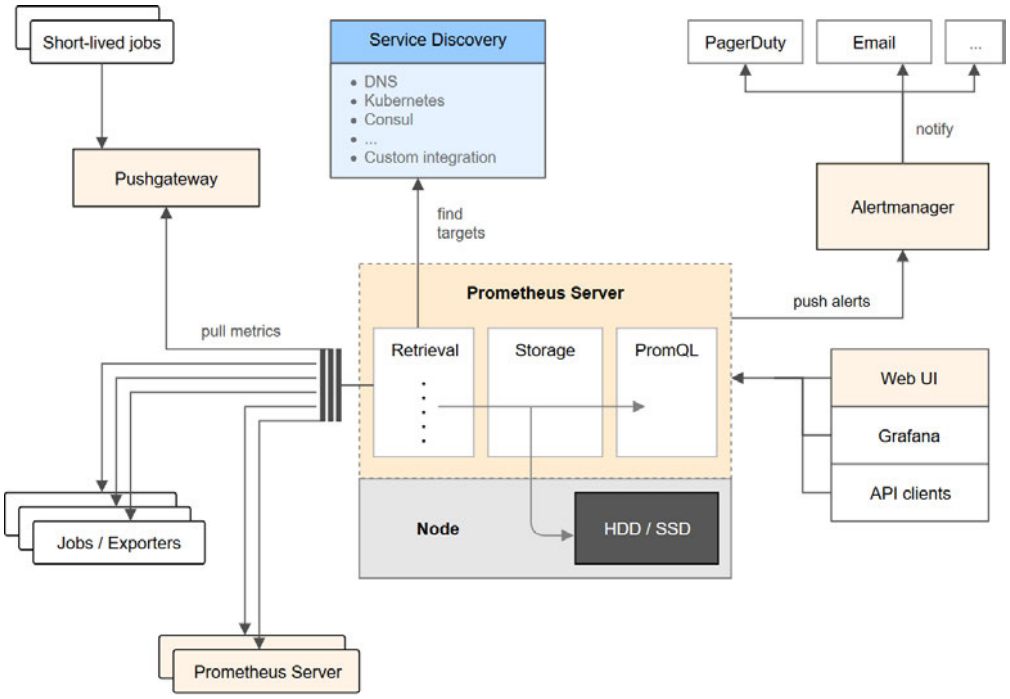

Abbildung 2.2: Schematischer Aufbau von Prometheus. [22]

Die Idee, IT-Systeme zu überwachen, ist nicht neu. Schon früh boten Betriebssysteme und Programme ihre eigenen Möglichkeiten, Fehlerfälle durch Log-Dateien zu erkennen und zu beheben. Aufgrund der zunehmenden Vernetzung war auch das Monitoring über die Grenzen des einzelnen Computers hinaus über das Netzwerk nur der nächste, logische Schritt.

Diese Arbeit hegt nicht den Anspruch eine umfassende Marktübersicht, über die Vielzahl verschiedener Netzwerk-Monitoringsysteme zu bieten, eine Erwähnung wert sind jedoch die seit 1999 entwickelte Softwares „Nagios“ und dessen Ableger „Icinga“ Beide erlangten große Bekanntheit und sind nach wie vor noch vielerorts im Einsatz und werden gepflegt. Sie erlaubten auch schon die Alarmierung eines Technikers bei Überschreitung von Grenzwerten oder Ausfall der Kommunikation mit einzelnen Maschinen. Der Ansatz, den die beiden Dienste folgten, war, dass die zu überwachenden Geräte sich entweder selbst bei der Monitoring-Software meldeten (passive Checks) oder aufgefordert wurden, Daten zu übermitteln (aktive Checks). Aktive Checks entsprechen einer „Pull-Strategie“, da die Daten vom Monitor eingeholt werden. Passive Checks gehören dagegen zur „Push-Strategie“, weil die Systeme selbstständig an den Monitor Meldung erstatten. Beide Vorgehensweisen haben ihre Vor- und Nachteile, doch im Falle von Prometheus wurde festgelegt, dass die Metriken mittel Pull-Strategie eingeholt werden. Dies wird als „scraping“ (Deuscht: kratzen) bezeichnet und geschieht in regelmäßigen, einstellbaren Abständen. Dazu bedient sich Prometheus einfach dem durch den Erfolg des Internets ohnehin weit verbreiteten Hypertext Transport Protocol (HTTP).

Wie in der Abbildung 2.3 zu sehen ist, listet die Datenbanksuchmaschine dbengines.com Prometheus unter allen Zeitseriendatenbanken auf Platz 3. Dies stellt keine Einschätzung der Fähigkeiten oder Reife der Software dar, sondern bezieht sich auf die Verbreitung dieses Datenbanktyps im Internet und wie häufig berichtet oder Fragen dazu gestellt werden. Es ist also nur ein grober Richtwert für die Bekanntheit und inwieweit bei Problemen mit Unterstützung gerechnet werden kann. Ersichtlich ist aber, dass es auch andere bekannte Zeitseriendatenbanken, also beispielsweise InfluxDB und Grpahite gibt, die eine Alternative darstellen können. Allen drei Systemen gemeinsam ist, dass sie zur Open-Source-Software gehören. Das bedeutet, dass die Quellcodes, in dem die Programme geschrieben sind, der Öffentlichkeit zur Verfügung gestellt werden und prinzipiell auch von jedem bearbeitet und erweitert werden können.

Eingangs wurde erwähnt, dass einer der Haupteinsatzzwecke von Monitoringsoftware auch die Alarmierung in dringenden Fällen ist. In der Komponentendarstellung von Prometheus in Abbildung 2.2 lässt sich sehen, dass zu diesem Zweck ein Alertmanager mitgeliefert wird. Dieser kommt mit einer Vielzahl separater Funktionen, um im Falle von zu hohen bzw. zu niedrigen Messwerten oder bei Nichterreichbarkeit von zu überwachenden Diensten bestimmte Personenkreise zu informieren. Dabei können die Alarme gesammelt, stummgeschaltet und ggf. auch verzögert übermittelt werden. Die Weitergabe erfolgt per E-Mail, in Chatsysteme oder an einen selbst erstellten http-Server.

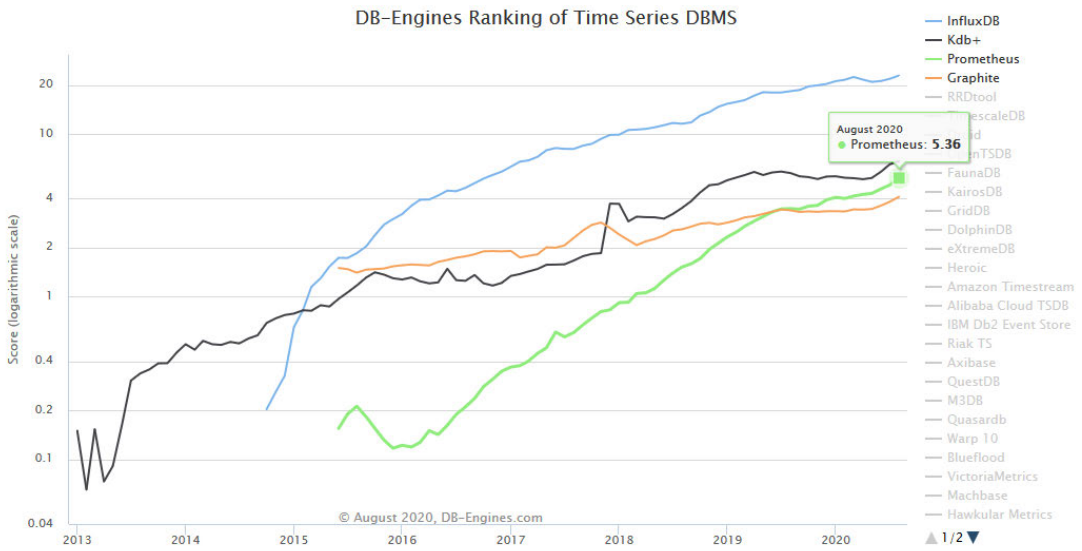

Abbildung 2.3: Gemäß der Datenbanksuchmaschine db-engines.com nimmt Prometheus zum Zeitpunkt dieser Arbeit den dritten Platz unter allen Zeitseriendatenbanken ein. [24]

2.1.4 Grafana

Prometheus bietet zwar die Möglichkeit, Querys direkt auf der Weboberfläche aufzurufen, dabei sind die grafischen Formatierungsmöglichkeiten jedoch begrenzt. Aus diesem Grund verweisen die Entwickler explizit auf die Nutzung der Software Grafana, die für die Aufbereitung und Wiedergabe verschiedener Datenbanken entwickelt wurde. [2, S. 14] Für die übersichtliche Darstellung von Sachverhalten sind die sogenannten Dashboards besonders hilfreich.

Abbildung 2.4: Beispiel eines Grafana-Dashboards, dass die Auslastung auf einem Server anzeigt. [13]

Um es aufzurufen wird nicht mehr benötigt, als ein Browser und der Netzwerkzugang zum Grafana-Server.

Abbildung 2.5: Ein VSM-Bedienpanel mit 33 farbig beleuchteten Knöpfen. Jeder Knopf beherbergt ein LC-Display. [15]

2.2 Virtuelles Studiomanagement

VSM ist ein Produkt der deutschen Aktiengesellschaft Lawo, die auf Regie- und Studioequipment in Medienunternehmen spezialisiert ist. Eigenen Angaben zufolge kann das System in mobilen Produktionsfahrzeugen, in Fernsehstudios und in Schalträumen für Fernsehen und Radio eingesetzt werden. Es basiert vordergründig auf herkömmlichen IP-Netzwerkverkehr, kann aber durch Erweiterungsgeräte auch Serielle Interfaces, General Purpose Input/Outputs (GPIOs) und Infrarotverbindungen ansteuern. Eine der wichtigsten Geräte, die via VSM gesteuert werden, ist hierbei die Kreuzschiene, an welcher die verschiedenen Eingänge diversen Ausgängen zugeordnet werden. Die Signallaufpfade werden also via Software geregelt. Dabei sitzen die Bediener in der Regel jedoch nicht vor einem PC, sondern haben ein physisches Panel (siehe Abbildung 2.5) vor sich, dessen Belegung dynamisch verändert und farbkodiert werden kann. VSM kommuniziert dabei mit verschiedenen Protokollen, insbesondere pro-bel SW-P-08, SNMP, Rot16, MIDI und Ember+. Nach Geschäftsangaben sollen über 150 Protokolle entweder nativ gesprochen oder am sogenannten Gadget-Server übersetzt werden können.

2.3 Verteilte Systeme

Tanenbaum und van Steen definieren ein verteiltes System als eine Sammlung von einzelnen Recheneinheiten, die auf den Benutzer erscheinen wie ein einzelnes, zusammenhängendes System. [28, S. 2] Für die vorhandenen Programme Prometheus und VSM trifft dies im Einzelnen bereits zu. Und auch das zu entwickelnde, kooperative Konzept soll

am Ende die Bedienung derart erleichtern, dass beispielsweise bei einem von Prometheus festgestellten Alarmzustand auch ein Techniker am VSM-Panel informiert werden soll. Ein mit der Alarmmeldung mitgeschickter Link oder gar eine Einbettung von Grafana, um automatisch auf die korrekte Datenlage zu verweisen, würde die Transparenz noch weiter erhöhen. Dies bedeutet, dass die folgenden Anforderungen an verteilte Systeme nach Tanenbaum und van Steen ebenfalls zu beachten sind:

Verteilungs-transparenz [28, S. 8]	Offenheit [28, S. 12–14]	Skalierbarkeit [28, S. 15]
Zugriff	Schnittstellendefinition	Größe
Ort	Interoperabilität	Geographie
Umzug	Zusammensetzbarkeit	Administration
Migration	Erweiterbarkeit	
Replikation	Trennung von Konzepten und Mechanismen	
Nebenläufigkeit		
Fehler		

Tabelle 2.1: Entwicklungsziele verteilter Systeme.

2.4 Protokolle

Um abgrenzen zu können, was mit Anwendungsprotokollen und mit „Schichten“ oder „TCP/IP-Stack“ gemeint ist, hilft ein Blick auf das OSI-Referenzmodell in Abbildung 2.6, das von der International Telecommunication Union (ITU) entwickelt wurde und als ISO-Standard veröffentlicht wurde.

Hieran lässt sich erkennen, dass die Kommunikation in verschiedene Ebenen, den Schichten, eingeteilt werden kann und die oberen Schichten auf den unteren aufbauen, indem sie dessen Struktur und Funktionen nutzen. Bildlich gesprochen ergibt sich damit ein Stapel an unterliegenden Protokollen, je nachdem, welches Protokoll im Detail betrachtet wird. Bei einem Protokoll der vierten Schicht, wie beispielsweise dem Transmission Control Protocol (TCP), ergibt sich damit ein Stapel (engl. „Stack“), der unter der vierten auch die Schichten drei bis eins beinhaltet. Die Unterteilung der Schichten kann für manche Betrachtungen zu grob sein, sodass Schichten noch feiner eingeteilt werden müssten. Sie

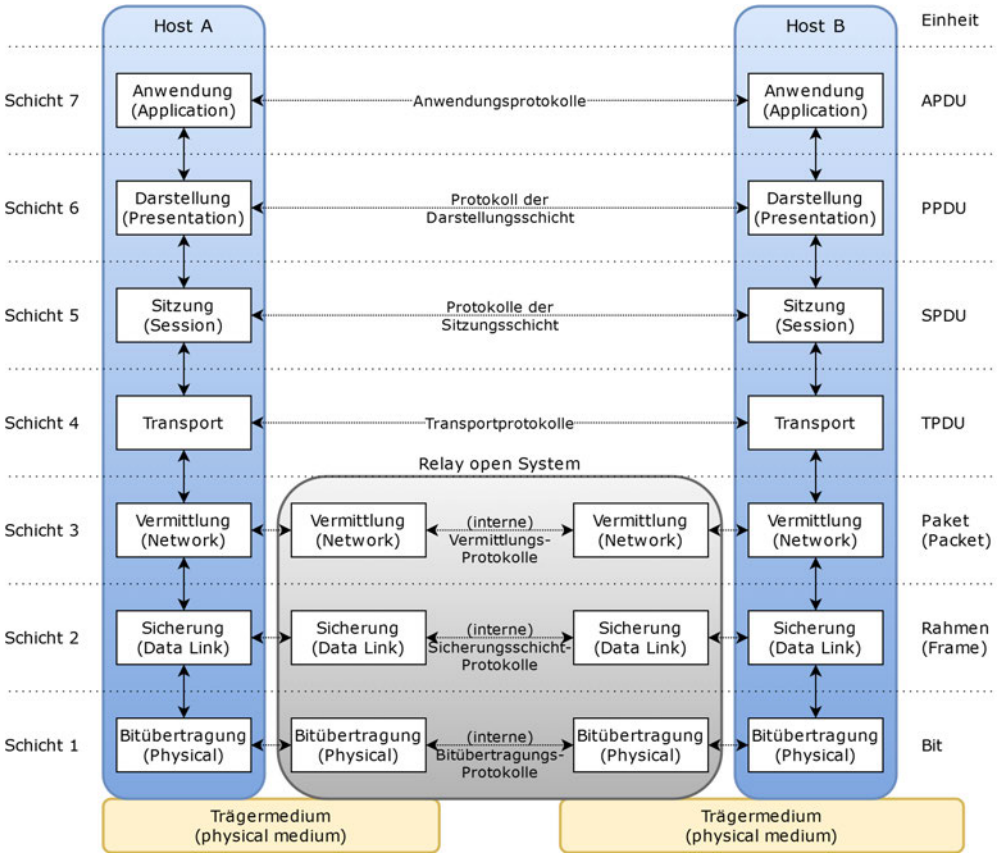

Abbildung 2.6: Das OSI-Referenzmodell nach Tanenbaum [27, S. 42] und ITU-T [10, S. 29]. [13]

kann allerdings auch zu fein sein, sodass Schichten einfach weggelassen oder zusammengefasst werden. So werden bei dem ebenfalls häufig genutzten TCP-Modell die erste Schicht mit der zweiten Schicht kombiniert und die Schichten fünf und sechs entweder ignoriert oder der siebten Schicht zugeordnet. [27, S. 41-48]

In den Abschnitten 2.1.3 und 2.2 wurde bereits angesprochen, mit welchen Protokollen Prometheus und VSM kommunizieren. Die beiden Programme haben keine Überschneidung der Anwendungsprotokolle, was die Entwicklung des hier vorgestellten Konzepts überhaupt erst nötig macht, die damit der Definition einer Middleware entspricht. [28]

Da beide Systeme aber mittels TCP/IP kommunizieren, wird nachfolgend verkürzt auf dieses Protokoll eingegangen. Anschließend werden die im weiteren Verlauf genutzten Anwendungsprotokolle kurz vorgestellt.

2.4.1 TCP/IP

Das Transmission Control Protocol (TCP) bildet zusammen mit dem Internet Protocol (IP) die Grundlage heutiger paketbasierter Netzwerke und des Internets. TCP wurde entwickelt, um verlässliche Datenströme über ein potenziell unzuverlässiges Netzwerk zu senden. Innerhalb von TCP wurden auch die sogenannten Ports definiert, die für darüberliegende Protokolle heute bekannt sind: z.B. Port 80 für HTTP. [27, S.552–554] In TCP werden die ankommenden Datenströme in Pakete aufgeteilt, die gelegentlich auch fälschlich als „Frames“ bezeichnet werden, obwohl die Bezeichnung für die Einheiten der zugrundeliegende Sicherungsschicht reserviert ist.[3, S. 10] Wie viele andere aktuelle Standards im Internet, wurde auch TCP definiert, indem beteiligte Forschungsgruppen Kommentare an Request for Comments (RFCs) vorgenommen haben, welche nach Veröffentlichung zum Standard wurden. Grundlegend für dieses Protokoll war dabei das RFC 793 [8] und viele weitere darauf aufbauende und ergänzende Schriften, die in RFC 4614 [9] gesammelt sind. [27, S. 553]

2.4.2 HTTP

Das Hypertext Transport Protocol (HTTP) ist ein Protokoll der Applikationsschicht und hat auch bei Menschen ohne technischen Hintergrund eine gewisse Bekanntheit erlangt, da es im Internet zum Zugriff auf Webseiten genutzt wird und daher am Anfang von URLs zu sehen ist. Auch mit den typischen Statuscodes, insbesondere *404 - Not found*, haben viele Internetnutzer vermutlich schon Bekanntschaft gemacht. Da es mit der im Jahr 2000 spezifizierten und inzwischen fast ausschließlich genutzten Erweiterung HTTPS auch automatisierte Verschlüsselung anbietet, hat es zum sicheren Datenaustausch beigetragen. Der Vorteil für den Nutzer ist vor allem, dass er sich in den meisten Fällen nicht um den Austausch der für die Verschlüsselung nötigen Schlüssel und Zertifikate kümmern muss. Ursprünglich war es jedoch nicht nur für die Übertragung von Webinhalten, sondern auch für verteilte Systeme gedacht, weswegen es neben den im Internet üblichen Funktionen „POST“ und „GET“ noch weitere anbietet. Die wichtigsten RFCs für dieses Protokoll sind RFC 2616(HTTP/1.1) [7] und RFC 7540(HTTP/2) [6].

2.4.3 SNMP

Um Geräte im Netzwerk verwalten zu können, wurde ab 1988 das Anwendungsprotokoll Simple Network Management Protocol (SNMP) entwickelt. Es erlaubt die Abfrage und die Veränderung von Kennwerten, die in einer Baumstruktur, der Management Information Base (MIB) abgelegt werden. Dazu sendet ein Server eine Anfrage an das Gerät, in dem es sich am Baum entlang orientiert. Dies ist insbesondere dann nötig, wenn die MIB nicht bekannt ist. Die zu verwaltenden Geräte können auch selbstständig Daten an das Netzwerk übergeben, ohne von einem Server dazu aufgefordert worden zu sein, was als „Trap“ bezeichnet wird. Obwohl die erste Version von SNMP keinerlei Sicherheitsmechanismen besaß und bei fehlender MIB kompliziert zu bedienen war, wurde und wird es von einer Vielzahl von Geräten unterstützt. Es entwickelte sich daher zum Standard für das Monitoring in Netzwerken mit Nagios (siehe 2.1.3), wobei die Datenpunkte in MySQL oder einer Round Robin Database (RRD) gespeichert wurden. Ebenso wurde es zum wichtigen Protokoll für Medienunternehmen, da SNMP auch zur Verwaltung von Kreuzschienen, Abspielgeräten und Mischpulten produktiv eingesetzt werden konnte. Während also modernere Monitoring-Anwendungen zunehmend HTTP statt SNMP verwenden, ist letzteres nach wie vor im Einsatz. Im Gegensatz zu HTTP benutzt SNMP aber nur in wenigen FällenTCP sondern setzt stattdessen auf das User Datagram Protocol (UDP). Dieses garantiert allerdings keine verlässliche Datenübertragung und wird daher in dieser Arbeit nicht weiter betrachtet. Als einen möglichen, inoffiziellen Nachfolger könnte man Ember+ betrachten, das sehr ähnlich aufgebaut ist.

2.4.4 Ember+

Im Gegensatz zu den vorherigen Protokollen ist der Name dieses Protokolls nur teilweise ein Akronym. Genau wie SNMP basiert auch Ember+ auf der Abstract Syntax Notation 1 (ASN.1) [11] und den darauf aufbauenden Basic Encoding Rules (BER) [12] die beide von der Internationalen Fernmeldeunion ITU spezifiziert wurden.

Allerdings handelt es sich nicht um ein einzelnes, untrennbares Protokoll, sondern ist in drei Teile geteilt: Glow, EmBER und S101, wie das Schema in Abbildung 2.7 darstellt.

Dabei ist Glow der Name des Schemas, dass die Datentypen definiert, EmBER die Codierung der Daten und S101 das Protokoll, dass diese Daten überträgt und dessen Integrität

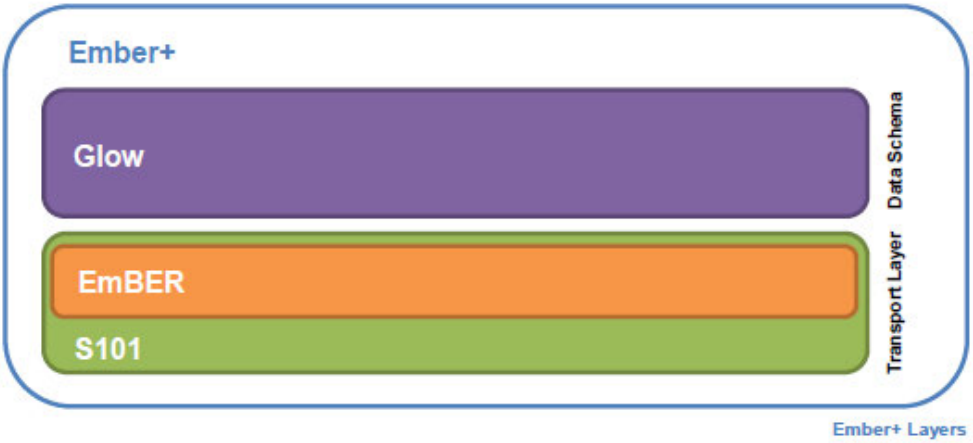

Abbildung 2.7: Das Ember+ Schichtenmodell, wie es in Referenzdokumentation dargestell ist. [14]

sicherstellt. In der Spezifikation wird der Vergleich mit einer herkömmlichen Webseite aufgestellt mit den Analogien, dass Glow in etwa wie XHTML wäre, EmBER der Codierung in XML ähnelt und S101 HTTP entspräche. [14, S. 9]

Eine weitere Gemeinsamkeit mit SNMP ist die Speicherung der Daten in einer Baumstruktur auf dem Endgerät, hier „Provider“ genannt, die von einem Server, dem „Consumer“, abgefragt und verändert werden können. Als Neuerung bietet Ember+ jedoch Service Discovery, also das automatisierte Finden von und Verbinden zu Geräten im Netzwerk. Außerdem erlaubt das Protokoll, dass Provider mathematische Formeln für numerische Datentypen angeben können, sodass beispielsweise Umrechnungen vereinfacht werden sollen. Diese beiden Besonderheiten sind jedoch nicht Gegenstand dieser Arbeit und werden daher nicht näher betrachtet.

2.5 Robustheit

Obwohl an vielen Stellen von robusten Systemen geschrieben wird, ist häufig nicht ganz klar, wie diese Eigenschaft definiert ist. Tannenbaum et al. erklären, dass Robustheit bedeutet, zu verhindern, dass eine Anwendung eine andere Anwendung zum Absturz bringt. [26, S. 839] Beyer et al. führen weiter aus, dass Robustheit die Resilienz gegen Fehler von Systemen ist, von denen das eigene System abhängt [1, S. 108, 111] und die Fähigkeit, offensichtliche Fehler abzufangen.:

> *Robust*
> Catches obvious errors [1, S. 372]

Schließlich spezifieren Lui et al.in ihrer Analyse über die Robustheit von Soft-State-Protocols wie folgt:

> By robust, we mean that the protocol's performance under a variety of network conditions is above an acceptable threshold, but need not be optimal. [17, S. 51]

Als „Die acht Irrtümer der verteilten Systeme“ wird eine Sammlung von Fehlannahmen bezeichnet, die Entwickler angeblich häufig über ein Netzwerk anstellen [28, S. 24]:

1. Das Netzwerk ist zuverlässig.
2. Die Latenzzeit ist null.
3. Die Bandbreite ist unendlich.
4. Das Netzwerk ist sicher.
5. Die Netztopologie änder sich nicht.
6. Es gibt nur einen Administrator.
7. Es gibt keine Transportkosten.
8. Das Netzwerk ist homogen.

Im Rahmen dieser Arbeit werden einige dieser Annahmen zur Vereinfachung tatsächlich als gegeben angesehen. Andere werden genutzt, um die Robustheit der Lösung zu prüfen.

Das sogenannte CAP-Theorem besagt, dass jedes verteilte System nur zwei der drei Eigenschaften *Konsistenz*, *Verfügbarkeit* und *Partitionierungstoleranz* bieten kann. [28, S. 461] Partitionierung bedeutet dabei die ungeplante Aufteilung des Netzes. Diese hat zur Folge, dass Teile des Systems sich in voneinander getrennten Subnetzen befinden und nicht mehr miteinander kommunizieren können.

> „During a partition, nodes might as well be on opposite sides of a wormhole: there is no way to know the state on the other side.“ [29]

In der Theorie heißt das folgendes: Kommt es zum Beispiel durch Netzwerkausfall zur Partitionierung und auf beiden Seiten arbeitet das verteilte System weiter und sichert neue Zustände, dann wissen die jeweils anderen Teile des Systems nichts von diesen - das verteilte System wird also inkonsistent.
Einfach keine neuen Zustände anzunehmen würde dagegen die Verfügbarkeit verletzen. Damit ist gezeigt, dass bei der Konzeptionierung des Systems auch Inkonsistenzen und Nichtverfügbarkeit zu berücksichtigen ist.

2.6 Entwicklungsumgebung

Zum Zeitpunkt der Erstellung dieser Arbeit wurden Installationsprogramme für Prometheus und Grafana für eine Vielzahl von Betriebssystemen, darunter auch Windows und Linux, angeboten. Lediglich VSM war nur für Windows verfügbar. Um die Bearbeitung einfacher zu gestalten und mobiles Arbeiten zu ermöglichen, war es nötig, sich auf ein mobiles Endgerät zu beschränken. Dies erforderte nicht nur den Einsatz von Virtualisierung, sondern vor allem auch die Abwägung verschiedener Ansätze für eine hohe Performance und Effizienz. Zur Auswahl standen dabei also verschiedene Kombinationen von Betriebssystemen in jeweils eigenen virtuellen Maschinen oder in Containern.

2.6.1 Virtualisierung

Virtualisierung ist ein weit gefasster Begriff und bezeichnet die Simulation einer virtuellen, also nicht realen, Hard- oder Software. Sie lässt sich grob in drei Stufen einteilen [28, S. 118]:

1. Eine Schnittstelle zwischen Hard- und Software, die Instruction Set Architecture (ISA), die privilegierte und allgemeine Maschinenbefehle bereitstellt.
2. Eine Schnittstelle, die aus Systemaufrufen des Betriebssystems besteht
3. Eine Schnittstelle, die aus Bibliotheksaufrufen besteht und eine Programmierschnittstelle bzw. ein Application Programming Interface (API) bildet.

Bezeichnend für erstere ist, dass sie noch unterhalb von Betriebssystemen angesiedelt ist. Auf Serverhardware kommt hier häufig die Software VMware zum Einsatz. Windows-Administratoren kennen dies eventuell als Hyper-V.

Ein Beispiel für Virtuelle Maschinen, die eben nicht ganz so Hardware-Nah sind, aber trotzdem die Installation eines eigenen Betriebssystems erlauben wäre für Privatnutzer kostenlose Software Virtual Box.
An dritter Stelle finden sich dann die besonders leichtgewichtigen Container.

2.6.2 Docker

Virtuelle Maschinen erlauben zwar die Kombination verschiedener Betriebssysteme mit ihren jeweils eigenen Kerneln auf einer einzelnen Maschine, erfordern dazu allerdings einen Monitor (VMM). Komplette Betriebssysteme vorzuhalten hat aber auch einen größeren Speicherbedarf sowie höhere Latenzen durch Übersetzungen zur Folge. Container dagegen teilen sich den Kernel und haben „nur“ einen eigenen User Mode. Sie bieten also eine kompakte und leichtgewichtige Möglichkeit der Isolation an, erfordern gleichwohl den Einsatz von Linux, während auf dem vom Autor genutzten Laptop aber Windows 10 vorinstalliert war. Dem kommt entgegen, dass für dieses Betriebssystem kurz vor Beginn dieser Arbeit die Unterstützung für leichtgewichtige virtuelle Maschinen mit vollständigem und nativem Linux-Kernel implementiert wurde - das WSL 2. Es verspricht dabei einen geringen Overhead und erlaubt trotzdem die Bereitstellung von Containern mit der Software Docker. Den Schematischen Aufbau kann man sich mit Blick auf die Abbildung 2.8 vergegenwärtigen:

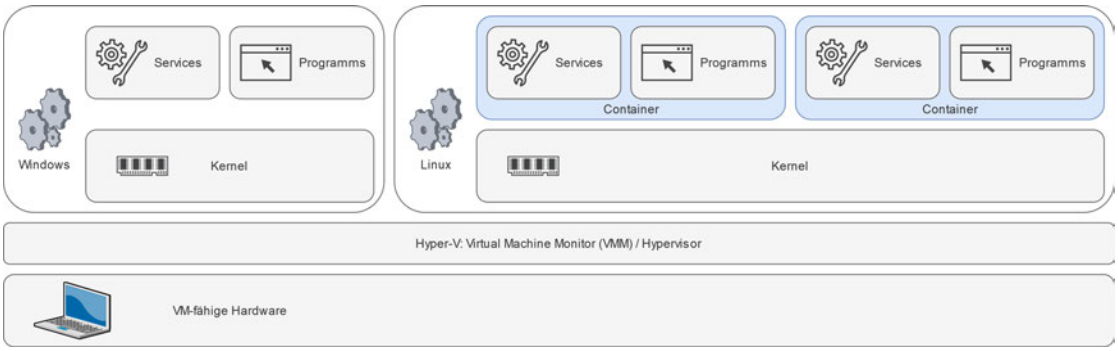

Abbildung 2.8: Struktur von Containern in Windows 10 mit WSL2. [13]

Ein Installationshilfe vom Hersteller findet sich leicht im Internet. [19]

Wie die Entwicklungsumgebung im Betrieb aussehen kann, zeigt die Abbildung 2.9:

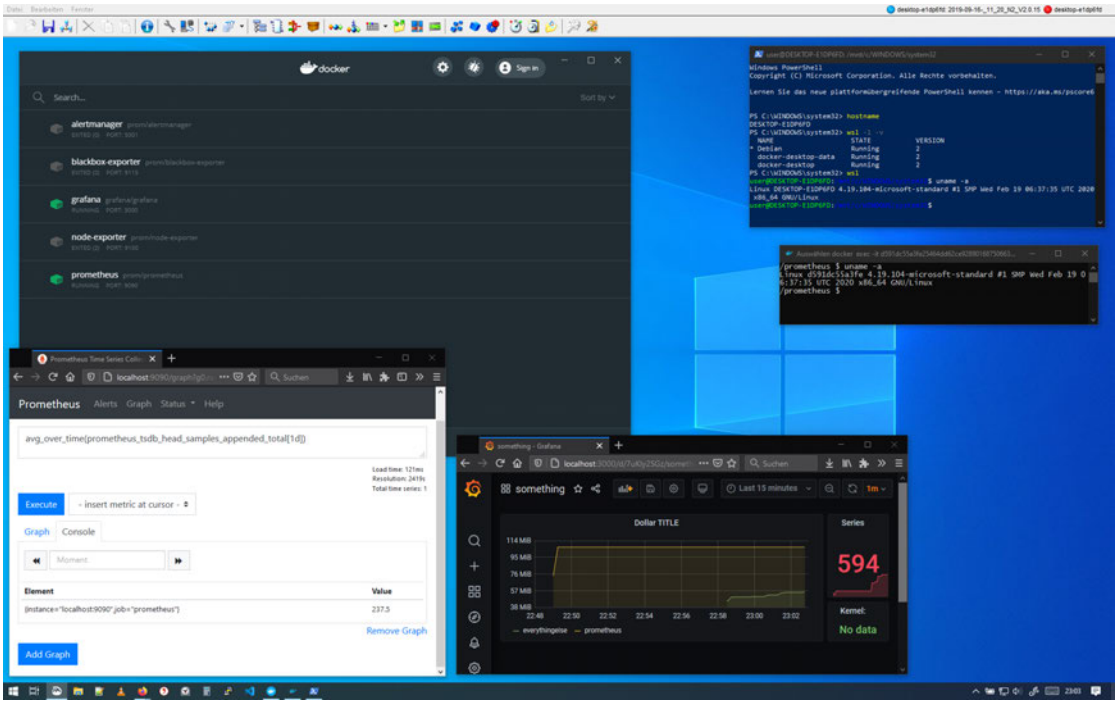

Abbildung 2.9: Entwicklungsumgebung auf einem Windows 10. Von oben nach unten: VSM-Studio, Docker-Dashboard (links), Powershell mit WSL2 (rechts), Bash in Container (rechts, darunter), Prometheus in Container (links), Grafana in Container, Task-Leiste. [13]

3 Problem- und Anforderungsanalyse

Die detaillierte Analyse und die Lösungsstrategien zur Erstellung der Software orientieren sich in Struktur und Vorgehensweise am ARC42-Modell. [5] Da in den beiden vorangegangenen Kapiteln schon die Einführungen und Ziele vorgestellt wurden, wird darauf nur noch verkürzt eingegangen. Hauptsächlich werden in diesem Kapitel daher die Randbedingungen sowie der technische und fachliche Kontext beleuchtet.

3.1 Randbedingungen

Zuvorderst wird festgehalten, welche Interessenvertreter an dem Projekt beteiligt oder davon betroffen sind:

Rolle	Erwartungshaltung
Prometheus-Administrator	Konfiguration von Prometheus und Scrape-Regeln wie gewohnt, Grafana-Dashboards in Alarmen exportierbar
VSM-Administrator	Konfiguration von VSM wie gewohnt, mehr Informationen abrufbar
Netzwerk-Administrator	Keine überlastung der Netzwerkkapazitäten, keine ungeprüfte Drittanbietersoftware im gesicherten Netz, keine Netzwerkverbindung in andere Netze oder in das Internet.
Schaltraum-Techniker	Erhält relevante Daten im Falle eines Alarms, wird nicht unnötig alarmiert

Tabelle 3.1: Stakeholder, die am Projekt beteiligt oder davon betroffen sind.

Zu beachten ist, dass Schaltraum-Techniker und VSM-Administrator die selbe Person sein können.

Anschließend werden die folgenden Bedingungen festgelegt:

ID Bezeichnung	Beschreibung
R1 Prometheus-Verwaltung	Das Management der Alarme und die Steuerung des Exports von VSM nach Grafana erfolgt an zentraler Stelle durch Prometheus-Einstellungen. Es soll möglichst wenig am Exporter selbst zu konfigurieren sein
R2 Protokoll	Es ist ein Protokoll zu verwenden, das von Prometheus nativ beherrscht wird. Ember+ ist ein Vorschlag, aber nicht zwingend vorgeschrieben.
R3 Schreibvorgänge	Es darf kein schreibender Vorgang, also keine Veränderung von Werten auf Geräten der Sendertechnik erfolgen
R4 Betriebssystem	Die Software muss auf einem Windows-Server lauffähig sein.
R5 Abhängigkeiten	Kommunikation findet ausschließlich innerhalb des gesicherten Netzes statt. Abhängigkeiten und Zugriffe in andere Netze oder das Internet dürfen nicht stattfinden.
R6 Skalierbarkeit	Die Software muss in der Lage sein, von zwei Prometheus-Instanzen gescraped zu werden.
R7 Timeout	Metriken müssen in weniger als einer Sekunde bereitgestellt werden können.

Tabelle 3.3: Randbedingungen.

3.2 Fachlicher Kontext

Auf dem Use-Case-Diagramm in Abbildung 3.1 lässt sich ausmachen, dass der größte Teil der Vorgänge zwischen den Programmen stattfindet und für den Benutzer transparent ist.

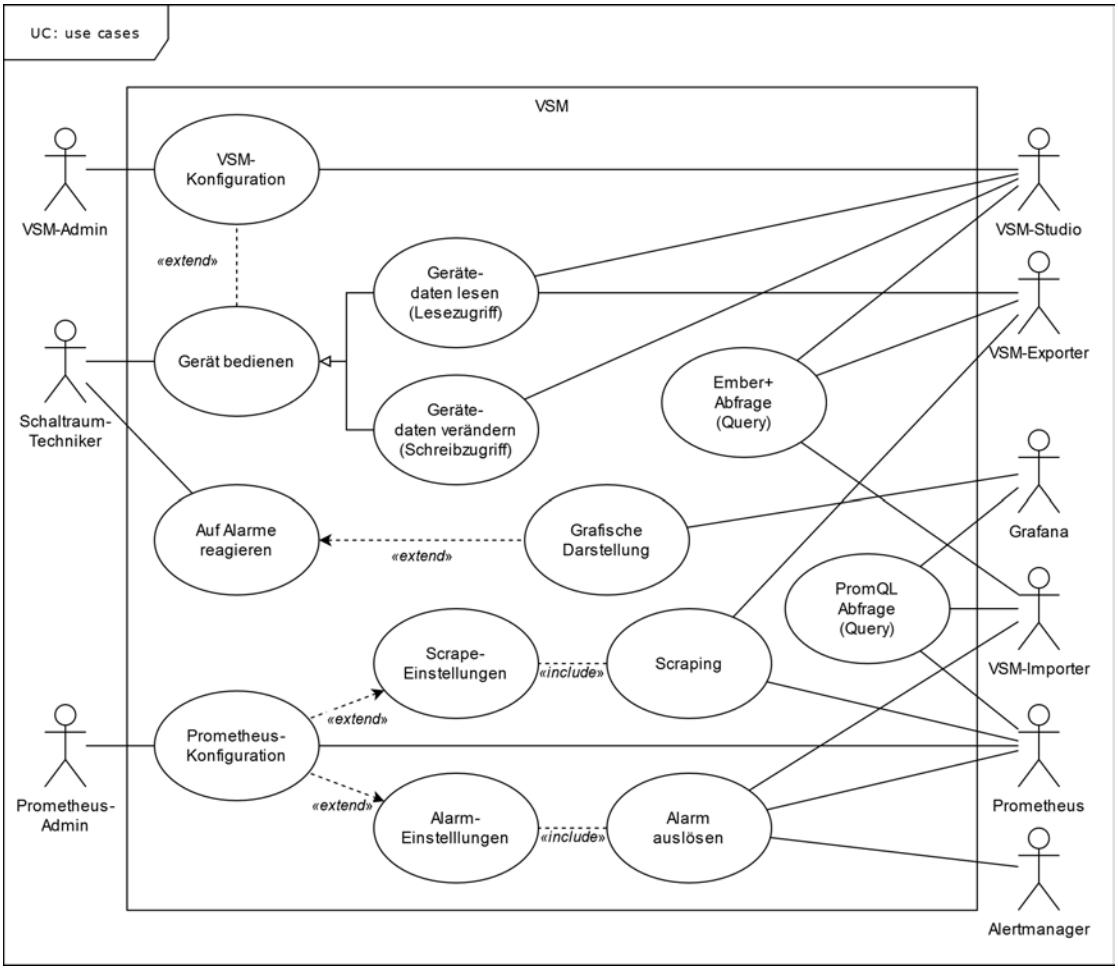

Abbildung 3.1: Use-Case-Diagramm für das gesamte System. [13]

Außerdem ist auffällig, dass der Stakeholder „Netzwerk-Administrator“ nicht aufgeführt ist, da die Erwartungen von dieser Seite passiver Natur sind und keinen eigenständigen Use-Case darstellen. Die für die Stakeholder relevanten Anwendungsfälle oder auch Use-Cases sind noch ein mal in der Tabelle 3.4 zusammengefasst:

Feld	Beschreibung
UC01	**VSM-Konfiguration**
Beschreibung	Administrator ändert Gerätezuweisungen, Abfragen oder sonstige Konfigurationen manuell. Kann auch UC03 beinhalten.
Akteure	VSM-Admin, VSM-Studio
Vorbedingungen	keine

Feld	Beschreibung
Ergebnis	geänderte Konfigurationseinstellungen, neue Geräte präsent oder alte Geräte entfernt, veränderte Gerätedaten
Ablauf	1. Der Adminstrator ändert Konfigurationen direkt über das Graphical User Interface (GUI) von VSM-Studio.
UC02	**Prometheus-Konfiguration**
Beschreibung	Administrator erweitert/kürzt/ändert Scraping-Regeln, Relabeling-Regeln, Alartering-Regeln
Akteure	Prometheus-Administrator, Prometheus, Alertmanager
Vorbedingungen	keine
Ergebnis	geänderte Konfigurationseinstellungen,
Ablauf	1. Der Administrator ändert die Konfiguration innerhalb der YML-Konfigurationsdatei 2. Der Administrator startet Prometheus bzw. den Alertmanager mit dem Signal SIGHUP oder mit einem HTTP POST an den Endpunkt /-/reload neu
UC03	**Geräte bedienen**
Beschreibung	Während des Sendebetriebes wird eine Vielzahl von Geräten über das Netzwerk gesteuert. Dabei werden Werte wie beispielsweise die Position eines Schiebereglers ausgelesen und auch verändert. Dieser Use-Case ist eine Generalisierung von UC04 und UC05. Die Datenübertragung kann mit einem der vielen unterstützen Protokolle erfolgen, also auch mit Ember+, UC08.
Akteure	Techniker, VSM
Vorbedingungen	keine
Ergebnis	siehe UC04 oder UC05
Ablauf	1. Der Techniker bedient Endgeräte, um eine Sendung zu Produzieren. 2. Bei jeder Aktion (Knopfdruck, Drehregler drehen, Fader aufziehen etc.) werden Daten gelesen und/oder geschrieben.
UC04	**Gerätedaten lesen**
Beschreibung	Spezialisierung von UC03, bei dem Daten nur gelesen, aber nicht geschrieben werden.
Akteure	siehe UC03

Feld	Beschreibung
Vorbedingungen	keine
Ergebnis	Techniker kennt Werte.
Ablauf	siehe UC03
UC05	**Gerätedaten schreiben**
Beschreibung	Spezialisierung von UC03, bei dem Daten nur geschrieben, aber nicht gelesen werden. Tritt in der Regel gemeinsam mit UC04 auf.
Akteure	siehe UC03
Vorbedingungen	Datenwert ist veränderbar
Ergebnis	Datum enthält neuen Wert
Ablauf	siehe UC03
UC06	**Grafische Darstellung**
Beschreibung	Aufruf eines Grafana-Dashboards, grafische Darstellung von Datenwerten im Browser, als HTML-Frame bzw. Embed
Akteure	Grafana
Auslöser	Seite wird aufgerufen, Alarm wird ausgelöst
Vorbedingungen	Dashboard existiert, Prometheus ist erreichbar
Ergebnis	Techniker kennt Werte, kann auf Alarme reagieren
Ablauf	1. Grafana-Webseite wird via Browser oder als HTML-Frame bzw. Embed aufgerufen. 2. PromQL-Abfrage von Metriken gegenüber Prometheus (siehe UC09) 3. Daten werden grafisch aufbereitet dargestellt.
Alternativablauf	Anstelle von Grafana kann auch der Browser des Nutzers den PromQL-Aufruf ausführen. Dies ist für diese Arbeit allerdings unerheblich und wird nachfolgend ignoriert.
UC07	**Scraping**
Beschreibung	Prometheus führt in regelmäßigen Abständen ein Scrape durch, sammelt Metriken ein und speichert sie. Eines der Ziele ist der Exporter, der die Daten von den Endgeräten erhält.
Akteure	Prometheus, VSM-Exporter, VSM
Auslöser	Timer, in UC02 definiert
Vorbedingungen	Scrape wurde konfiguriert, Exporter reagiert auf HTTP-Anfragen, Exporter ist konfiguriert, Zielgeräte sind erreichbar siehe UC08

Feld	Beschreibung
Ergebnis	Die Gerätedaten sind in Prometheus gespeichert
Ablauf	1. Prometheus öffnet Anfrage via HTTP GET an Exporter. 2. Der Exporter erhält Zielinformationen via HTTP GET 3. Der Exporter kontaktiert die Ziele in Ember+ und speichert angefragte Daten ab. 4. Der Exporter sendet die Daten als HTTP Antwort. 5. Prometheus speichert die eingegangenen Daten ab.
UC08	**Ember+-Abfrage (Query)**
Beschreibung	Die Kommunikation zwischen den Endgeräten innerhalb des VSM-Systems kann via Ember+ erfolgen. Der VSM-Exporter und der -Importer kommunizieren via Ember+ mit VSM-Studio.
Akteure	VSM-Exporter, VSM-Studio, VSM-Importer, Endgerät
Auslöser	Prometheus-Scrape, Keep-Alive von Ember+-Consumer
Vorbedingungen	keine
Ergebnis	Daten wurden übermittelt.
Ablauf	1. VSM-Consumer stellt Verbindung mit VSM-Provider via S101 her 2. Der angeforderte Datensatz wird komplett oder in Teilen übertragen.
Alternativablauf	1. VSM-Consumer stellt Verbindung mit VSM-Provider via S101 her 2. Der angeforderte Datensatz wird komplett oder in Teilen übertragen. 3. Das zu ändernde Datum wird geändert und zurückübertragen.
UC09	**PromQL-Abfrage (Query)**
Beschreibung	Die in Prometheus gespeicherten Zeitserien können jederzeit mittels PromQL-Query an die API angefragt werden.
Akteure	Prometheus, Grafana, VSM-Importer
Auslöser	Aufruf von Grafana, Direktaufruf durch Nutzer, Aufruf durch VSM-Importer
Vorbedingungen	Prometheus läuft und akzeptiert Anfragen
Ergebnis	Rückgabewert gemäß Query.
Ablauf	1. Anfragendes Programm öffnet HTTP GET an Prometheus 2. Prometheus sendet angefragte Daten als HTTP Antwort

Feld	Beschreibung
UC10	**Alarm auslösen**
Beschreibung	Eine der wichtigsten Aufgaben des Systems ist, bei Überschreitung (bzw. Unterschreitung) eines Grenzwerts, eine Person zu informieren.
Akteure	Prometheus, Alertmanager, VSM-Importeur
Auslöser	Grenzwertüberschreitung
Vorbedingungen	Alarmierungsregeln wurden definiert
Ergebnis	Alarmierung ist eingegangen und Techniker wurde informiert
Ablauf	1. Prometheus registriert, dass ein Grenzwert überschritten wurde, der Alarm wird von inaktiv auf „pending“ gesetzt und an Alertmanager übergeben 2. Der Grenzwertfür bleibt festgelegte Zeit überschritten, der Alarm wird auf „firing“ gesetzt und an den Alertmanager übergeben. 3. Alertmanager gruppiert mit Alarmen gleicher Kategorie und sendet an Verbreitungswege, wie Mail und Webhook des VSM-Importers. 4. Der VSM-Importer ändert voreingestellten Wert für ein Endgerät (z.B. rote Lampe oder Warnmeldung auf Bildschirm inkl. Grafana-Link).
UC11	**Auf Alarme reagieren**
Beschreibung	Wurde ein Techniker informiert, ist es seine Aufgabe, sich um die Fehlerbehebung zu kümmern.
Akteure	Techniker, VSM, Grafana
Auslöser	Alarm ausgelöst (UC10)
Vorbedingungen	Alarm ausgelöst
Ergebnis	Alarm-Ursache beseitigt
Ablauf	1. Der Techniker nutzt die angebotenen Informationen von VSM, Prometheus und Grafana, um den Fehler zu identifizieren. 2. Der Techniker behebt das Problem.

Tabelle 3.4: Use-Cases im Detail.

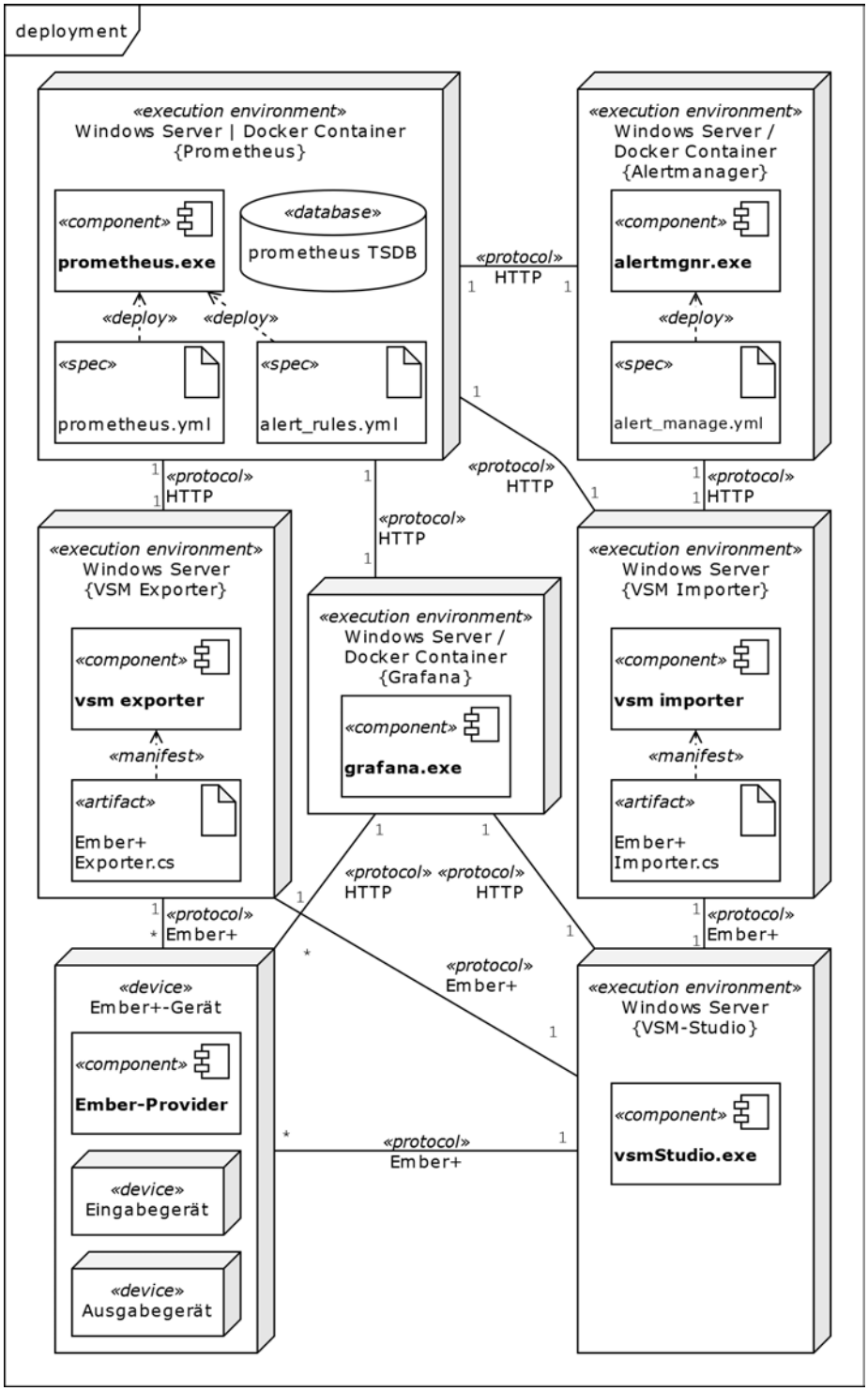

Abbildung 3.2: Deployment der Konfigurationen und Server mit den jeweiligen Kardinalitäten, Schnittstellen und Protokollen. [13]

3.3 Technischer Kontext

Das Deployment-Diagramm in Abbildung 3.2 zeigt die Verbindungspunkte der verschiedenen Systeme, sowie die genutzten Protokolle vereinfacht an. Die verschiedenen Anwendungsumgebungen können ggf. auch auf der selben Hardware eingesetzt werden, was aus Gründen der Einfachheit nicht in die Abbildung aufgenommen wurde. Die VSM-Exporter und Importer, die die Übersetzung von Ember+ in HTTP und zurück übernehmen, finden sich Mittig links und rechts.

Anhand dieser Ansicht lässt sich nun eine Liste an Netzwerkfunktionen definieren.

1. Exporter bietet HTTP-Endpunkt an und reagiert auf Scrape-Anfragen.
2. Exporter öffnet Ember+-Verbindung mit Ember+-Provider.
3. Importer agiert als Ember+-Provider und reagiert auf Verbindungen.
4. Importer öffnet Ember+-Verbindung mit VSM-Studio
5. Importer bietet HTTP-Endpunkt an und reagiert auf Nachrichten des Alertmanagers.
6. Importer öffnet HTTP-Verbindung an Prometheus und übermittelt PromQL-Query.

Diese Netzwerkfunktionen werden mit der Matrix in Tabelle 3.5 den jeweiligen Use-Cases zugeteilt.

ID	Beschreibung	F1	F2	F3	F4	F5	F6
UC01	VSM-Konfiguration						X
UC02	Prometheus-Konfiguration	X					
UC03	Geräte bedienen						
UC04	Gerätedaten lesen	O	X				
UC05	Gerätedaten schreiben						
UC06	Grafische Darstellung						
UC07	Scraping	X	O				
UC08	Ember+-Abfrage (Query)		X	X			
UC09	PromQL-Abfrage (Query)						X
UC10	Alarm auslösen					X	
UC11	Auf Alarme reagieren				X	X	

Tabelle 3.5: Zuweisungsmatrix - direkte Abhängigkeiten sind mit einem X markiert, indirekte mit einem O.

4 Lösungsstrategie

Obwohl Prometheus ausschließlich in der Programmiersprache Go geschrieben wurde, bieten die Entwickler auch Bibliotheken zum Monitoring eigener Applikationen in den Programmiersprachen Java, Python und Ruby an. Gleichzeitig bietet Lawo eine Referenzimplementation von Ember+ und dazugehörige Bibliotheken an, die jedoch in C und C++ geschrieben sind. Hier kommt die Fähigkeit der von Google entwickelten Programmiersprache Go zum Tragen, auch in C geschriebene Bibliotheken benutzen zu können. Dazu ist die Erweiterung CGO zu nutzen aber gleichzeitig auch auf einige Features zu verzichten. Obwohl also als erster Implementationsansatz nahe lag, die direkt angebotenen Programmroutinen zu nutzen, erwiesen sich die Hürden und die Lernkurve als Ausschlusskriterium.

Neben der nativen Implementation bietet Lawo allerdings auch in einem separaten Repository eine Implementation in C# an, das ähnlich wie Java eine eigene Laufzeitumgebung benötigt. Auch für Prometheus existieren Bibliotheken in ca. 16 anderen Programmiersprachen, die durch Programmierer der Open-Source-Gemeinde zur Verfügung gestellt wurden. Unter diesen ist auch eine Implementation in C#, sodass der zweitnächste Ansatz ist, den Exporter und den Importer in dieser Programmiersprache zu schreiben.

Im vorigen Kapitel war bereits ersichtlich, dass die Umsetzung in zwei getrennten Programmen, dem Exporter und dem Importer erfolgen soll. Der Grund dafür liegt in dem Entwicklungsprinzip Keep it simple, stupid (KISS), das dafür steht, möglichst einfache Lösungen zu finden und so den Arbeitsaufwand aber vor allem auch das Fehlerrisiko zu senken.

4.1 Laufzeitsicht

Mit Hilfe der Sequenzdiagramme in Abbildung 4.1 soll dargestellt werden, wie der Ablauf beim Scraping und bei der Ausgabe von Alarmen aussieht.

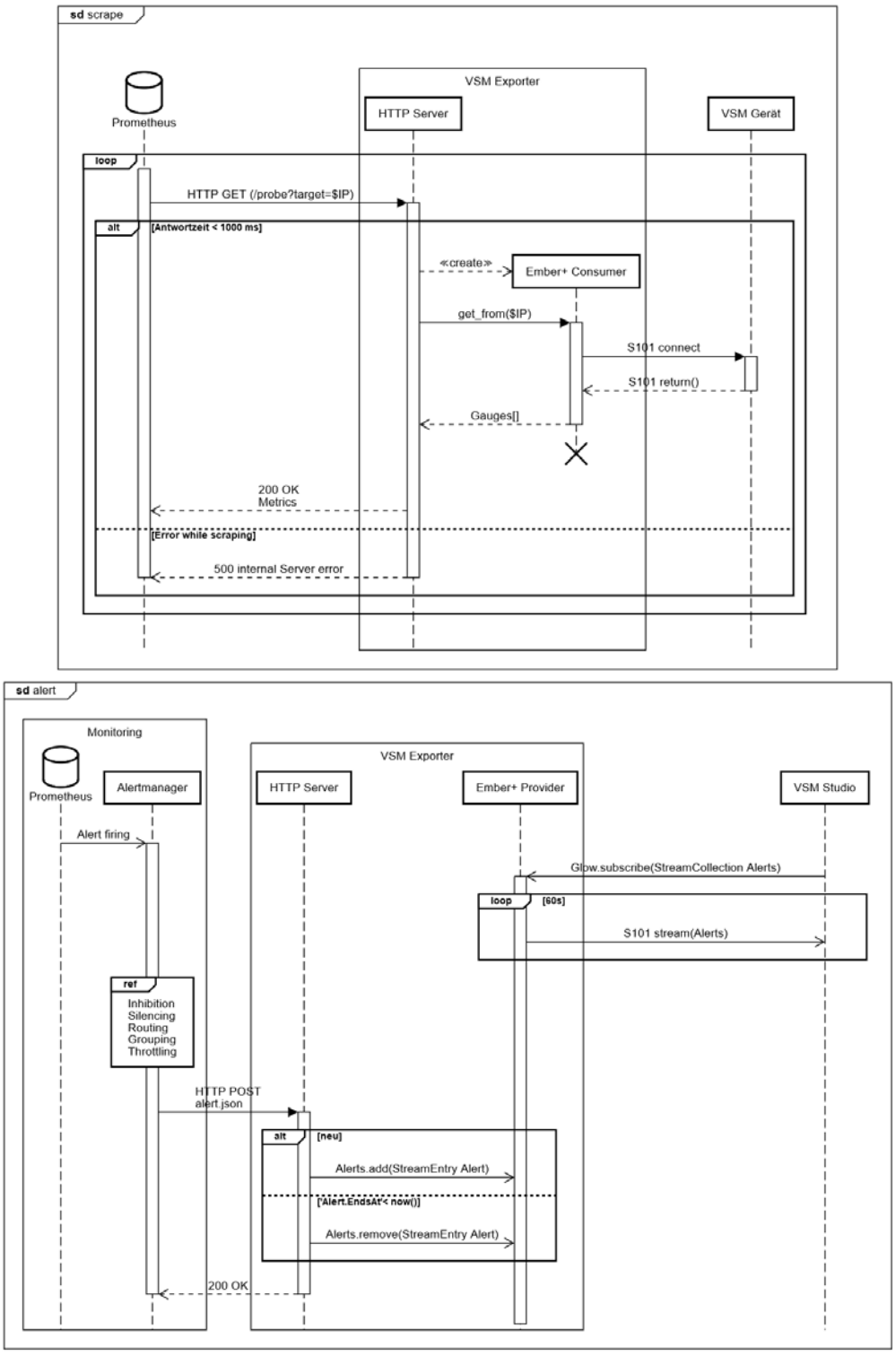

Abbildung 4.1: Ablauf eines Scrape-Vorgangs und einer Alarmierung. [13]

Es lässt sich erkennen, dass ausschließlich mit synchronen Funktionsaufrufen gearbeitet werden kann, beim VSM-Importer muss jedoch der gleichzeitige Zugriff auf die Liste der Alarme durch den Provider abgefangen werden. Dies kann beispielsweise mit einem Mutex geschehen. Synchrone Aufrufe zu verwenden hat dabei allerdings den Nachteil, dass in den Zeiträumen keine anderen Aufgaben wie beispielsweise andere Scrape-Anfragen durch Prometheus bearbeitet werden können. Da ein Scrape bis zu 1000 ms bis zum Abbruch durch Timeout in Anspruch nehmen darf und die Standardeinstellung für Scrape-Durchläufe 15 Sekunden beträgt, würden im schlechtesten Falle nicht mehr als 15 verschiedene Ziele abgefragt werden können. Für einen produktiven Einsatz ist dieser Ansatz also nicht geeignet.

5 Experimente

Wie in Kapitel 2 erläutert wurde, bedeutet Robustheit zum einen, nicht von Fehlern anderer Systeme abhängig zu sein und zum anderen, trotz Fehlfunktionen im Netzwerk seine Aufgaben zu erfüllen. Um ersteres zu testen benötigt es daher zusätzliche Software, von der bekannt ist, dass und in welchem Rahmen sie fehlerhaft funktioniert. Ein bekanntes Testverfahren dafür ist das sogenannte „Fuzzing“ oder auch Fuzz-Testing, das erstmals 1988 auf Computerprogramme angewendet wurde. [20, S. 2] Bei dem Verfahren werden durch einen Generator Eingaben generiert und an das zu testende Programm übergeben. In der einfachsten Form handelt es sich bei den Eingaben um zufällige Zeichenfolgen, in spezialisierter Form lassen sich die Eingaben auch dem Programm anpassen. Da dies Kenntnisse über den inneren Aufbau von Programmen voraussetzt, wird es als „White-Box-Testing“ bezeichnet, während die triviale Generierung keinerlei Wissen verlangt und daher „Black-Box-Testing“ genannt wird. Beide Begriffe sind auch anderen Testverfahren inhärent, bei denen eventuell andere Ziele verfolgt werden. Eine bekannte Software, mit der derartige Tests ausgeführt werden, ist das Open-Source-Werkzeug *ClusterFuzz*.
Das Testen von Netzwerkausfällen andererseits kann verschiedene Ausmaße annehmen. Am einen Ende steht dabei die vollständige Partition, deren Ergebnis gemäß des im Unterpunkt 2.5 vorgestellten CAP-Theorems trivial ist: Wenn die grundsätzlich nötige Netzwerkverbindung nicht vorhanden ist, kann das System nicht funktionieren. Von Bedeutung wäre dann lediglich die Betrachtung, ob es erkennt, dass die Verbindung wieder da ist und wie lange dies dauert. Andererseits, ist es auch nicht nötig, ein ideal funktionierendes Netzwerk zu betrachten, bei dem alle „Acht Irrtümer der verteilten Systeme“ keine Irrtümer wären. Interessant wird es dagegen, wenn die Betrachtung sich von den Extrempunkten wegbewegt, wo also die Kennwerte für Jitter, Latenz, Paketverlust und Datenstau (engl. congestion) größer als Null aber deutlich kleiner als Unendlich sind. Um diese Szenarien künstlich herbeizuführen wird also ein Netzwerk und entsprechende Software benötigt. Die unter 2.6 vorgestellte Umgebung erlaubt dabei die Simulation eines Netzwerkes und die zum Linux Kernel gehörenden Werkzeuge Network Emulator (NetEm) und Traffic Control (TC).

5.1 Aufbau

Die Container, in denen die verwendeten Programme ausgeführt werden, enthalten in der Regel nur einen sehr beschränkten Satz an Kernelfunktionen, um den Speicherverbrauch gering zu halten. Es wird daher auf das Programm „BusyBox“ zurückgegriffen, dessen Entwickler es auch gerne als das „Schweizer Taschenmesser für embedded-Linux“ bezeichnen. Diese Bezeichnung legt nahe, dass es ursprünglich für Hardware mit beschränkten Ressourcen gedacht war und deswegen darauf ausgelegt ist, sparsam mit diesen umzugehen. Der Nachteil für den Testaufbau ist dabei, dass TC nur eingeschränkt und NetEm gar nicht zur Verfügung steht. Abhilfe schafft hierbei erneut der Modulare Aufbau der virtuellen Umgebungen. So ermöglicht die Software „Pumba“ [16], als Docker-Container die simulierte Netzwerkverbindung zwischen anderen Containern zu stören oder ausgewählte Container gleich ganz auszuschalten. Letzteres ist, ähnlich wie das Fuzz-Testing, zufallsbasiert und zielt darauf ab, der Orchestrierung von Containern einem andauernden Härtetest zu unterziehen. Dies wird auch als Chaos-Testing bezeichnet.

Folgende Tests werden dabei bei dem in dieser Arbeit entwickelten System durchgeführt und das Ergebnis geprüft:

1. Simulation einer Netzwerkpartition: kein Scrape
2. Simulation einer Netzwerkpartition: Target nicht erreichbar
3. Simulation von Packet-Loss: 10%, 50%, 90%
4. Simulation von Latenz: 50ms, 150ms, 500ms

5.2 Durchführung

Repräsentativ für das System wird dabei der VSM-Exporter genauer geprüft. Die Implementation der im Kapitel 4 vorgestellten Lösung erfüllt den Status eines Minimum Viable Product (MVP) und ist daher als fragil anzusehen. Um die Tests 01 und 02 auszuführen, genügt es bereits, den jeweiligen Kommunikationspartner auszuschalten, ein Eingriff in die Netzwerktopologie ist nicht nötig. Bei Tests haben dabei die folgenden Ergebnisse gezeigt:

Test	Beschreibung
01	Test bestanden
02	Absturz: unhandled Exception
03	Test nicht ausführbar
04	Test nicht ausführbar

Tabelle 5.1: Testergebnisse

Wie zu erkennen ist, waren die Tests 03 und 04 nicht ausführbar. Grund dafür war die fehlgeschlagene Portierungsmöglichkeit in einen Docker-Container, da es nicht gelang, ein lauffähiges Docker-Image bauen zu lassen. Der manuelle Build-Prozess schlug mit einer nicht behebbaren Fehlermeldung fehl. Die automatisierte Erstellung bot hingegen keine Optionen an, die Ports und Adressen für die Verbindung mit dem Ember+ Provider festzulegen. Als Folge war es nicht möglich, NetEm bzw. Pumba einzusetzen. Test 02 hingegen ist insofern erfolgreich gewesen, als dass er gezeigt hat, dass vom Entwickler mindestens ein Fehlerfallfall nicht bedacht wurde.

6 Evaluation

Obwohl die Tests zeigen konnten, dass die Software funktioniert, haben die Testergebnisse gezeigt, dass grundlegende Konzeptionierungsfehler vorliegen. Diese sorgten dafür, dass die Software nicht wie erwartet reagiert hat, sobald die Verbindung ausgefallen ist. Anhand dieses Beispiels wird deutlich, wie wichtig ausreichendes Testen bei der Entwicklung von Software ist. Ein möglicher Ansatz der Fortführung wäre, Test-Driven-Development einzusetzen, bei denen das Testen von Software derart im Vordergrund steht, dass zuerst eine funktionierende Testumgebung entwickelt wird, ehe das Produkt in Angriff genommen wird. Für den produktiven Einsatz ist die vorgestellte Software in der Form noch nicht nutzbar.

Es ist nie mit völliger Sicherheit auszuschließen, dass noch andere Fehler existieren können. Diese zu finden erfordert den Einsatz weiterer Testfälle, wie z.B. WhiteBox-Tests. Darüber hinaus ist es dringend nötig, auch den für die Alarmierung zuständigen VSM-Importer ausgiebig zu testen, denn beide Programme sind ohne das jeweilig andere nur von eingeschränktem Nutzen. Die Testumgebung zu optimieren und die konsequente Anwendung von Test-Driven-Development, kann dabei helfen, zu einem robusten Produkt zu gelangen.

7 Zusammenfassung

Ziel dieser Bachelorarbeit war, einen Überblick über die Grundlagen des Monitorings im Allgemeinen und die Nutzung von Prometheus sowie VSM im Speziellen zu gewinnen, um anschließend diese beiden Systeme robust und kooperativ zu betreiben. Dabei wurde auf die Grundlagen zum Verständnis auf die besondere Situation eingegangen und welche Bedeutung Robustheit im Umfeld eines Fernsehstudios hat. Weiterhin wurde vorgestellt, wie der Aufbau des Systems angedacht war und wie die Implementation anhand der Testversuche als ungenügend erkannt wurde.

7.1 Fazit

In der Bachelorarbeit wird deutlich, dass Verteilte Systeme einen hohen Anspruch an die Fähigkeiten der Entwickler stellen. Die Entwicklungsziele für sich allein genommen bedeuten schon einen oftmals nicht umsetzbaren Anspruch, bei dem die einzelnen Ziele gegeneinander abgewogen werden müssen. Ein sicherer Umgang mit den Entwicklungswerkzeugen und den Programmiersprachen ist deshalb eine wichtige Voraussetzung. Rückschläge wie fehlende Bibliotheken, fremde Programmiersprachen und neuartige Werkzeuge müssen dabei mit Mehrarbeit und Planung im Zeitkontingent berücksichtigt werden. Als Folge zeigt das vorgestellte Minimalsystem zwar die grundlegenden Funktionen, erfüllt aber nicht die gesetzten Ansprüche.

7.2 Ausblick

Obgleich der Versuch einer robusten Implementation nicht erfolgreich war, bildet diese Arbeit eine mögliche Grundlage zur Konzeptionierung eines Monitoringsystems.

Die gezeigte Entwicklungsumgebung bietet dabei die Basis für weitere Entwicklung. Die vorgestellte Liste der Testszenarien und -programme kann für die künftigen Kontrollen

erweitert werden. Ein besonderer Fokus kann dabei besonders auf die nebenläufige Gestaltung, aber muss vor allem auf die saubere Struktur gelegt werden, damit Fehler wie unbehandelte Exceptions nicht zum Absturz einer Anwendung führen.

Literaturverzeichnis

[1] Beyer, Betsi ; Jones, Chris ; Petoff, Jennifer ; Niall, Murphy R.: *Site reliability engineering: How Google runs production systems*. First edition. Beijing : O'Reilly Media, 2016. – URL https://landing.google.com/sre/books/. – Zugriffsdatum: 03.08.2020. – ISBN 978-1-4919-2912-4

[2] Brazil, Brian: *Prometheus: Up & Running: Infrastructure and application performance monitoring*. First edition. Sebastopol CA : O'Reilly Media and O'Reilly, 2018 // July 2018. – ISBN 1492034142

[3] Donahue, Gary A.: *Network warrior: Everything you need to know that wasn't on the CCNA exam ; covers Nexus*. 2nd ed. Sebastopol, Calif. : O'Reilly, 2007 (Safari Tech Books Online). – ISBN 978-1-449-38786-0

[4] Güldener, Arne: *SNMP-Überwachung eingebetteter Echtzeit-Ethernet-Systeme im Fahrzeug*. Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 03.07.2017. – URL https://edoc.sub.uni-hamburg.de/haw/volltexte/2017/4076/

[5] Hruschka, Peter ; Starke, Gernot: *arc42.de*. 2020. – URL https://arc42.de/. – Zugriffsdatum: 29.07.2020. – Lizenz siehe A.1

[6] IETF: *Hypertext Transfer Protocol Version 2 (HTTP/2)*. 05.2015. – URL https://tools.ietf.org/html/rfc7540. – Zugriffsdatum: 20.08.2020

[7] IETF: *Hypertext Transfer Protocol – HTTP/1.1*. 06.1999. – URL https://tools.ietf.org/html/rfc2616. – Zugriffsdatum: 20.08.2020

[8] IETF: *TRANSMISSION CONTROL PROTOCOL: DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION*. 09.1981. – URL https://tools.ietf.org/html/rfc793. – Zugriffsdatum: 20.08.2020

[9] IETF: *A Roadmap for Transmission Control Protocol (TCP) Specification Documents*. 09.2006. – URL https://tools.ietf.org/html/rfc4614. – Zugriffsdatum: 20.08.2020

[10] ITU-T: *ITU-T Rec. X.200 (07/94): Information technology - Open Systems Interconnection - Basic reference model: The basic model*. 07.1994. – URL https://www.itu.int/ITU-T/recommendations/rec.aspx?id=2820&lang=en. – Zugriffsdatum: 20.08.2020

[11] ITU-T: *ITU-T Rec. X.680 (08/2015): Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation*. 13.08.2015. – URL https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12479&lang=en. – Zugriffsdatum: 01.08.2020

[12] ITU-T: *ITU-T Rec. X.690 (08/2015): Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. 13.08.2015. – URL https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12483&lang=en. – Zugriffsdatum: 01.08.2020

[13] Krause, Hendrik: *Eigne Darstellung*

[14] Lawo AG: *Ember+ Documentation*. 11.11.2017. – URL https://github.com/Lawo/ember-plus/tree/master/documentation. – Lizenz siehe A.2

[15] Lawo AG: *VSM: IP BROADCAST CONTROL AND MONITORING SYSTEM*. 2020. – URL https://lawo.de/fileadmin/content/Products/Control/VSM/Lawo_VSM_EN.pdf. – Zugriffsdatum: 17.08.2020

[16]

[17] Lui, J.C.S. ; Misra, V. ; Rubenstein, D.: On the robustness of soft state protocols. In: *Proceedings of the 12th IEEE International Conference on Network Protocols, 2004. ICNP 2004 // Proceedings of the 12th IEEE International Conference on Network Protocols, ICNP 2004*. Los Alamitos, Calif. : IEEE and IEEE Computer Society, 2004, S. 50–60. – ISBN 0-7695-2161-4

[18] McGuire, Bill: *Monitoring Active Volcanoes: Strategies, procedures and techniques*. London : UCL Press, 1995. – URL https://external.dandelon.com/download/attachments/dandelon/ids/DE006E07F8395EA97CD7FC1257AC9003C426B.pdf,. – ISBN 1857280369

[19] Microsoft Corporation: *Windows and containers: Containers vs. virtual machines*. 22.10.2019. – URL https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/#how-containers-work. – Zugriffsdatum: 01.08.2020

[20] Miller, Barton P. ; Zhang, Mengxiao ; Heymann, Elisa R.: *The Relevance of Classic Fuzz Testing: Have We Solved This One?*. – URL http://arxiv.org/pdf/2008.06537v1

[21] Patel, P. ; Keogh, E. ; Lin, J. ; Lonardi, S.: Mining motifs in massive time series databases. In: Kumar, Vipin (Hrsg.): *2002 IEEE International Conference on Data Mining, 2002. Proceedings // Proceedings / 2002 IEEE International Conference on Data Mining, ICDM 2002*. Los Alamitos, Calif. : IEEE Comput. Soc and IEEE Computer Society, 2002, S. 370–377. – ISBN 0-7695-1754-4

[22] Prometheus ; Prometheus (Hrsg.): *Prometheus Architecture overview*. – URL https://github.com/prometheus/prometheus/. – Zugriffsdatum: 20.08.2020. – License: Apache License 2.0, see https://github.com/prometheus/prometheus/blob/master/LICENSE

[23] Rice, Liz: *Containers from Scratch*. 19.06.2018. – URL https://gotoams.nl/2018/sessions/429/containers-from-scratch

[24] solid IT: *DB-Engines Ranking - Trend of Time Series DBMS Popularity*. – URL https://db-engines.com/en/ranking_trend/time+series+dbms

[25] Soltesz, Stephen ; Pötzl, Herbert ; Fiuczynski, Marc E. ; Bavier, Andy ; Peterson, Larry: Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors. In: *Proceedings of the 2nd ACM SIGOPSEuroSys European Conference on Computer Systems 2007: EuroSys '07*. New York, NY : ACM, 2007, S. 275–287. – URL http://dl.acm.org/citation.cfm?id=1272996. – ISBN 9781595936363

[26] Tanenbaum, Andrew S. ; Bos, Herbert: *Modern operating systems*. Fourth edition. Boston : Pearson and Pearson Education, 2015. – ISBN 9780133591620

[27] Tanenbaum, Andrew S. ; Wetherall, David: *Computer networks*. 5. ed., internat. ed. Boston, Mass. : Pearson, 2011. – ISBN 978-0-13-212695-3

[28] van Steen, Maarten ; Tanenbaum, Andrew S.: *Distributed systems*. Third edition (Version 3.01 (2017)). London : Pearson Education, February 2017.

– URL https://www.distributed-systems.net/index.php/books/ds3/. – ISBN 9789081540629

[29] Yu, Denise: *Why are distributed systems so hard? A network partition survival guide*. 21.09.2018. – URL https://deniseyu.io/distsystalk

A Lizenzen

A.1 Arc42

arc42 is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

You are free to:

Share — copy and redistribute the material in any medium or format
Adapt — remix, transform, and build upon the material for any purpose, even commercially.

This license is acceptable for Free Cultural Works.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

A.2 Lawo - Ember+

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.3 Prometheus-net

The MIT License (MIT)

Copyright (c) 2015 andrasm

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

B Bilder der Anwedung

C Prometheus im NDR

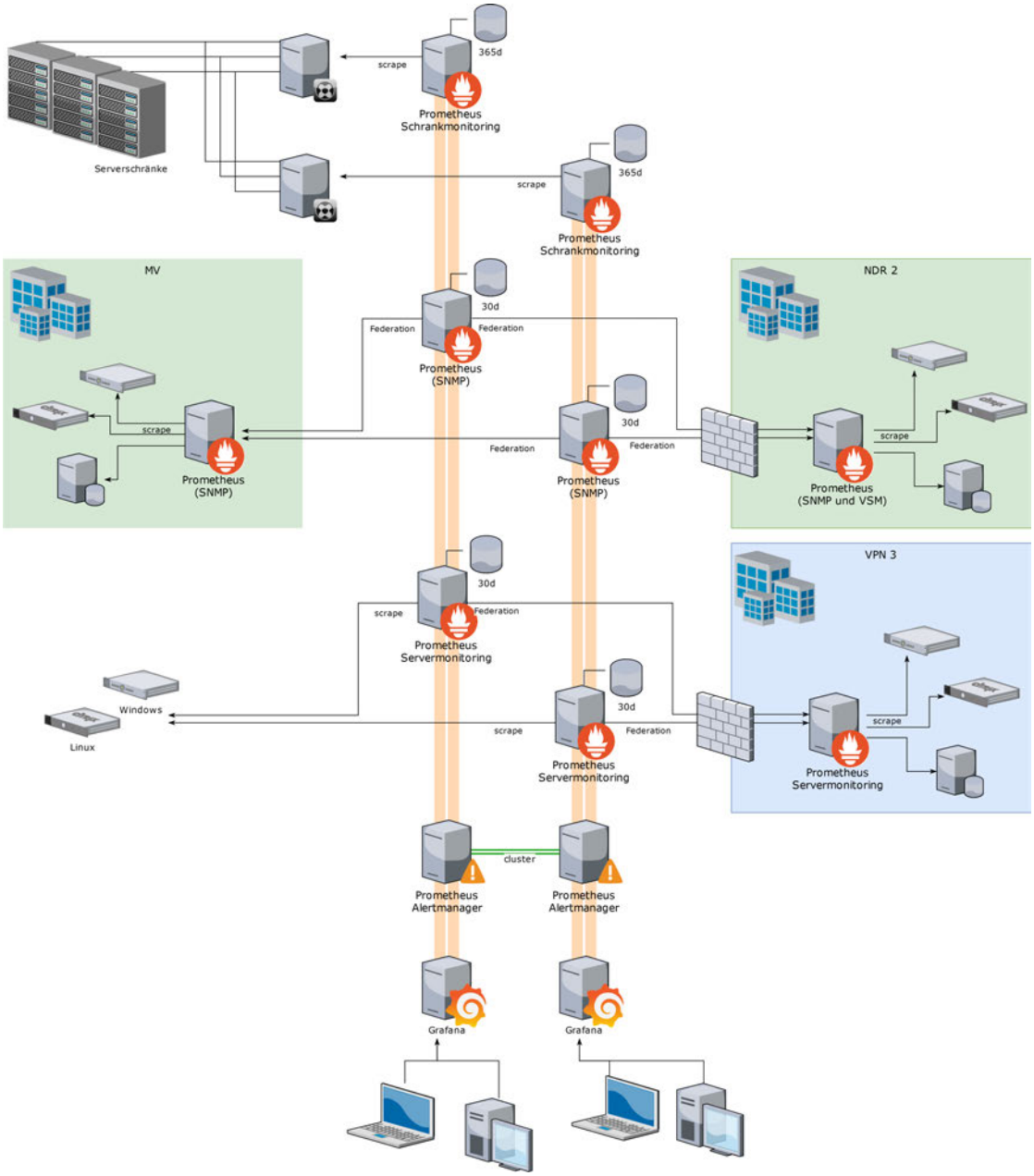

Abbildung C.1: Symbolbild: Prometheus im NDR, Status Quo vor Einsatzen dieser Arbeit. [13]

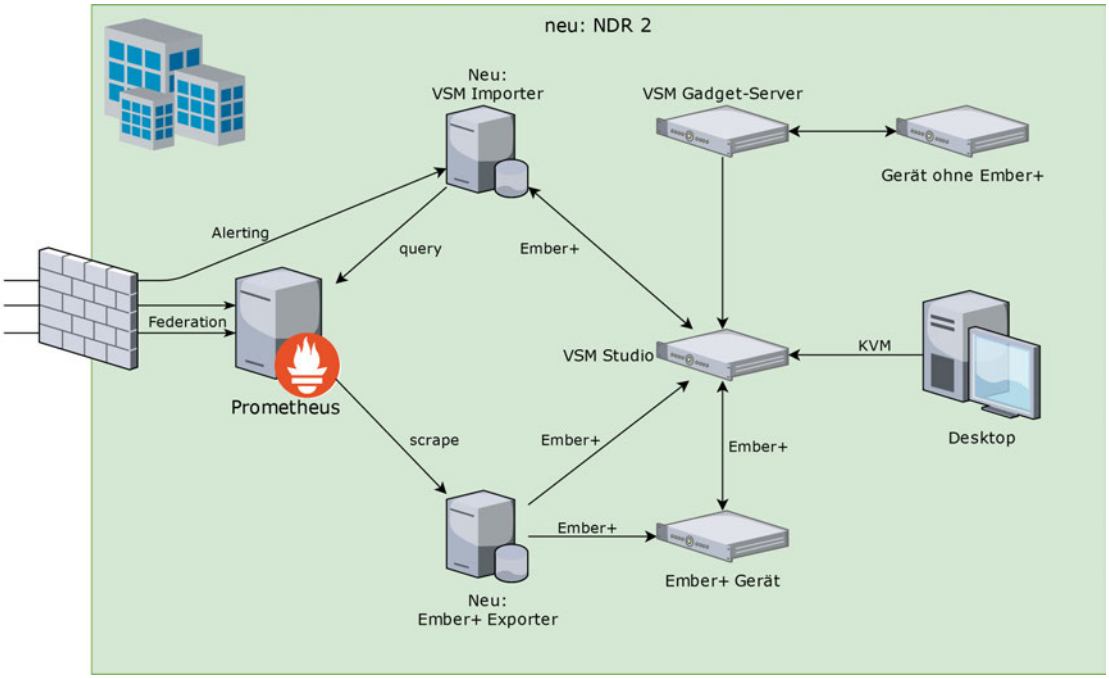

Abbildung C.2: Symbolbild: Prometheus im Studio NDR 2 im Detail. Zielvorgabe. [13]

Glossar

HAW Hamburg Die Hochschule für Angewandte Wissenschaften (HAW) Hamburg ist die vormalige Fachhochschule am Berliner Tor

Kreuzschiene Zentrales Gerät, das Signalwege miteinander verschalten kann. Verschiedene Quellen können auf beliebige Senken geleitet werden

Mischpult Bediengerät, um mehrere Bild- oder Tonsignale zusammenzuführen, zu steuern, zu überlagern und zu mischen

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung eines robusten, kooperativen Monitoring-Konzepts im Umfeld eines Fernsehstudios

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	[redacted]
Ort	Datum	Unterschrift im Original