Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Micha Severin

Konzeption und Realisierung einer Plattform zur Echtzeit-Netzwerkdatenanalyse

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Micha Severin

Konzeption und Realisierung einer Plattform zur Echtzeit-Netzwerkdatenanalyse

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Zukunft
Zweitgutachter: Prof. Dr. Hübner

Eingereicht am: 8. Mai 2017

Micha Severin

Thema der Arbeit
Konzeption und Realisierung einer Plattform
zur Echtzeit-Netzwerkdatenanalyse

Stichworte
Netzwerkdatenanalyse, Konzeption Plattform, Stream-Processing, Echtzeit, Big Data

Kurzzusammenfassung
Eine Verarbeitung und Analyse von Netzwerkdaten ist durch verschiedene Herausforderungen im Kontext von Big Data geprägt. Die Konzeption sowie Realisierung einer Plattform für Netzwerkdaten ist das Thema dieser Arbeit, wobei das Stream-Processing ein zentrales Charakteristikum darstellt. Nach der Fertigstellung der Plattform wird anhand zweier Szenarien jeweils eine konkrete Einsatzmöglichkeit aufgezeigt und es werden deren Analyseergebnisse vorgestellt.
Micha Severin

Title of the paper
Conception and implementation of a platform for real-time network analytics

Keywords
network analytics, conception platform, stream-processing, real-time, big data

Abstract
Networkdata processing and analysis are quite challenging in the context of big data. Conception and implementation of a platform for real-time network analytics is the main purpose of this thesis. Furthermore stream-processing is an important characteristic of the platform. Finally two scenarios will show a possibility of how to use the platform and the application results will be described.

Inhaltsverzeichnis

1 Einleitung

Das Mooresche Gesetz von 1965 besagt, dass sich die Anzahl der Transistoren pro Flächeneinheit eines Prozessors alle zwei Jahre verdoppelt (Blau, 2009; Conconi, 2015). Dieses Gesetz wurde von Gordon Earle Moore, der 1929 in San Francisco geboren wurde und Mitgründer des Unternehmens Intel ist, aufgestellt. Lässt man außer Acht, dass dieses Gesetz vermutlich erst nach mehr als 50 Jahren seine Gültigkeit verlieren wird (Waldrop, 2016), drückt es doch eines genau aus: den enormen Fortschritt der Technologie und den damit einhergehenden Zuwachs an Leistung von Computerprozessoren.
Ein steter Gewinn an Leistung bildet somit den Grundstein für neue und vielfältige Technologien und Techniken der Verarbeitung von Daten durch einen Prozessor. Die Quantität und Komplexität von Daten sind die zwei Faktoren, die beim Prozess der Datenverarbeitung eine besondere Herausforderung darstellen.
Bei dieser Art der Betrachtung steht der reine Prozess der Verarbeitung im Mittelpunkt. Auf die Verarbeitung von Daten folgt meist die Notwendigkeit der Persistierung dieser und neuer Daten als Resultat der Verarbeitung. Das Aufkommen von vielen Daten und der Durchsatz von Daten prägen unter anderem den Begriff „Big Data“ (Cox und Ellsworth, 1997a; Pippal u. a., 2014), der erstmals 1997 Erwähnung fand. Ein Charakteristikum von Big Data besteht darin, dass konventionelle Methoden der Datenverarbeitung häufig nicht den Herausforderungen der Verarbeitung gewachsen sind. Daher entstehen neue Technologien und Anwendungen, die diesen Herausforderungen besser begegnen.
Es zeigt sich, dass die Entwicklung des steigenden Aufkommens von Daten einem ähnlichen wie dem von Moore formulierten Gesetz unterliegt. Eine Studie aus dem Jahr 2011 beschreibt die Entwicklung des Volumens von Datenbanken (American Multinational Corporation, 2011). Dabei wurde festgestellt, dass sich die Größe von Datenbanken alle zwei Jahre verdoppelt (LL.M. u. a., 2014; Hillebrand und Finger, 2015).
Diese Tatsachen beschreiben sowohl den steten Entwicklungsfortschritt der Prozessortechnik sowie den damit einhergehenden Anstieg des Volumens von Datenbanken und führen zu der Notwendigkeit leistungsfähigerer Möglichkeiten der Datenverarbeitung.

1.1 Motivation

Es gibt zahlreiche Bereiche der Informatik, in denen die Auseinandersetzung mit den Auswirkungen von Big Data spürbar sind. Vielen ist gemeinsam, dass der physische Ort der Erhebung von Daten nicht immer dem Ort der Verarbeitung entsprechen muss. Dieser Annahme geht voraus, dass die Erhebung und Verarbeitung zeitlich unmittelbar aufeinander folgen. Betrachtet man diesen Fall genauer, folgt daraus die Erkenntnis, dass auch Bereiche der Netzwerktechnik auf Seiten der Hard- sowie Software wichtig für die Bewertung von Systemen der Verarbeitung von Massendaten sind.
Betrachtet man diese Aspekte vor dem Hintergrund wirtschaflicher Entscheidungsprozesse auf der Ebene des Managements eines Unternehmens, lässt sich die folgende Kausalität aufzeigen: Eine Entscheidung, die im Sinne eines Unternehmens getroffen werden muss, beruht auf Erkenntnissen, die auf den Prozess der Verarbeitung von Daten zurückzuführen sind. Somit bildet das Verarbeitungsergebnis die Ursache für die Wirkung der unternehmerischen Entscheidung. Hierbei erweist sich der Zeitraum zwischen der Verarbeitung von Daten und dem Ergebnis als wichtig für die zu treffende Entscheidung.
Ein unmittelbarer Gewinn an Erkenntnissen beschreibt somit die Echtzeitfähigkeit von Systemen zur Verarbeitung von Daten im Zusammenhang mit einer nicht zentralisierten Anwendung.

1.2 Zielsetzung

Diese Masterthesis hat zum Ziel eine Plattform zu konzipieren und zu entwickeln, die sich für die Verarbeitung von Daten im Bereich der Netzwerkdatenanalyse einsetzen lässt. Hierbei liegt der Fokus mehr auf dem Stream Processing als auf dem Batch Processing. Dies schließt jedoch nicht grundsätzlich eine Batchverarbeitung aus.
Bei der Entwicklung der Plattform stehen einige Faktoren wie der Betrieb einer konkreten Anwendung als Verteiltes System im Mittelpunkt. Hierbei leitet sich diese Zielsetzung von der Notwendigkeit eines performaten Systems unter den Einflüssen von Big Data ab. Weiterhin wird die Anforderung einer Mikroarchitektur an die Komponenten der Plattform gestellt. Der Grund für diese Entscheidung stellt unter anderem darauf ab, etwaige Komponenten einfacher aus dem Gesamtkomplex lösen zu können um sie durch andere zu ersetzen.
Die in der Motivation beschriebene Echtzeitfähigkeit bildet weiterhin ein zentrales Charakteristikum in der Entwicklung und sie wird auch durch die konkreten Analyseszenarien geprägt. Neben den Möglichkeiten der Visualisierung von Analyseergebnissen ist die Echtzeitfähigkeit schlussendlich auch für die Plattform ausschlaggebend.

Die entwickelte Plattform wird in den letzten beiden Kapiteln dieser Arbeit für zwei praktische Analyseszenarien zum Einsatz gebracht. Im ersten Szenario wird eine Analyse von Primär- und Sekundärrequests von Webseiten durchgeführt. Dabei gilt es zu untersuchen, welche Server eine Schnittmenge bilden, die von einer definierten Anzahl von Primärseiten aus aufgerufen werden. Im zweiten Szenario findet dann eine Untersuchung des Netzwerkdatenverkehrs eines Tor-Exit-Nodes statt. Das heißt im konkreten Fall: Anonymisierter Netzwerkdatenverkehr wird in der Art untersucht, dass Erkenntnisse darüber entstehen, welche Adressen einer hohen Frequentierung unterliegen, ohne dass eine Deanonymisierung stattfindet.

1.3 Abgrenzung

Die Ausarbeitung umfasst die Bereiche Konzeption und Entwicklung einer Plattform. Das Augenmerk liegt hierbei auf der Netzwerkdatenanalyse. Das Batch Processing ist dabei weder Bestandteil noch Ziel der Plattform. Ein vorhandener Bestand an Massendaten ließe sich aber prinzipiell auch in einen Stream überführen, wobei dabei auftretende Probleme nicht in der Arbeit berücksichtigt werden.
Ein plattformübergreifender Betrieb steht nicht primär während der Konzeption und Realisierung im Mittelpunkt. Das heißt, es wird am Ende kein Szenario auf verschiedenen Betriebssystemen gleichermaßen initiierbar und realisierbar sein.
Die Analysen in der Arbeit dienen auch als konkrete Anwendungen um den Einsatz der Plattform zu beschreiben. Hierbei werden die Analysen valide Ergebnisse produzieren, sie erheben aber nicht den Anspruch einer erschöpfenden Behandlung aller Analysemöglichkeiten. Das bedeutet, sie sind als konkreter Einsatz der Plattform zu sehen.
Diese Ausarbeitung hat das primäre Ziel der Entwicklung einer Plattform zur Netzwerkdatenanalyse. Dabei gibt es in diesem Themenbereich bereits verschiedene Systeme und Produkte. Es findet in der Analyse zwar eine Auseinandersetzung mit diesen Systemen statt, aber es entspricht nicht der Zielsetzung dieser Arbeit, eine allumfassende Evaluation dieser Systeme zueinander sowie bezogen auf das eigene Vorhaben durchzuführen.

2 Grundlagen

Dieses Kapitel beschreibt die grundlegenden Technologien, Systeme und Konzepte, die für diese Arbeit, die Plattform und die zu analysierenden Szenarien von Bedeutung sind. Behandelt werden zumeist die zentralen Aspekte der jeweiligen Themen und deren Bezug zum Thema dieser Arbeit. Das bedeutet, es finden primär die Themenbereiche Erwähnung, die eine wichtige inhaltliche Voraussetzung für die späteren Kapitel dieser Ausarbeitung bilden.
Die Grundlagen behandeln unter anderem Big Data mit dem Augenmerk auf die Bereiche des Themas, die als ursächlich für die Entwicklung neuer Technologien anzusehen sind. Weiterhin wird das Themengebiet des Complex Event Processings genauer beleuchtet, da dieses als Grundlage für den Bereich der echtzeitverarbeitenden Systeme oder allgemeiner gesagt der Echtzeitverarbeitung anzusehen ist. Verteiltes Rechnen (engl. distributed computing) ist ein wichtiger Bereich der Informatik, der sich in den Grundlagen unter dem Themengebiet der Verteilten Systeme wiederfindet. Zudem werden hierbei die Aspekte hervorgehoben, die für die entwickelte Plattform von zentraler Bedeutung sind. Abschließend werden Microservices bezogen auf die Bedeutung des Betriebs einer automatisierten Anwendungsverteilung behandelt.

2.1 Big Data

Der Begriff Big Data fand erstmalig Erwähnung im Jahr 1997 in einer wissenschaftlichen Arbeit der National Aeronautics and Space Administration (NASA)-Mitarbeiter Michael Cox und David Ellsworth (Buyya u. a., 2016; Freiknecht, 2014, 5,8). Sie beschreiben in ihrer Arbeit das Problem, dass große Datensätze durch vorhandene Computer nicht verarbeitet werden können. Grund hierfür ist, dass der Arbeitsspeicher, lokale Speicher und auch externe Speicher nicht ausreichen würden (Cox und Ellsworth, 1997b).
Die Weiterentwicklung der Computertechnologie hat diese und weitere Probleme über die Jahre auch in anderen Bereichen der Wissenschaft entstehen lassen. Mit dem Einzug der Digitalisierung in viele Bereiche des Lebens und der Arbeit hielt der Begriff auch Einzug in die Wirtschaft.

2.1.1 Definition

Der Begriff Big Data wurde anfänglich zur Beschreibung der eigentlichen Datenmenge (Quantität) verwendet. Dagegen haben sich mit der Zeit verschiedene Definitionen gebildet, die dem Begriff noch weitere Eigenschaften zuordnen. Gemeinsam ist zumeist aber allen Definitionen, dass die Quantität, oft auch als Volumen (engl. volume) bezeichnet, ein zentrales Charakteristikum des Begriffs ist. Somit berufen sich einige Definitionen des Begriffs Big Data ausschließlich auf die Quantität.

> **Definition 1** (Big Data)**.** *Mit „Big Data“ werden große Mengen an Daten bezeichnet, die u.a. aus Bereichen wie Internet und Mobilfunk, [...] sowie Flug- und Fahrzeugen stammen und die mit speziellen Lösungen gespeichert, verarbeitet und ausgewertet werden.*

Abbildung 2.1: Definition nach Wirtschaftslexikon (2016)

Diese Definition geht nicht näher auf die eingangs beschriebenen Faktoren der neuen Technologien ein, die auch im Zuge der Veränderungen und des Einsatzes von Computern auftreten. Detailliertere Definitionen hingegen ziehen meist noch zwei weitere Kernpunkte zu einer Definition heran. Das ist einerseits die Komplexität (engl. complexity), die die Art der Daten beschreibt, um die es sich handelt. Andererseits werden auch die Technologien der Verarbeitung mit herangezogen und definieren Big Data somit wie folgt:

> **Definition 2** (Big Data)**.** *Big data is a term describing the storage and analysis of large and or complex data sets using a series of techniques including, but not limited to: NoSQL, MapReduce and machine learning.*

Abbildung 2.2: Definition nach Ward und Barker (2013)

Die unterschiedliche Betrachtung einiger Aspekte der Definitionen von Big Data zeigen eines auf: Je nach Standpunkt, Bereich und Branche wird die Komplexität von Daten wichtig für eine Definition oder nicht. Im folgenden Abschnitt werden zentrale Charakteristika von Big Data beschrieben, die nach einem Modell des Unternehmens Gartner[1] populär wurden.

[1] http://www.gartner.com/

2.1.2 3-V-Modell

Das ursprünglich 2001 entstandene 3-V-Modell wurde im Zuge einer wissenschaftlichen Arbeit des Unternehmens Gartner entwickelt. Thema der Arbeit war die Verwaltung von 3D-Daten (Laney, 2001) anhand dreier spezieller Gesichtspunkte. Dieses Modell beschreibt wichtige Charakteristika von Big Data (Assunção u. a., 2015). Mitunter wird das Modell auch als Multi-V-Modell bezeichnet (Yu und Guo, 2016), das beschreibt, dass dem Modell nach den drei ersten Eigenschaften noch weitere gefolgt sind. Im folgenden Abschnitt werden die drei ursprünglichen Eigenschaften sowie zwei später hinzugekommene erläutert.

2.1.2.1 Volume

Die erste Eigenschaft **Volume** (dt. Volumen, Umfang, Menge) betrifft, wie schon in der Definition erwähnt, die Quantität der Daten - also die Größe aller betroffenen oder anfallenden Daten. Hierbei ist die Rede von Datenmengen, die in Petabyte oder Zetabyte quantifiziert werden und nicht mit gewöhnlichen Systemen verarbeitet werden können.

2.1.2.2 Variety

Als zweite wird **Variety** (dt. Vielfalt, Verschiedenheit) genannt. Sie drückt eine Unstrukturiertheit aus, die den Daten zugrundeliegt. Das bedeutet, etwaige Erkenntnisse sind nicht ohne Verarbeitung erreichbar. Insofern sind direkte Ergebnisse aus Daten nicht erkennbar.

2.1.2.3 Velocity

Velocity (dt. Geschwindigkeit) beschreibt die Geschwindigkeit, mit der Daten generiert, verarbeitet oder ausgewertet werden können. Hierbei ist zu beachten, dass Attribute wie Echtzeitfähigkeit in der Verarbeitung der Daten eine wichtige Rolle spielen.

2.1.2.4 Veracity

Eine häufig dem V-Modell zusätzlich zugeordnete Eigenschaft beschreibt **Veracity** (dt. Wahrhaftigkeit, Aufrichtigkeit). Sie beschreibt die Messgenauigkeit der Daten und ihrer Analyseergebnisse. Das beinhaltet auch die spätere Aussagekraft etwaiger Ergebnisse, die beim Prozess der Verarbeitung gewonnen werden.

2.1.2.5 Value

Die fünfte Eigenschaft im V-Modell wird **Value** (dt. Wert) genannt: Gemeint ist damit der Wert der Information, der nach einer Analyse erlangt wird. Verarbeitungsprozesse ohne konkret messbaren Nutzen sind hierbei zu vermeiden.

2.1.3 Digitalisierung

Der Begriff des „Digitalen Zeitalters“ wird häufig verwendet um den Wandel der Nutzung von analogen und digitalen Daten auszudrücken. Dabei wird angenommen, dass es in den frühen 2000er Jahren erstmals mehr digital als analog gespeicherte Daten gab (Hilbert und López, 2011). Dieser Wandel in der Art der gespeicherten Informationen hat vielseitige Ursachen, zum Beispiel den Fortschritt im Bereich der Mikroelektronik und Kommunikationstechnik (Langheinrich und Mattern, 2003).
Die Digitalisierung beschreibt originär den Prozess der Datenumwandlung von analogen in digitale Daten. Mittlerweile hat sich diese strikte Auslegung aber eher in die Richtung einer Definition verändert, in der allgemein die Generierung von Informationen in digitaler Form gemeint ist.
Daten in digitaler Form haben grundsätzlich eine höhere Zugänglichkeit gegenüber analogen Daten, da die Reproduktion - unabhängig der zugrundeliegenden Information - technisch keine große Herausforderung darstellt. Diese Veränderung hat zur Folge, dass die Infrastruktur zur Datenübermittlung auf der Seite der Hard- und Software als Bindeglied wichtig ist.

2.1.4 Datenschutz

Die Entwicklung im Bereich der Datenverarbeitung bringt neue Möglichkeiten und Verbesserungen in vielen gesellschaftlichen Bereichen mit sich. So hat das Los Angeles Police Department (LAPD) ein mathematisches Modell zur Vorhersage von Nachbeben bei Erdbeben entwickelt. Die Intention dieses Modells ist der effektivere Schutz der Bevölkerung in gefährdeten Gebieten. Es zeigte sich, dass sich dieses Modell aber auch für Daten von Verbrechen aus der Vergangenheit verwenden ließ. Im Ergebnis konnten so wiederum bereits erfolgte Verbrechen vorhergesagt werden. Folglich ließ sich dieses Modell auch für aktuelle Verbrechensbekämpfung einsetzen (Uskali und Kuutti, 2015; Siegel, 2016).
Es lassen sich auch Beispiele finden, in denen der Bereich der „Predictive Analytics“ für die Vorhersage von Kaufentscheidungen einsetzbar ist (Weiss, 2009).
In Deutschland gilt das Recht auf informationelle Selbstbestimmung, die als zentrale Errungenschaft angesehen wird (Weichert, 2013, 2). Sie definiert, dass die eigene Person ausschließlich

selbst über die Weitergabe und Verwendung der eigenen personenbezogenen Daten bestimmen kann. Außerdem gibt es den Grundsatz der Datenvermeidung und Datensparsamkeit (§ 3a Bundesdatenschutzgesetz (BDSG)). Es ist bereits im Vorwege der Erhebung von Daten darauf zu achten, dass möglichst wenige Daten gespeichert werden. Weiterhin besagt der Grundsatz, dass auch möglichst keine oder wenige personenbezogenen Daten zu speichern sind.
Der Grundsatz und das Gesetz definieren somit die innerdeutsche Handhabung im Umgang mit Daten. Ersichtlich wird hierbei aber, dass die Eigenschaften des V-Modells (2.1.2) mitunter mit diesen Gesetzen kollidieren können. Die Chancen und Risiken werden somit nicht ausschließlich unter dem Gesichtspunkt der technischen Machbarkeit gesehen, sondern sie betreffen sowohl Gesellschaftspolitik als auch die nationale Gesetzeslage.

2.1.5 Der Wert von Daten

Der Wert einer Information bemisst sich anhand vieler verschiedener Faktoren, die von Fall zu Fall variieren können. Im Kontext von Big Data beschreibt ein „V“ des V-Modells den Wert (engl. value) von Daten. Handelt es sich zum Beispiel um Gesundheitsdaten einer Person, die Aufschluss über eine Erkrankung und deren Verlauf geben, so ist der informationelle Wert dieser Daten hoch. Gemeint ist hierbei zum Beispiel ein potenzieller Arbeitgeber, der eine Person einzustellen beabsichtigt. Dieses ist nur ein Beispiel für das Interesse Dritter an Informationen aus Daten. Weitere Beispiele lassen sich für Zugangsdaten, das Kaufverhalten, die Kaufhistorie oder mediale Daten finden.
Die erwähnten Beispiele sind exemplarisch für direkt zugängliche Informationen aus Daten. Im Kontext von Big Data sind diese Informationen aber mitunter nicht direkt abrufbar beziehungsweise auch nur indirekt in den Daten enthalten. Somit geht die Verarbeitung der Daten dem informationellen Erkenntnisgewinn voraus. Gliederbar ist der Prozess im Allgemeinen in verschiedene Bereiche, die einer Wertschöpfungskette entsprechen. Beschrieben wurde der Prozess einer Wertschöpfungskette erstmals von Michael E. Porter (Porter, 1985). Gilbert Miller und Peter Mockhaben haben in ihrer Arbeit „From Data to Decisions: A Value Chain for Big Data“ diesen Bezug hergestellt (Miller und Mork, 2013). Sie beschreiben und untergliedern den Prozess in verschiedene Bereiche, um am Ende der Kette einen messbaren und beim Konsumenten abrufbaren Wert zu erreichen. In der Abbildung 2.3 ist die Wertschöpfungskette ersichtlich.
Im Allgemeinen lässt sich der konkrete Wert einer Information, die in großen Datenmengen enthalten ist, durch eine wiederkehrende Formulierung aufzeigen. Die hergestellte Analogie in der Formulierung beschreibt „Daten als das Öl des 21. Jahrhunderts“ (Henze, 2014). Dennis D. Hirsch spricht in diesem Zusammenhang auch von Daten als dem „neuen Öl“ (Hirsch, 2013).

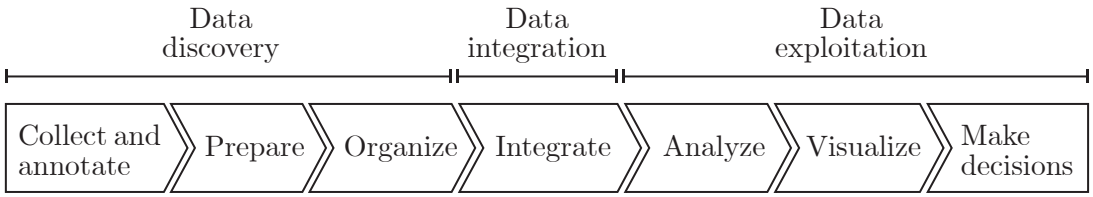

Abbildung 2.3: Wertschöpfungskette nach Miller und Mork (2013)

2.2 Verteilte Systeme

Eine wichtige Eigenschaft des entwickelten Systems zur Netzwerkdatenanalyse ist der Betrieb in Form eines verteilten Systems. Zweifellos ist das größte Verteilte System in der Informatik das World Wide Web (Schill und Springer, 2012). In diesem Abschnitt werden wichtige Grundvoraussetzungen und Prinzipien beschrieben, die im Zusammenhang mit dem entwickelten System hervorzuheben sind.
Vor dem Hintergrund des CAP-Theorems müssen Web-Services immer einen Kompromiss zwischen den Eigenschaften der Konsistenz (engl. consistency), Verfügbarkeit (engl. availability) und Ausfalltoleranz (engl. partition tolerance) finden (Gilbert und Lynch, 2012). Der Nutzen von verteilten Systemen lässt sich an mannigfaltigen Herausforderungen aufzeigen. Dazu zählt unter anderem das kooperative Arbeiten von Gruppen mit Hilfe von technischen Systemen. Hierzu gibt es ein eigenes Forschungsgebiet: Computer Supported Cooperative Work (CSCW). Außerdem ist auch die Lastverteilung von Anfragen an ein System sowie eine entsprechende Verarbeitung der jeweiligen Anfragen an das System von Bedeutung. Aus der Verteilung der Last eines Systems ergibt sich somit eine Nebenläufigkeit von Prozessen, die unter Umständen bei transaktionalem Verhalten neue Herausforderungen schafft.
Mitunter sind nicht nur technische Aspekte von Bedeutung, sondern auch die Wirtschaftlichkeit verteilter Systeme. Hierzu sind viele kostengünstige „kleine“ Rechner in der Finanzierung unter Umständen günstiger gegenüber einem „großen“ und leistungsstarken Computer.
Dieser Abschnitt beschreibt Bereiche wie die Architektur von Computerclustern. Die konkreten Anwendungen zur späteren Analyse laufen auf entsprechenden Clustern, sodass die zugrundeliegende Architektur von Bedeutung ist. Weiterhin finden noch Charakteristika von Anwendungen Erwähnung sowie der Bereich des Cloud Computings.

2.2.1 Cloud Computing

Im Allgemeinen weist das Cloud Computing die zentrale Eigenschaft auf, dass es eine lokale Interaktion mit einem System gibt, wobei die Verarbeitung global geschieht. Dieser Grundsatz ist unter anderem auch eine Konsequenz der Herausforderungen von Big Data, die in Abschnitt 2.1 Erwähnung finden. Die Einsatzbereiche, in denen die Eigenschaft einer lokalen Interaktion und einer entfernten Verarbeitung vorkommen, sind vielseitig.
Das National Institute of Standards and Technology (NIST) unterscheidet vier verschiedene Arten von Cloud-Strukturen: private cloud, community cloud, public cloud, hybrid cloud (Mell u. a., 2011). Das Modell des Cloud Computings lässt sich in drei verschiedene Bereiche unterteilen, die sich in ihrer Art der bereitgestellten Dienstleistung unterscheiden. Diese Unterscheidung führt zu dem sogenannten Cloud-Stack, der eine Hierarchie der Dienstleistung dergestalt beinhaltet, dass die verschiedenen Arten technisch betrachtet aufeinander aufbauen. So lässt sich der Cloud-Stack in die folgenden Kategorien gliedern:

2.2.1.1 Infrastructure as a Service (IaaS)

Die erste Art des Cloud Computing definiert sich anhand der bereitgestellten technischen Infrastruktur. Diese Form ist auch als Basis des Cloud-Stacks zu betrachten. Hierbei wird eine komplette technische Infrastruktur - zum Beispiel virtuelle Instanzen von Rechnern im Verbund als Dienstleistung - angeboten. Diese Instanzen haben zumeist keine weitere oder spezifische Konfiguration, sodass es dem Nutzer überlassen ist, welche Art von Service er darauf aufbauen möchte.

2.2.1.2 Platform as a Service (PaaS)

In der zweiten Ebene des Cloud-Stack ist die Art des PaaS angesiedelt. Diese zeichnet sich unter anderem dadurch aus, dass es entgegen der IaaS keinen direkten Zugriff auf die jeweiligen Instanzen mehr gibt. Vielmehr bringt der Entwickler die Logik seiner Anwendung in das System ein. Die Nutzung der Plattform findet dann nach außen über Schnittstellen statt, die in der Anwendung implementiert sind. Die konkrete Verteilung der Anwendung sowie der Betrieb werden dann durch die zugrundeliegende Schicht der Infrastruktur übernommen (IaaS).

2.2.1.3 Software as a Service (SaaS)

In der obersten und letzten Schicht sind Dienstleistungen zu verorten, die sich nach dem Prinzip einer nach außen komplett angebotenen Software richten. Diese Art bietet dem Benutzer eine fertige Anwendung, die häufig über entsprechende Abo-Modelle von Unternehmen angeboten

werden. Somit hat der Benutzer keinen Zugriff auf die Instanzen und etwaige Schnittstellen wie in den darunter liegenden Schichten des PaaS und IaaS. Diese Art des Betriebs einer Software bringt somit gute Möglichkeiten mit sich um hochskalierbare Anwendungen zu schaffen, sodass durch das Schichtenmodell schnell weitere Instanzen hinzugeschaltet werden können.

2.2.2 Cluster Computing

Die Domäne des Cluster Computing definiert sich über eine Vereinigung verschiedener Bereiche der Informationstechnologie (Buyya u. a., 2000). Hierzu zählen unter anderem die parallele Berechnung und Verarbeitung von Aufgaben eines Prozesses (engl. parallel computing). Eine Konsequenz der Parallelität ist wiederum eine hohe Performance des Systems (engl. high-performance), die als wichtige Eigenschaft zu nennen ist. Weiterhin ist die Hochverfügbarkeit eines Systems zu gewährleisten (engl. high-availability), die primär durch den Aspekt der Verteilung eines Systems erreicht wird. Somit bildet auch die Verteilung an sich eine weitere Eigenschaft in der Domäne des Cluster Computings. Anhand der verschiedenen Eigenschaften und Bereiche des Cluster Computings sind zwei Arten des Betriebs von Clustern zu unterscheiden. Im Folgenden werden das HA- sowie HPC-Cluster eingehender beschrieben.

2.2.2.1 HA-Cluster

Das Akronym „HA“ steht für das Attribut der Hochverfügbarkeit (engl. high-availability) eines Clusters. Sie werden mitunter auch als „Failover Cluster“ bezeichnet (Irfani, 2015). Diese Anforderung der Hochverfügbarkeit wird an ein Cluster gestellt, wenn ein System beziehungsweise eine Anwendung nur geringe bis keine Ausfallzeit haben darf. Es gibt verschiedene Techniken, anhand derer versucht wird die Ausfallzeit möglichst gering zu halten. Dazu zählt das Spiegeln der lokalen Festplatte um im Fehlerfall den Knoten verlustfrei wieder dem Verbund hinzufügen zu können. Auf der Seite der Infrastruktur wird eine doppelte Konnektivität gewährleistet, um im Fall des Verlusts eines Netzwerkinterfaces auf ein anderes Interface ausweichen zu können. Diese Doppelung findet häufig auch bei der Stromversorgung eines jeden Knotens des Systems statt.

2.2.2.2 HPC-Cluster

Ein HPC-Cluster definiert sich durch eine hohe Rechenkapazität, wobei die Abkürzung „HPC” für High Performance Computing steht. Diese Cluster haben das Ziel eine Aufgabe möglichst schnell zu erledigen, wobei die zuvor beschriebene Eigenschaft der Verfügbarkeit in den Hintergrund tritt. Jeder Knoten eines Clusters bekommt in diesem Cluster zumeist nur einen

Teil einer Gesamtaufgabe zur Berechnung beziehungsweise Verarbeitung zugewiesen. Bei diesem Verfahren findet durch das Job-Scheduling eine Verteilung der Teilaufgaben anhand unterschiedlicher Kriterien statt. Eine wichtige infrastrukturelle Voraussetzung ist ein schnelles zugrundeliegendes Netz, da mittels Message Passing eine Kommunikation zwischen den Teilaufgaben stattfinden muss.

2.3 Event Processing

Dieser Abschnitt beschreibt das Teilgebiet des Complex Event Processing (CEP) sowie thematisch und inhaltlich relevante Bereiche wie das Event Stream Processing (ESP). Der im Zusammenhang mit dem Event Processing häufig gebräuchliche Begriff der „Echtzeit“ findet weiterhin Erwähnung und eine klare Definition. Abschließend werden wichtige Architekturansätze wie die Event-Driven Architecture und die Service-Oriented Architecture beschrieben. Zuvor bedarf der Begriff des Events aber einer genaueren Definition, da diese wichtig für das Verständnis der folgenden Arbeit ist. Hierzu lässt sich die Definition von Michelson heranziehen, der ein Event wie folgt beschreibt:

> **Definition 3** (Event). *Each event occurrence has an event header and event body. The event header contains elements describing the event occurrence, such as the event specification ID, event type, event name, event timestamp, event occurrence number, and event creator. These elements are consistent, across event specifications.*

Abbildung 2.4: Definition nach Michelson (Michelson, 2006, S.3)

2.3.1 Complex Event Processing

Das CEP ist ein Teilgebiet der Informatik, das sich im Allgemeinen mit der Analyse verschiedenartiger Ereignisströme befasst (Bruns und Dunkel, 2015).
Konsens herrscht zumeist darüber, dass das Buch „The Power of Events“, das im Jahr 2002 von David Luckham veröffentlicht wurde, als Primärliteratur für das CEP anzusehen ist (Robins, 2010; Schlegel, 2013; Retter, 2011). Obwohl der Begriff zuerst von Luckham genannt wurde, hat sich mit der Zeit die Definition dessen, was dem CEP zuzuordnen ist, verändert. Kurzum werden Ereignisse als „voneinander abhängig“ (Schulte und Bülchmann, 2016) beschrieben.

Das hat zur Folge, dass etwaige Muster in Datenströmen zu erkennen sind, anhand derer entsprechende Analysen auf den Ereignissen durchlaufen werden.

2.3.2 Event Stream Processing

Das ESP ist inhaltlich vom CEP nicht weit entfernt. Dennoch lassen sich ein paar Eigenschaften hervorheben, anhand derer der Unterschied eindeutig wird. Zum einen wird beim Complex Event Processing nicht nur auf eine singuläre Quelle zugegriffen. Das bedeutet, die Summe aller Datenquellen bildet die Eingabe. Den verschiedenen Datenquellen muss keine Homogenität ihrer entsprechenden Events zugrundeliegen. Luckham beschreibt diese Summe der Quellen auch als Cloud (Luckham, 2006). Anders verhält es sich beim Event Stream Processing. Beim Event Stream Processing handelt es sich um einen Stream. Dabei liegt einem Stream zugrunde, dass seine auftretenden Ereignisse anhand der Zeit geordnet sind. Sie bilden somit eine eindeutige Sequenz. Grundsätzlich kann eine Cloud aber auch aus vielen einzelnen Streams bestehen, wobei diese zueinander keine Beziehung aufweisen müssen.
So lässt sich herausstellen, dass beim ESP typischerweise eine homogene Datenbasis vorhanden ist. Weiterhin liegt der Fokus mehr auf der reinen Analyse der Streams gegenüber dem CEP, bei dem durch Pattern Matching und Aggregation häufig neue Events geschaffen werden (Robins, 2010).

2.3.3 Real-time

Der Begriff der Echtzeit (engl. real-time) ist ein häufiger Terminus, der im Zusammenhang mit Systemen zur Erhebung und Analyse von Daten verwendet wird. Dabei beschreibt er zunächst allgemein, dass etwaige Ergebnisse eines Prozesses zur Verarbeitung von Daten nach einer gewissen Zeitspanne erwartet werden. Diese Anforderung wird an Systeme zur Verarbeitung großer Datenmengen (2.1) gestellt oder diese Eigenschaft wird ihnen zugesprochen.
Im Kontext dieser Arbeit ist die Echtzeitfähigkeit einer konkreten Anwendung zur Analyse von Netzwerkdaten somit ein zentraler Aspekt. Das führt zu der Notwendigkeit, dass es einer klaren und eindeutigen Definition bedarf, sodass etwaigen Missverständnissen vorgebeugt werden kann. Im Kontext der Informationsverarbeitung - beziehungsweise Informatik - ist der Begriff der Echtzeit durch die DIN-ISO/IEC-2382 erfasst und definiert (International Organization for Standardization, 2016).

Aus der Definition wird somit deutlich, dass die Verarbeitungsergebnisse das Ende einer Zeitspanne markieren. Die Zeitspanne selbst ist in dieser Definition also klar in einer Zeiteinheit vorgegeben, sodass das zu erwartende Ergebnis schlussendlich verfügbar sein muss.

Definition 4 (Real-time). *Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.*

Abbildung 2.5: Definition nach International Organization for Standardization (2016)

Das bedeutet, dass die Eigenschaft der Echtzeitanforderung zwar verschiedenen Systemen zugesprochen werden kann, die Zeitspanne aber kein übertragbares Attribut ist.

2.4 Technologien und Plattformen

Dieser Abschnitt beschreibt die wichtigsten und zentralsten Technologien, Plattformen und Sprachen, die in der Entwicklung dieser Arbeit zum Einsatz kommen. Dabei werden die Bezüge zu den zuvor genannten Bereichen von Big Data, Verteilten Systemen und dem Event Processing hergestellt.
Der Anspruch ist es, keine allumfassende Erklärung der jeweiligen Abschnitte anzubieten, da dies den Rahmen dieser Ausarbeitung übersteigen würde. Vielmehr finden die wichtigen Bereiche mit entsprechenden Querbezügen zu den zuvor beschriebenen Grundlagen Erwähnung.

2.4.1 Docker

Bei Docker handelt es sich um eine Plattform zum Erstellen, Verteilen und Ausführen von Software, die durch einen entsprechenden Container verpackt wurde. Ein häufig gebräuchliches Architekturmuster bei Docker sind die Microservices (siehe 2.5.1).
Die Container von Docker, in die Softwareanwendungen verpackt werden, eignen sich nicht nur für den Betrieb der entsprechenden Anwendungen. Ein Vorteil ist zum Beispiel, dass schon zum Zeitpunkt der Entwicklung die spätere Produktionsumgebung bekannt ist und durch die Container genutzt werden kann. Dabei verringern sich die Probleme beim Betrieb einer Software, deren Entwicklungsumgebung sich von der Produktionsumgebung unterscheidet. Das wird gewährleistet durch eine „isolierte Instanz eines Betriebssystems“ (Mouat, 2016).

2.4.1.1 Container

Das Kernkonzept von Docker ist die Containerisierung. Dabei teilen sich die Container mit dem Hostsystem die Ressourcen. Entgegen einer Virtuellen Maschine, auf der eine Anwendung installiert werden kann, abstrahiert keine Hypervisor-Schicht das Host-OS vom Guest-OS. Dies geschieht durch die Docker-Engine. Das hat zur Folge, dass kein komplettes Guest-OS oberhalb der Docker-Engine zugänglich sein muss, sondern die Docker-Engine den Kernel des Host-OS für den Betrieb des Containers zur Verfügung stellt. Trotzdem läuft die Applikation in einer komplett isolierten Umgebung. Alle weiteren Bibliotheken, die für den Betrieb der Anwendung innerhalb des Containers notwendig sind, werden innerhalb des Containers installiert und sind nur dort zugänglich. In der Abbildung 2.6 wird die Trennung ersichtlich ebenso wie der Unterschied zum traditionellen Betrieb innerhalb einer VM.

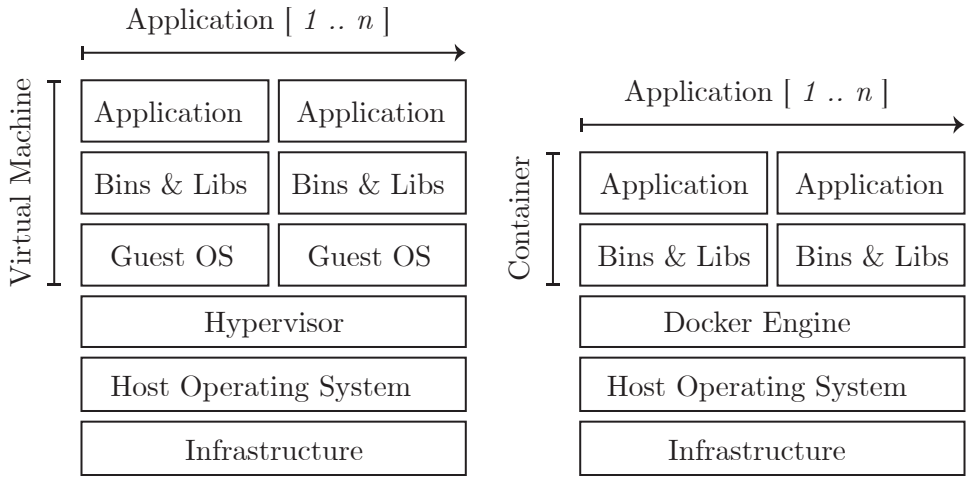

Abbildung 2.6: Virtualisierung gegenüber Containerisierung nach Docker Inc. (2016)

2.4.1.2 Images und Layering

Jeder Docker Container ist ein zur Ausführung gebrachtes Image. Bei der Ausführung bleibt das Image unverändert. Die Änderungen, die zur Laufzeit des Containers am Dateisystem vorgenommen werden, geschehen auf einer eigens dafür erstellten Schicht oberhalb des zugrundeliegenden Images. Durch dieses Verfahren können verschiedene Container auf demselben Image basieren - somit teilen sie es sich.
Images hingegen bestehen aus verschiedenen Schichten, die während des Image-Build-Process geschrieben werden.

2.4.1.3 Dockerfile

Für die Erstellung von Images, die wiederum als Container zur Ausführung gebracht werden, müssen entsprechende Dockerfiles geschrieben werden. Diese werden in einer Docker-eigenen Syntax formuliert. Unterhalb dieser Syntax wird dann aber - je nach Baseimage - die entsprechende Routine zumeist in Bash formuliert. Die Dockerfiles selbst haben kein Schichten-System, sondern sie sind reine Textdateien von zumeist nur ein paar Kilobytes.

2.4.2 Apache Kafka

Von zentraler Bedeutung für die Entwicklung der Plattform dieser Arbeit ist Apache Kafka[2]. Die Open-Source Software wurde - ursprünglich von LinkedIn[3] entwickelt - im Jahr 2012 aber der Apache Software Foundation[4] übergeben. Es handelt sich nach Angaben der Projektverantwortlichen um eine „distributed streaming platform“ (Apache Software Foundation, 2016a).
Zu Beginn der Entwicklung von Kafka stand zuerst die Notwendigkeit eines Systems im Vordergrund ein Messaging System zu entwickeln, das primär für die Akquirierung und das Versenden von Log-Daten gedacht war. Dabei standen die Herausforderungen von großen Datenmengen (siehe Volume in 2.1) und möglichst geringen Latenzzeiten im Vordergrund (Kreps u. a., 2011).
Die Veränderung im Zweck und Nutzen von reiner Log-Aggregation hin zu einer Streaming-Plattform bringt auch eine Veränderung in Hinblick auf die aktuelle Ausrichtung mit sich. So sind drei Fähigkeiten von Kafka zu nennen, die die Plattform heute auszeichnen. Zuerst bleibt die Kernaufgabe eines Messaging Systems erhalten. Kafkas Messaging System arbeitet nach dem Publish/Subscribe Entwurfsmuster. Dabei werden Streams (siehe 2.3.2) von Datensätzen veröffentlicht und abonniert. Des Weiteren findet eine Persistierung der Streams statt und es wird gewährleistet, dass das System nicht komplett zusammenbricht, sofern Fehler bei diesem Prozess entstehen (fault tolerant). Als dritte wichtige Eigenschaft ist das Stream-Processing zu nennen. Hierbei können nicht nur Streams abonniert werden, sondern es existieren zwei Application Programming Interface (API) für deren Verarbeitung.
Neben der Stream-API werden noch drei weitere API's zur Verfügung gestellt. Um Streams generieren zu können, die Daten in das System leiten, existiert die Producer-API. Der entgegengesetzte Fall des Konsumierens von Daten durch Streams wird durch die Consumer-API gewährleistet. Schlussendlich bietet die Connectors-API die Möglichkeit Daten zu Datenbanken zu persistieren.

[2] https://kafka.apache.org/
[3] https://www.linkedin.com/
[4] https://www.apache.org/

Das Kafka System wird als Cluster bereitgestellt beziehungsweise betrieben, sodass jede Komponente auf einem oder mehreren Servern instanziiert werden kann. Der Terminus, der im Kontext von Kafka immer wieder gebräuchlich ist, sind die „Topics“. Diese sind ein Strukturelement, anhand dessen eine Kategorisierung von Daten durchgeführt wird. Das Zusammenwirken aller Systemteile ist in Abbildung 2.7 ersichtlich. Allerdings ist es nur eine schematische Darstellung, die viele wichtige technische Aspekte nicht umfassend abbildet.

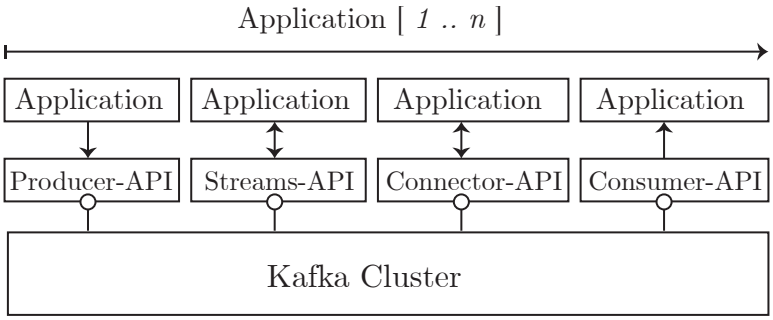

Abbildung 2.7: Kafka Plattform nach Apache Software Foundation (2016a)

2.4.2.1 Streams

Die Streams-API ist die Schnittstelle des Systems, anhand derer Daten auf Basis der Topics konsumiert werden. Dabei werden zwei unterschiedliche Varianten an API's bereitgestellt: eine Low- und High-Level-API. Die Low-Level-API bietet Entwicklern eine „geringe Einstiegshürde“, sodass kleine Proof of concept (POC)'s einfach realisierbar sind (Apache Software Foundation, 2016b). Kafka-Streams hat eine eigene Domain specific language (DSL), in der zwei wichtige Begriffe einer eingehenderen Erklärung bedürfen. Es sind die KStream und KTable.

KStream Ein KStream ist, im Kontext der DSL von Apache Kafka, eine spezielle Form eines Streams. Hierbei ist jeder Datensatz für sich selbst konsistent. Das bedeutet, er hat ein eigenes Datum und ist in seiner Ordnung im Stream eindeutig identifizierbar. Das hat zur Folge, dass zwei inhaltlich identische Datensätze grundsätzlich durch den Zeitpunkt ihres Auftretens zu unterscheiden sind.

KTable Eine KTable unterscheidet sich von einem KStream in der Behandlung von unterschiedlichen Datensätzen. Es liegt die Annahme zugrunde, dass jeder Datensatz grundsätzlich redundant vorkommen kann. Daraus folgt, dass jeder Datensatz nur ein Update des vorherigen

ist und nach diesem Schema nur der letzte Stand - ein Update - verfügbar ist. Vergleichbar ist dieses Verhalten mit dem eines Changelog.

Windowing Das Windowing ist eine Möglichkeit bei Kafka-Streams Daten eines Streams nach Zeitspannen zu gruppieren. Dieses Verhalten ist für die Eigenschaft der Echtzeitfähigkeit (siehe 2.3.3) einer konkreten Anwendung der Plattform von Bedeutung. Dabei stehen drei Arten des Windowing zur Verfügung: *hopping-time*, *tumbling-time* und *sliding*-Windows.

2.4.3 Zookeeper

Apache Zookeeper ist ein Open-Source Server, dessen primäre Aufgabe in der Koordination verteilter Anwendungen liegt. Dabei zeichnet sich Apache Zookeeper dadurch aus, dass er entweder als eigenständiger Server oder aus mehreren Instanzen auf einem Cluster betrieben wird. Angesprochen von außen wird der Zookeeper dann über das Quorum an Servern.
Die zentrale Eigenschaft der verteilten Server der jeweiligen Zookeeper-Instanzen sowie die Aufgabe etwaige andere verteilte Anwendungen zu koordinieren, ist beispielhaft für die Verteilten Systeme (siehe 2.2). Vier Ziele stehen bei Zookeeper im Vordergrund. Das ist zum einen ein geteilter und verteilter Namespace, anhand dessen die Rezipienten sich koordinieren. Zookeeper nutzt kein Filesystem, sondern nur den Hauptspeicher für die Erfüllung dieser Aufgabe. Dadurch können ein hoher Durchsatz sowie geringe Latenzen erzielt werden. Das ist Ausdruck der **Einfachheit** von Zookeeper. Die **Replikation** von Zookeeper selbst ist das zweite Ziel. Hierbei ist das Nutzen eines Clusters ausschlaggebend. Das Versehen jeder Aktion mit eindeutigen Nummern gewährleistet, dass High-Level Abstraktionen möglich sind. Das ist der Aspekt der **Ordnung**. Schlussendlich ist Zookeeper **schnell**, was sich in Anwendungsszenarien bewahrheitet, in denen ein Schreib/Leseverhältnis von *10:1* vorherrscht (Apache Software Foundation, 2016c).

2.4.4 Zusammenfassung

Abschließend werden hier ein paar weitere Plattformen, Frameworks sowie Bibliotheken aufgelistet, die für die eigene Plattform zum Einsatz kommen. Die Tabelle 2.1 ist aber keine umfassende Auflistung, sondern sie ist vielmehr Abschluss dieses Abschnitts mit dem Ziel eines besseren Verständnisses dafür, auf welcher Basis die Entwicklung stattfindet.

Name	Typ	Beschreibung	Präsenz
Angular2	Framework	Framework zur Erstellung von Single-Site-Pages mit dem Fokus auf einem Framework für sowohl Desktop-Anwendungen als auch Mobilen-Anwendungen	angular.io
Node.js	Plattform	Plattform zum Entwickeln in Javascript im Back-End auf Basis der Google-V8 Engine	nodejs.org
MongoDB	Datenbank	Schema-freie NoSQL Datenbank auf Basis der JavaScript Object Notation	mongodb.org

Tabelle 2.1: Auflistung von weiteren Plattformen, Frameworks, Bibliotheken

2.5 Systemarchitektur

Dieser Abschnitt beschreibt einige wichtige Paradigmen und Architekturmuster, die für die Realisierung der Plattform von Bedeutung sind. Dabei findet eine erste Beschreibung der Microservicearchitektur statt, die im Kontext von Docker stark an Bedeutung gewonnen hat. Anschließend werden die Event-Driven Architecture sowie die Service-Oriented Architecture beschrieben. Letztere gewinnt zusehends an Bedeutung im Zusammenhang mit sogenannten Cloud-Plattformen.

2.5.1 Microservicearchitektur

Die Architektur von Softwaresystemen anhand von Microservices ist ein in den letzten Jahren stark an Popularität gewinnendes Entwurfsmuster (Sill, 2016).
Allgemein ist die Idee hinter Microservices, dass sie möglichst kleine und unabhängige Aufgaben erledigen, wobei sie untereinander möglichst entkoppelt sind. Dabei ist ein Charakteristikum, dass sie grundsätzlich verteilbar, skalierbar und testbar sind. Weiterhin haben sie zumeist nur eine Verantwortlichkeit beziehungsweise eine Aufgabe in einem Gesamtsystem zu erledigen (Thönes, 2015). Die Strukturierung von Anwendungssoftware nach diesem Muster steht einer monolithischen Architektur entgegen. Im Kontext von Verteilten Systemen haben die Microservices enorm wegen dem Aufkommen der Containerisierung verschiedener Softwarekomponenten an Popularität gewonnen. Hierbei ist Docker[5] sicherlich die prominenteste Plattform.

[5] https://www.docker.com/

2.5.2 Event-Driven Architecture

In der ereignisgesteuerten Softwarearchitektur werden zumeist drei verschiedene Arten von Prozessen unterschieden. Hierzu zählen die beiden zuvor beschriebenen Arten des ESP und CEP. Eine weitere Prozessart ist das Simple Event Processing (SEP), bei dem Prozesse direkt einfache Ereignisse auslösen.
Einer ereignisgesteuerten Architektur liegt zugrunde, dass alle Komponenten des Systems durch entsprechende Ereignisse aufeinander reagieren können. Das heißt, beim Auftreten eines Ereignisses werden alle Komponenten über das Stattfinden informiert und evaluieren das Ereignis, sodass darauf dann eine Aktion erfolgen kann oder auch nicht (Michelson, 2006).
Systeme wie die der Event-Driven Architecture fußen auf vier Ebenen. Die Gesamtheit beschreibt das Auftreten einer Ebene bis hin zur etwaigen Reaktion auf das Ereignis. Die Ebenen werden wie folgt beschrieben:

2.5.2.1 Event Generator

Der Ereignis-Produzent (engl. event generator) ist für die Generierung von Ereignissen zuständig. Hierbei können die Ereignisse verschiedenen Ursprungs (verschiedene Ereignis-Generatoren) sein. Schlussendlich muss aber gewährleistet werden, dass ein eigenes Format der Ereignisse allen gemeinsam ist.

2.5.2.2 Event Channel

Der Ereignis-Kanal hat die Zuständigkeit der Bereitstellung der Ereignisse anhand eines passenden Speicher- oder Transportmediums: Kurzum er ist der Ereignisträger. Das bedeutet, dass auf dieser Ebene die Schnittstelle angesiedelt ist, die eine Kommunikation mit der dritten Schicht der Event Processing Engine ermöglicht.

2.5.2.3 Event Processing Engine

Diese Ebene umfasst die konkrete Klassifikation und Verarbeitung der Ereignisse. Durch ein entsprechendes Regelwerk kann anhand implementierter Programmlogiken eine Verarbeitung stattfinden.

2.5.2.4 Downstream Event-driven Activity

Die letzte und vierte Ebene umfasst das Verhalten aufgrund eines aufgetretenen Ereignisses; also die Reaktion, die auf Grundlage der vorangegangenen Evaluation und Verarbeitung der dritten Ebene erfolgt.

2.5.3 Service-Oriented Architecture

Die Service-Oriented Architecture (SOA) ist ein Muster zur Strukturierung und Vereinigung verschiedener Dienste einer IT-Infrastruktur. Dabei findet dieser Ansatz Verbreitung im Bereich des Cloud Computings und hat sich dort schon etabliert (Barbier und Recoussine, 2014). Ein Ziel ist es unter anderem, durch die Bildung verschiedener Services eine Basis zu schaffen, auf deren Grundlage neue Produkte entstehen (Papazoglou, 2003).
Die Architektur sieht vor, dass ein Service eine Funktionalität nach außen hin anbietet, ohne dass das nutzende System eine weitere Kenntnis über den genauen Verarbeitungsprozess im Inneren des Services hat. Das führt dazu, dass verschiedene IT-Dienste im Zusammenspiel einen Service definieren, wobei nicht ausgeschlossen wird, dass die Services untereinander aufeinander aufbauen.
Auf diesem Wege wird aus ökonomischer Sicht versucht eine Kostenreduktion zu erreichen, da etwaige Services wiederverwendet werden können ohne eine teilweise oder komplette Neuentwicklung zu durchlaufen. Hierbei wird die Bedeutung von standardisierten Schnittstellen ersichtlich, da das die einzige Möglichkeit der Kommunikation/Interaktion mit dem Service ist.

2.6 Zusammenfassung

Das Kapitel Grundlagen hat zum Ziel die Themenbereiche, die wichtig für die Ausarbeitung und die Plattform sind, eingehender zu beschreiben. Am Anfang steht eine Auseinandersetzung mit dem Thema Big Data. Es wird Versucht, eine Begriffsschärfung herbeizuführen, indem die Entwicklung des Begriffs in ihrem zeitlichen Kontext beleuchtet wird. Im weiteren Verlauf findet eine nicht-technische Auseinandersetzung mit dem Thema der Digitalisierung sowie des Datenschutzes statt. Diese Abschnitte schildern den Wandel in der Informationstechnologie und unterstreichen damit die Relevanz des Arbeitsthemas. Abschließend wird Big Data im Zusammenhang mit einigen beschriebenen Szenarien aus ökonomischer Sicht betrachtet.
Daran schließt sich der Bereich der Verteilten Systeme an, in dessen Verlauf das Themengebiet des Cloud Computings ein erstes Mal behandelt wird. Außerdem werden die verschiedenen Betriebsarten beschrieben, die auch eine Relevanz für die eigene Plattform haben. Die Microservices sind zwar originär nicht unmittelbar dem Themenbereich der Verteilten Systeme zuzuordnen, aber sie passen inhaltlich gut zu deren Domäne. Grund hierfür ist, dass die Architektur entsprechender Systeme zumeist eine ganz eigene Herausforderung darstellt und Microservices dieser begegnen kann. Im Anschluss daran findet das Cluster Computing Erwähnung.
Der Themenkomplex des Event Processings bildet einen zentralen Bereich der Grundlagen.

Dabei werden die grundlegenden Teilgebiete des Complex Event Processings sowie Event Stream Processings genauer beschrieben. Hinzu kommt noch die Auseinandersetzung mit dem Begriff der Echtzeit, da dieser ein zentraler Bestandteil der Arbeit ist. Im Weiteren werden dann unterschiedliche Architekturen beschrieben, die häufig im Zusammenhang des CEP wiederzufinden sind.
Den Abschluss finden die Grundlagen mit einer ersten Aufzählung der verschiedenen Technologien, Frameworks und Plattformen, die für die eigene Plattform zum Einsatz kommen. Hier ist primär Docker hervorzuheben, da die Plattform auf der Containerisierung fußt.

3 Analyse

In diesem Kapitel werden die Herausforderungen und Aufgaben beschrieben, die für die Konzeption und Entwicklung der Plattform zur Echtzeit-Netzwerk-Datenanalyse von Bedeutung sind.
Es ist wichtig, vorab zu erwähnen, dass die Entwicklung aus eigenem Bestreben geschieht und nicht durch ein Unternehmen forciert wurde. Das heißt, die Anforderungen und Notwendigkeiten in diesem Kapitel resultieren aus der eigenen Analyse und sind nicht Teil einer Spezifikation eines Kunden.
Zu Anfang wird der Kontext der Arbeit spezifiziert um so die Ausrichtung der Plattform zu verdeutlichen. Im Anschluss daran werden wichtige Begriffe dieser Arbeit beziehungsweise die Terminologie des Fachbereichs eingehender erläutert. Das Stream Processing (siehe 2.3.2) ist ein weiterer Bestandteil des Analyse-Kapitels. Es stehen weniger die Grundlagen im Mittelpunkt als vielmehr das Zusammenwirken der Komponenten und deren Einfluss im Bereich der Network-Analytics. In diesem Zusammenhang wird die Domäne der Plattform anfänglich genauer beschrieben. Anschließend werden dann die Anforderungen genauer spezifiziert, die an die Plattform gestellt werden. Diese sind nach funktionalen, technischen und nicht-funktionalen Anforderungen strukturiert.
Die verschiedenen Anwendungsszenarien werden in einem eigenen Abschnitt behandelt. Es steht im Vordergrund, dass etwaige Einsatzbereiche eingehender Erwähnung finden. Auf diese Weise wird die zuvor beschriebene Domäne konkreter Anwendungen auf der Basis der Plattform verdeutlicht.
Das Kapitel findet seinen Abschluss in einer Auflistung verschiedener bereits existierender Arbeiten und Projekte. Vorrangig geht es darum, Analogien zwischen Teilkomponenten der eigenen Plattform und anderen Systemen zu finden. Außerdem werden andere Möglichkeiten benannt um eine Abgrenzung zu den eigenen Anwendungsszenarien aufzuzeigen und gegebenenfalls weitere Plattformen für andere Vorhaben benennen zu können.

3.1 Plattformkontext

Der Kontext der Plattform wird aus Abbildung 3.1 ersichtlich. Drei verschiedene Akteure sind zu unterscheiden. Zum einen muss das System zur Ausführung gebracht werden. Der Akteur *Betreiber* ist hauptverantwortlich für das System beziehungsweise für den Betrieb der Plattform. Da die Plattform viele verteilte Komponenten auf einem Cluster umfasst, ergibt sich daraus die Notwendigkeit einer zentralen Persion für die Verwaltung dieser Komponenten. Außerdem ist eine zusätzliche Entwicklung von Verarbeitungskomponenten notwendig. Diese können auch als „Worker“ im Kontext der Plattform gesehen werden. Diese Worker sind nur ein Teil einer konkreten Anwendung, die auf der Basis der Plattform ausgeführt wird. Um Daten in das System einzuleiten sind Data Producer notwendig. Diese sind auf dem zu analysierenden System anzusiedeln und schneiden definierten Netzwerkverkehr mit. Die beiden Teilkomponenten Data Producer und Data Processor sind im Verantwortungs- beziehungsweise im Entwicklungsbereich des Akteurs *Entwickler* angesiedelt. Der *Konsument* oder *Rezipient* ist der dritte Beteiligte an der Plattform. Dieser hat keine Schnittmenge in seinem Aufgaben- oder seinem Verantwortungsbereich mit den anderen beiden Akteuren. Er ist alleiniger Empfänger von Informationen, die als Resultat am Ende der Verarbeitung von Daten der Plattform stehen.

3.2 Terminologie

Das Kapitel Analyse (3) beschreibt die Plattform zur Analyse von Netzwerkdatenverkehr. Hierbei finden bestimmte Begriffe immer wieder Verwendung, die für das Verständnis der weiteren Kapitel eine wichtige Voraussetzung darstellen. Diese Begriffe und Akronyme werden im folgenden Abschnitt Terminologie eingehender erklärt und definiert.
Zuerst ist das Verständnis der Data Producer wichtig. Diese verbinden eine Datenquelle mit der Plattform zur Verarbeitung dieser Daten. Sie stehen am Anfang der Verarbeitungskette und der Datenanalyse und leiten somit zumeist ungefiltert alle Daten in das System ein. Sie sind einzelne Komponenten, die eine geringe Kopplung mit der Plattform aufweisen, da sie außerhalb des Gesamtkomplexes angesiedelt sind. Das heißt zum Beispiel: Auf einem autarken System sind nur die Producer angesiedelt und nutzen eine Schnittstelle der Plattform.

Nachdem Daten zur Verfügung stehen beziehungsweise Datenstreams erstellt wurden, steht der Prozess der Verarbeitung und Analyse an. Hierbei sind die Data Processor wichtig. Sie bilden die Verarbeitungseinheit der Plattform, die nicht eindeutig nur einer Art der Verarbeitung zuzuordnen sind. Vielmehr übernehmen sie vielfältige Verarbeitungsaufgaben. Am Ende muss nicht ein konkretes Ergebnis mit Aussagekraft stehen, sondern es können auch Teilschritte oder Zwischenergebnisse entstehen.

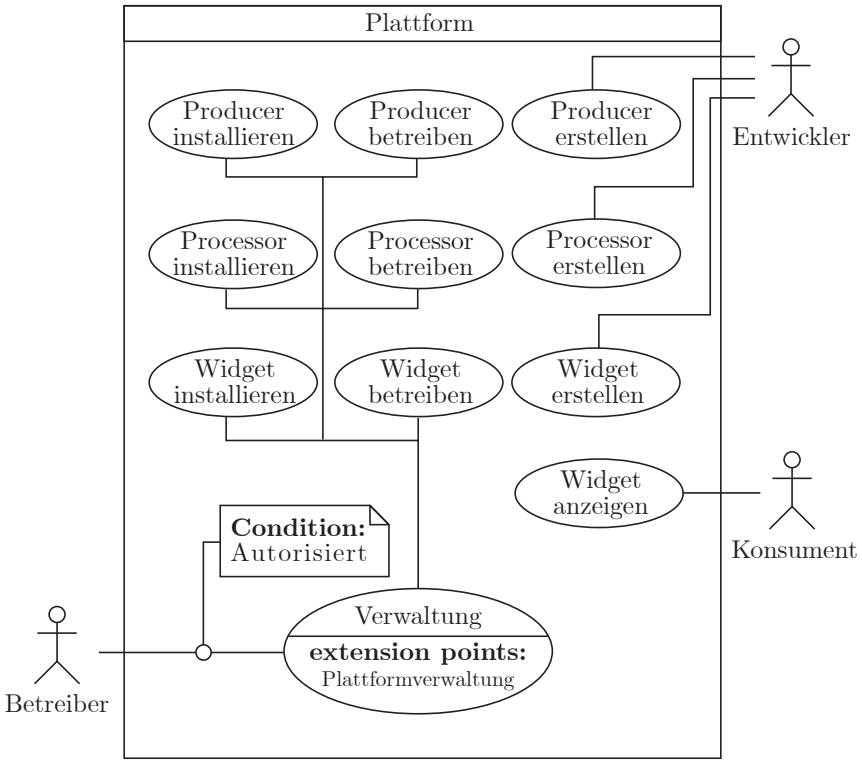

Abbildung 3.1: Kontext der Plattform als Anwendungsfalldiagramm (UML)

Auf der dritten Ebene oder auch am Ende des Verarbeitungsprozesses steht die Visualisierung von Verarbeitungsergebnissen. Diese Visualisierung erfolgt durch zu entwickelnde Widgets. Gemeint ist nicht nur eine graphische Aufbereitung von konkreten Ergebnissen, sondern vielmehr eine kontextbezogene Aufbereitung des Anwendungsziels.

3.3 Stream Processing

Beim Stream Processing ist ein kontinuierlicher Strom von Daten gegeben. Die einzelnen Datensätze dieses Stroms unterliegen einer Ordnung gemäß dem Zeitpunkt ihres Auftretens. Daraus folgt, dass das punktuelle Datenaufkommen tendenziell gering ist. Dieser Annahme liegt zugrunde, dass eine gewisse Gleichverteilung des Datenaufkommens gegeben ist. Im Fall der Analyse von Netzwerkdaten ist eine Gleichverteilung dann gegeben, wenn etwaige Services zur Kommunikation per TCP/IP zum Beispiel nicht erst auf Interaktion einer natürlichen Person Daten abfragen oder senden.

Definition 5 (Data Producer, *Producer*)**.** *Data Producer (kurz: Producer) sind eigenständige Komponenten, die nur eine Schnittstelle der Plattform nutzen und der Plattform über diese Schnittstelle Daten zuführen. Sie weisen außer über die Schnittstelle keine Kopplung zu der Plattform auf.*

Abbildung 3.2: Definition Data Producer

Definition 6 (Data Processor, *Processor*)**.** *Data Processor (kurz Processor) sind Komponenten, die im System die Verarbeitung von Datenstreams übernehmen und etwaige Ergebnisse wiederum in das System leiten können.*

Abbildung 3.3: Definition Data Processor

Beim Prozess der konkreten Verarbeitung der Datensätze wird dann auf spezielle Datensätze gewartet, die zu Ereignissen führen, auf die wiederum andere Prozesse warten. Das bedeutet zum Beispiel, dass der Versand eines Datenpakets von Host A zu Host B von zwei unabhängigen Daten-Prozessoren von Bedeutung ist. Das ist dann der Fall, wenn Prozessor A für eine reine Lokalisation von Quell- und Zieladresse (Host A und Host B) zuständig ist und Prozessor B wiederum eine reine Verarbeitung des Datenpaketinhalts übernimmt. Kontextuell betrachtet sind die beiden Stream Prozessoren somit unabhängig, benötigen aber beide Zugriff auf ein und dasselbe Datenpaket. Unter dem Gesichtspunkt des Prinzips der Separation of concerns (SoC) ist diese Verteilung wünschenswert. Hingegen ist die Orchestrierung der Events im Prozess der Verarbeitung dadurch schwieriger.
Bei dem beschriebenen Problem im Umgang mit Datenpaketen lassen sich zwei wichtige Eigenschaften im Kontext von Apache Kafka nennen, die bei der beschriebenen Herausforderung Abhilfe schaffen. Zum einen ist es die Fähigkeit, während oder nach abgeschlossenem Stream-Verarbeitungsprozess Daten wieder in entsprechende „Topics“ (siehe 2.4.2) zu schreiben und/oder zu lesen. Weiterhin ist auch die „multi-subscriber” Eigenschaft zu nennen. Diese ermöglicht es, dass ein Stream von mehreren Data Processors gelesen werden kann. Auf diese Weise herrscht eine *1 zu n* Beziehung zwischen Data Streams und Data Processors.

Definition 7 (*Widget*). *Widgets sind Komponenten mit der Aufgabe der visuellen Aufbereitung von Verarbeitungsergebnissen. Sie dienen der Verdeutlichung gewonnener Erkenntnisse.*

Abbildung 3.4: Definition Widget

3.4 Network Analytics

Die zentralen Anwendungsbereiche der Plattform sind die Akquise, Verarbeitung und Analyse von Netzwerkdaten. Primär ist der Bereich der Plattform wichtig, der am Anfang der Verarbeitungskette steht: kurzum die Datenakquise. Es ist hervorzuheben, dass die Datenquellen nicht direkt ein entsprechendes Netzwerk-Interface sein müssen. Das bedeutet, dass kommandozeilenbasierte Tools wie *tcpdump*[1], *tshark*[2] oder *dumpcap*[3] nicht ausschließlich als Datenproduzenten infrage kommen. Vielmehr ist genau am Beginn der Verarbeitung eine größere Bandbreite an Möglichkeiten der Datenakquise denkbar.
Zum einen bietet Google Chrome - beziehungsweise das zugrundeliegende Open-Source-Projekt Chromium[4] - mittlerweile eine Schnittstelle an, anhand derer der Datenverkehr des Webbrowsers mitgeschnitten werden kann. Auf diesem Weg und über andere Möglichkeiten ist es auch denkbar, Daten der Plattform zur Verfügung zu stellen. Die Chromium Schnittstelle ist aber kein Alleinstellungsmerkmal, das nur ein Webbrowser hat, sondern sie lässt sich auch bei weiteren Webbrowsern (Mozilla Developer Network, 2016) finden. Diese Art des Andockens einer Datenquelle beziehungsweise eines Streams ist gegenüber dem Mitschneiden am Interface des Host-Rechners eher als High-Level-Ansatz zu sehen.
Die verschiedenen Möglichkeiten eine Datenquelle mit der Plattform zu verbinden setzt natürlich eines voraus: eine Systemkomponente, die diese Daten annimmt und sie für die Plattform bereitstellt. Hierbei ist in der Analyse zu beachten, dass diese Komponente nur eine plattformspezifische Kopplung zu der angebotenen Schnittstelle aufweisen sollte und nicht dem Gesamtkomplex des Systems gegenüber. Das heißt, die Komponente sollte betriebssystemübergreifend mit der Schnittstelle der Plattform zur Verarbeitung der Daten kommunizieren können und somit eine leichte Kopplung zur Plattform selbst aufweisen.
Das Format, in dem die Daten bereitgestellt werden, ist bei der Analyse der Plattform wichtig.

[1] http://www.tcpdump.org/
[2] https://www.wireshark.org/docs/man-pages/tshark.html
[3] https://www.wireshark.org/docs/man-pages/dumpcap.html
[4] https://www.chromium.org/

Hierbei ist das Packet Capture (PCAP)-Format im Kontext von Wireshark ein häufig gebräuchliches Format zur Speicherung von Netzwerkpaketmitschnitten. Im klassischen Fall dient es aber der langfristigen Persistierung der Daten und nicht der Verarbeitung in Streams. Weitere Formate wären Packet Details Markup Language (PDML), eine auf Extensible Markup Language (XML) basierte Datenstruktur. Außerdem sind auch eigene Definitionen in Form von JavaScript Object Notation (JSON) denkbar, sofern eine einheitliche Objektstruktur definiert wird.

3.5 Anforderungsspezifikation

In diesem Abschnitt werden zuerst die Bereiche herausgearbeitet, die alle konkreten Anwendungen auf der Basis der Plattform gemeinsam haben. Auf diesem Wege werden die Gemeinsamkeiten herausgestellt und spezifiziert, sodass am Ende des Prozesses eindeutige Anforderungen herausgestellt werden können, die die Plattform gewährleisten soll.
Zuerst erfolgt eine Domänenanalyse, in der Gemeinsamkeiten gefunden werden. Die dadurch gewonnenen Erkenntnisse werden anschließend in Anforderungen überführt, wobei diese nach funktionalen, technischen und nichttechnischen Anforderungen gegliedert sind.

3.5.1 Domänenanalyse

Die Domänenanalyse gliedert sich in zwei Teilbereiche: zum einen in die Domäneneingrenzung. Diese hat zum Ziel den Themenkomplex der Arbeit einzugrenzen. Dabei werden die Entitäten beschrieben, die im Gesamtkomplex der Echtzeit-Netzwerk-Datenanalyse beziehungsweise der konkreten Anwendungen Allgemeingültigkeit besitzen. Im Anschluss daran findet die Domänenmodellierung statt.

3.5.1.1 Domäneneingrenzung

Die Abschnitte Stream Processing (3.3) und Network Analytics (3.4) beschreiben aus ihren jeweiligen Bereichen die Merkmale, die für die Echtzeit-Netzwerk-Datenanalyse wichtige Faktoren sind. Dabei findet bereits eine erste Eingrenzung statt.

Echtzeit Der Charakter der Echtzeitfähigkeit der Plattform beziehungsweise der konkreten Anwendungen, die auf Basis der Plattform entwickelt werden, ist eine erste Einschränkung in Hinblick auf die Ausrichtung der Plattform.

Datastream Ein kontinuierlicher Fluss an Daten ist außerdem eine Einschränkung gegenüber anderen Datenverarbeitungssystemen, die auch das Batch-Processing bedienen.

Visualisierung Die Visualisierung gewonnener Ergebnisse ist als weitere Einschränkung zu sehen. Dabei werden im Regelfall keine Daten langfristig persistiert, sondern das Ziel ist es, vielmehr auf aktuelle Erkenntnisse des analysierten Datenstroms zu reagieren.

3.5.1.2 Domänenmodellierung

In diesem Abschnitt werden nun die Entitäten beschrieben, die nach der Eingrenzung (3.5.1.1) eine Allgemeingültigkeit für konkrete Anwendungen auf der Basis der Plattform haben.

Kontext Zuerst einmal steht allem voran der zu analysierende Kontext. Hierbei muss grundsätzlich definiert sein, welcher Sachverhalt analysiert werden soll und wie dies zu erreichen ist: also eine klar definierte Zielsetzung.

Zeitraum Der Zeitraum definiert die Zeit, über die eine Analyse geschehen soll. Bei einer konkreten Eingabequelle haben die Stream-Ereignisse auch Einfluss auf den Zeitraum.

Quelle Sofern ein Kontext vorhanden ist, muss die Datenquelle entsprechend gewählt werden. Sie steht in Abhängigkeit zum Kontext. Es muss sichergestellt sein, dass die zu erwartenden Daten auch über die Datenquelle bezogen werden können.

Prozessor Je nach Kontext und Datenquelle müssen entsprechende Prozessoren entwickelt werden, die zu dem Ziel der Analyse beitragen. Sie können leicht- oder oder schwergewichtige Arbeiten durchführen.

Ergebnis Die Quintessenz jeglicher Anwendungen sind entsprechende Artefakte. Diese sollten visuell darstellbar oder zumindest final sein - also keiner weiteren Bearbeitung bedürfen.

3.6 Anwendungsszenarien

In diesem Abschnitt werden zwei exemplarische Szenarien beschrieben, bei denen die Plattform zum Einsatz kommen kann. Hierbei stehen weniger technische Aspekte im Mittelpunkt der Betrachtung als vielmehr die konkret zu tätigenden Analysen. Die hier folgenden Beispiele sind auch als Einleitung mit Blick auf die Kapitel 5 und 6 zu verstehen.

3.6.1 Werbevermarktung

Das erste Szenario befasst sich mit dem Platzieren von Werbeanzeigen im Bereich des Online-Marketings. In der Sparte des Vertriebs von Werbeflächen auf diversen Webseiten existieren verschiedene Anbieter, die es ihren Kunden ermöglichen, die Werbeflächen zu mieten. Nun liegt es im Interesse des Werbetreibenden, dass sowohl sein Kunde - also das Unternehmen, das etwas bewerben möchte - als auch der Konsument der Werbung zufriedengestellt werden sollen.
Im konkreten Szenario wird angenommen, dass das werbende Unternehmen ein Aftershave online platzieren möchte. Es erhofft sich durch die Vermarktung einen höheren Absatz des Produkts um sich damit gegenüber seinen Konkurrenten besser am Markt platzieren zu können. Der Vermarkter der Werbeanzeige möchte diese Werbung des Unternehmens nun derart vermarkten, dass der gewünschte höhere Absatz des Kunden erzielt werden kann. Hierbei wird die Werbung zumeist nicht direkt auf den Servern der Webseiten gehostet, die die Werbung auch präsentieren. Auf diesem Weg existieren Strukturen, über die die gesamte Werbung ausgesteuert werden kann.
Der Konsument der Werbung bekommt im Normalfall nicht direkt mit, wie der infrastrukturelle Aufbau im Hintergrund der Webseite beschaffen ist. Somit sieht er nur die Aftershave-Werbung eingeblendet. Trifft die Werbung auf sein Interesse oder weckt sie ein Bedürfnis bei ihm, dann klickt er auf die Werbefläche und erwirbt gegebenenfalls das Produkt.

3.6.1.1 Fazit

Zusammenfassend lassen sich bei dem zuvor beschriebenen Szenario verschiedene Sachverhalte konstatieren. Zum einen liegt es grundsätzlich im Interesse des Werbeflächenvermarkters, dass sowohl auf der Seite des Werbenden als auch auf der Seite des Konsumenten Zufriedenheit herrschen sollte, sofern es zum Erwerb des Produktes gekommen ist.
Außerdem liegt es im Interesse des Vermarkters, im Erfolgsfall weitere Werbeflächen an das werbende Unternehmen zu vermitteln, da auf diesem Weg sein Geschäftsprinzip funktioniert. Dabei wird der Vermarkter alles rechtlich Zulässige und technisch Mögliche unternehmen um bei zukünftigen Abschlüssen noch erfolgreicher zu sein. Vom Standpunkt des Konsumenten aus lassen sich auch nicht ausschließlich ökonomische Faktoren finden, die ihn dazu bewegen, einer Art von Werbung zuzusagen oder sie abzulehnen. Siehe hierzu auch „Datenschutz“ (2.1.4) und „Der Wert von Daten“ (2.1.5).
Dieses Szenario dient als einleitende Analyse für die später folgenden konkreten Anwendungen. An dieser Stelle wird aufgezeigt, dass mitunter bewusste Kaufentscheidungen bei

vorhandenem Wissen über die Vermarktungswege abweichend gegenüber intransparenten Vermarktungswegen ausfallen können.

3.6.2 Anonymisierung

Im zweiten Szenario der Analyse geht es um die Möglichkeit der Anonymisierung im Internet. Es gibt es viele verschiedene Ebenen und Möglichkeiten der Anonymisierung, die in der Regel erst gemeinsam zum Ziel der Anonymität führen. In diesem Szenario sind vorrangig die beiden Protokolle Hypertext Transfer Protocol (HTTP) und Hypertext Transfer Protocol Secure (HTTPS) im Einsatz.
Betrachtet wird in diesem Szenario eine Person (*Alice*), die Inhalte über das Internet verbreiten möchte. Es wird vorausgesetzt, dass Informationen, die *Alice* besitzt, in dem Land von strafrechtlicher Relevanz sind, in dem sich *Alice* aufhält. Nun gibt es eine Gruppe von Personen, die Interesse an den Daten von *Alice* hat. Gemeinsam ist allen beteiligten Personen das Wissen, dass die Inhalte verboten sind. Nehmen wir eine Person aus der Gruppe und nennen sie *Bob*. Beiden gemeinsam ist, dass sie wechselseitig keine Kenntnis voneinander haben und sie somit die Identität des jeweils Anderen nicht kennen. Nun ergibt sich aus der strafrechtlichen Relevanz der Inhalte, dass beide Seiten unerkannt bleiben möchten. Hierbei könnte *Alice* zum Beispiel die Informationen auf einem Server laden, der physikalisch nicht in dem Land steht, in dem sich *Alice* aufhält. Ein Austausch der Inhalte über diesen Server wäre dann möglich. Um die Identität beim Zugriff auf den Server geheim zu halten, könnte das Tor-Netzwerk für den Zugriff auf den Server verwendet werden. Auf diesem Weg könnten sowohl *Bob* als auch *Alice* ihre jeweiligen Identitäten schützen und trotzdem an die Inhalte auf dem Server per HTTP kommen.

3.6.2.1 Fazit

Im Fall des Tor-Netzwerks befindet sich am Ende jeder Verbindung immer ein Knoten, der den Netzwerkdatenverkehr aus dem Tor-Netz in das „offene“ Internet leitet. Dieser Tor-Exit-Node kann theoretisch von jeder Person betrieben werden. In der Struktur des Netzwerks sind zwar die Identitäten geschützt, aber im Fall von HTTP sind alle Datenpakete unverschlüsselt. Dieses Szenario zeigt, dass am Punkt des Tor-Exit-Nodes eine Stelle vorhanden ist, an der die Häufigkeit von Aufrufen verschiedener Ressourcen mitgeschnitten werden könnte. Auf diese Weise kann der Traffic zu bestimmten Servern untersucht werden und im Fall von strafrechtlich relevanten Inhalten können weitere Untersuchungen angestellt werden.
Das Szenario beschreibt den Aufruf von einer Person *Bob* mit seinem entsprechenden Exit-Node. Jede weitere Person würde wahrscheinlich eher einen anderen Exit-Node nutzen (bzw.

zugewiesen bekommen) als den von *Bob*. Über die Zeit verteilt und bei langer Laufzeit steigt aber die Aussagekraft jeglicher Analysen.

3.7 Anforderungen

Dieser Abschnitt beschreibt die Anforderungen, die an die Plattform gestellt werden. Dabei ist eine konkrete Anwendung, die auf der Basis der Plattform entwickelt wird, ausschlaggebend für die verschiedenen Arten von Anforderungen. Unterschieden wird im Folgenden zwischen funktionalen, technischen und nicht-funktionalen Anforderungen. Jede Anforderung wird mit einem Kürzel versehen, sodass diese leicht zugeordnet werden kann. Dabei ist das Schema wie folgt:

P – [FA | TA | NFA] – [0-9]

Abbildung 3.5: Schema Anforderungsbezeichner

Bei dem Schema in Abbildung (3.5) beschreibt das „**P**“ die Plattform, für die die Anforderungen definiert werden. Die darauf folgenden Akronyme „**FA**“, „**TA**“ und „**NFA**“ stehen für funktionale, technische und nicht-funktionale Anforderungen. Der abschließende Index wird dann fortlaufend inkrementiert.

3.7.1 Funktionale Anforderungen

Dieser Abschnitt beschreibt verschiedene funktionale Anforderungen, die an die Plattform gestellt werden. Das Akronym „FA“ steht für diese Art der Anforderungen.

P-FA-1
Die Plattform bietet verschiedene Möglichkeiten einen Netzwerkdatenverkehr unmittelbar für eine laufende Anwendung bereitzustellen.

P-FA-2
Das Mitschneiden von Netzwerkverkehr geschieht auf der Basis von Datenstreams, die dafür generiert werden.

P-FA-3
Datenstreams werden anhand von vordefinierten Schlüsseln identifiziert um diese in etwaige Daten Prozessoren zu laden.

P-FA-4
Die Plattform kann ohne konkrete Daten Prozessoren betrieben und zur Laufzeit mit weiteren Daten Prozessoren ergänzt werden.

P-FA-5
Für eine Visualisierung von Verarbeitungsergebnissen wird eine zentrale Schnittstelle angeboten, über die Ergebnisse abgerufen werden können.

3.7.2 Technische Anforderungen

In diesem Abschnitt werden die Anforderungen benannt, die die technischen Aspekte der Plattform beschreiben. Hierbei findet das Akronym „TA“ im Bezeichner Verwendung.

P-TA-1
Die Plattform soll verteilt betrieben werden. Dabei sollte möglichst jede Teilkomponente redundant vorkommen können.

P-TA-2
Die Plattform sollte in der Gesamtheit skalierbar sein. Das heißt, das Hinzufügen von 20 Data Producern sollte bei genügend vorhandenen Cluster-Knoten nicht die Gesamtperformance beeinträchtigen.

P-TA-3
Es wird eine Schnittstelle für das Einleiten von Daten in das System angeboten. Die Möglichkeiten, diese Schnittstelle anzusprechen, sollten vielfältig sein.

P-TA-4
Es wird eine Schnittstelle für das Abrufen von Daten aus dem System angeboten. Diese Schnittstelle dient der Bereitstellung von Verarbeitungsergebnissen für das Front-End.

P-TA-5
Der Ausfall einzelner Teilkomponenten sollte nicht das Gesamtsystem derart in Mitleidenschaft ziehen, dass andere Systemkomponenten dadurch ausfallen (fault tolerance).

P-TA-6
Jegliche Komponenten im Verbund der Plattform sollten der Strukturierung von Komponenten nach Microservices (siehe 2.5.1) entsprechen.

3.7.3 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen sind zumeist die Qualitätseigenschaften, die das System aufweisen soll. Hierbei findet das Akronym „NFA“ im Bezeichner Verwendung.

P-NFA-1
Innerhalb von fünf Sekunden sollen Verarbeitungsergebnisse visuell abrufbar sein um dem Benutzer direkt eine Rückmeldung geben zu können.

P-NFA-2
Eine konkrete Anwendung sollte innerhalb einer Minute zum Einsatz gebracht werden können - somit muss die Inbetriebnahme kurzfristig möglich sein.

P-NFA-3
Data Processors sollten eine existierende Anwendung einfach erweitern können. Dabei sollten die Programmteile leicht hinzugefügt werden können.

P-NFA-4
Entwickler sollten einfach Daten Prozessoren implementieren können ohne den Gesamtkomplex des Systems kennen zu müssen.

P-NFA-5
Konsumenten sollten über ein Graphical User Interface (GUI) visuell aufbereitete Ergebnisse abrufen können ohne ein tieferes technisches Verständnis haben zu müssen.

P-NFA-6
Die Inbetriebnahme sollte ohne spezielle Konfiguration einfach möglich sein. Dabei sollten noch keine Systemerweiterungen zum Einsatz kommen.

3.8 Anwendungsfälle

In diesem Abschnitt werden Anwendungsfälle der Plattform beschrieben. Es werden nicht konkrete Szenarien oder konkrete Anwendungen beschrieben, sondern Anwendungsfälle der Plattform. Das heißt, die Zielsetzung der Plattform wird herausgearbeitet - also das Einsatzgebiet. Strukturiert sind die Anwendungsfälle anhand der zuvor definierten Akteure der Plattform (siehe 3.1). Im folgenden Abschnitt wird nur eine Auswahl der Anwendungsfälle abgebildet. Alle weiteren Anwendungsfälle befinden sich im Anhang (siehe 9).

Name	**Producer erstellen**
Beschreibung	Die zu analysierenden Daten müssen zuerst der Plattform beziehungsweise einer konkreten Anwendung zugänglich gemacht werden.
Beteiligte Akteure	Entwickler
Ergebnis	Es besteht ein Datenstrom zwischen einer Datenquelle und der Plattform in Form eines Datenstreams.
Nachbedingung	–
Standardablauf	1. Der Entwickler wählt eine geeignete Datenquelle. 2. Die Datenquelle wird über eine Komponente der Plattform angebunden. 3. Es existiert ein Datenstream der angeschlossenen Quelle.

Tabelle 3.1: Anwendungsfall: Producer erstellen

3.9 Existierende Ansätze

In diesem Abschnitt werden existierende Ansätze vorgestellt, die im Bereich der Network-Analytics eingesetzt werden können. Betrachtet werden unter anderem auch verschiedene Arbeiten, die sich ebenfalls mit dem Thema beschäftigen. Es wird aber keine Gegenüberstellung beziehungsweise Evaluation des eigenen Ansatzes zu existierenden Ansätzen vollzogen, sondern es findet vielmehr eine ergänzende Ausführung statt.
Zuerst werden verbreitete Produkte beschrieben und bezogen auf den eigenen Ansatz beleuchtet.

3.9.1 Amazon Kinesis

Die Anfänge des amerikanischen Unternehmens Amazon[5] sind im Markt des Online-Versandhandels von Büchern zu finden. Im Jahr 2006 kam dann eine Erweiterung des Produktangebots um IT-Infrastrukturservices hinzu: die Amazon Web Services (AWS) (Amazon Web Services Inc., 2017).
Die AWS umfassen mittlerweile auch ein Produkt zum Umgang mit Echtzeit-Streaming-Daten. Dabei ist diese Plattform über die AWS-Cloud verfügbar und kann kommerziell genutzt werden. Amazon Kinesis (hier: AK) ist in drei Bereiche zu unterteilen. Zuerst ist dabei **AK-Firehose** zu nennen. Über diesen Service wird es Nutzern ermöglicht, Daten der Plattform zugänglich zu machen - also in das System zu laden. Dieser Service ist die Schnittstelle zur AWS-Cloud

[5] https://www.amazon.com/

Name	**Producer installieren**
Beschreibung	Der Producer muss auf dem System der Datenanalyse installiert werden.
Beteiligte Akteure	Betreiber
Ergebnis	Der Producer kann für die Generierung von Streams eingesetzt werden.
Nachbedingung	–
Standardablauf	1. Der Betreiber hat den Producer installationsbereit vorliegen. 2. Der Betreiber installiert den Producer auf dem Zielsystem. 3. Es existiert ein Datenstream, der angeschlossenen werden kann.

Tabelle 3.2: Anwendungsfall: Producer installieren

und ist nicht nur mit Amazon Kinesis konnektierbar, sondern auch mit Produkten wie S3[6], Redshift[7] und Elasticsearch Service[8].
Nachdem Daten im Kosmos von AWS verfügbar gemacht wurden, bietet Amazon Kinesis eine erste rudimentäre Möglichkeit der Analyse von Daten. Dieser zweite Teilbereich nennt sich **AK-Analytics**. Bei der ersten einfachen Art der Datenanalyse kommt Standard-SQL zum Einsatz. Von Vorteil ist hierbei, dass kurze und einfache Abfragen schnell implementiert werden können. AK-Analytics ist somit ein High-Level Ansatz.
Der dritte Teilbereich von AK behandelt auch die Analyse der Daten und nennt sich **AK-Streams**. Hinzu kommt aber nicht die Verarbeitung von Datenstreams. Dagegen steht bei AK-Analytics nur Structured Query Language (SQL) zur Verfügung. Es werden für den Zweck der Analyse oder Verarbeitung benutzerdefinierte Anwendungen implementiert und in der Cloud zur Ausführung gebracht. Dabei können diese Anwendungen entweder auf Basis der „Kinesis Client Library“ oder auf der Teilkomponente „Spark Streaming“ von Apache Spark[9] entwickelt werden.
In Analogie zu Apache Kafka ist Spark Streaming sowie die Kinesis Client Library als eine andere Möglichkeit der Analyse und Verarbeitung von Daten-Streams zu sehen. Der AK-Firehose Service ist dann im Vergleich zu Apache Kafka dessen Producer-Komponente. Grundsätzlich ist Kinesis aber - wie anfänglich erwähnt - im Kosmos der AWS-Cloud konzipiert. Dabei sind

[6] https://aws.amazon.com/de/s3/
[7] https://aws.amazon.com/de/redshift/
[8] https://aws.amazon.com/de/elasticsearch-service/
[9] https://spark.apache.org/

Name	**Processor erstellen**
Beschreibung	Data Processor's sind die konkreten Anwendungen, anhand derer die konnektierten Datenstreams verarbeitet werden.
Beteiligte Akteure	Entwickler
Ergebnis	Auf Grundlage vordefinierter Topics können Daten anhand der entsprechenden Programmlogiken verarbeitet und analysiert werden.
Nachbedingung	Processor muss der Plattform zugänglich gemacht werden
Standardablauf	1. Ein entsprechendes Software Development Kit (SDK) installieren. 2. Der Processor wird anhand definierter Zielsetzung implementiert. 3. Der Processor wird der Plattform zugänglich gemacht.

Tabelle 3.3: Anwendungsfall: Processor erstellen

die genannten weiteren Services nur ein Teil dessen, was Amazon zusätzlich an Produkten anbietet.

3.9.2 Google Cloud Dataflow

Im Jahr 1997 waren Larry Page und Sergey Brin nicht die Ersten, die sich zum Ziel gesetzt hatten, eine Suchmaschine für das World Wide Web zu konzipieren. Unbestreitbar ist aber, dass fast 20 Jahre nach der Veröffentlichung von Google diese die populärste Internet-Suchmaschine ist. Ähnlich wie bei Amazon war auch bei Google am Anfang ein anderer Geschäftsschwerpunkt vorhanden. 2014 hat Google auf seiner Entwicklerkonferenz „Google I/O" deren neuen Cloud-Service „Cloud Dataflow"[10] vorgestellt.
Bei Google Cloud Dataflow (hier: GCD) handelt es sich um einen „Fully-managed data processing service [...]" (Google Inc., 2017). Diese Beschreibung an sich weist schon auf einen generischen Ansatz gegenüber einem reinen Stream-verarbeitenden Service hin. GCD zielt, entgegen der Zielsetzung dieser Arbeit, auch auf das Batch-Processing ab. Die Möglichkeit der Verarbeitung und Analyse auf diesen beiden Wegen wird von der Plattform auch dadurch erleichtert, dass unabhängig vom Processing Type (Stream oder Batch) eine einheitliche Entwicklungsgrundlage bereitgestellt wird. Das bedeutet, Konzepte wie das Windowing (siehe 2.4.2.1) sind bei beiden Arten einsetzbar. Der Service wird in der Cloud angeboten, sodass der Benutzer/Entwickler sich nicht um eines kümmern muss: die Ressourcenverwaltung (siehe 2.2.1). Durch die größtenteils automatisierte Verwaltung der Services wird auch eine Minimie-

[10] https://cloud.google.com/dataflow/

Name	**Processor installieren**
Beschreibung	Der Processor muss der Plattform zugänglich gemacht werden.
Beteiligte Akteure	Betreiber
Ergebnis	Der Processor kann für die Verarbeitung von Streams eingesetzt werden.
Nachbedingung	–
Standardablauf	1. Der Betreiber hat den Processor im Java Archive (JAR) Format vorliegen. 2. Der Betreiber installiert den Processor für die Plattform. 3. Es existiert ein Datenstream, der angeschlossenen werden kann.

Tabelle 3.4: Anwendungsfall: Processor installieren

Name	**Widget erstellen**
Beschreibung	Widgets sind die Basis für die Visualisierung von Ergebnissen der Daten Prozessoren.
Beteiligte Akteure	Entwickler
Ergebnis	Die Daten entsprechender Prozessoren können abgerufen werden.
Nachbedingung	Das Widget wird der Plattform zugänglich gemacht.
Standardablauf	1. Das Widget wird auf Grundlage der entsprechenden Frameworks implementiert. 2. Das Widget wird der Plattform zugänglich gemacht.

Tabelle 3.5: Anwendungsfall: Widget erstellen

rung der Latenzzeiten erreicht.
Vergleichbar mit der Konnektierung weiterer Amazon Services bei Kinesis bietet auch Google ähnliche Möglichkeiten. Genannt werden unter anderem Cloud-Services wie: „Cloud Storage, Cloud Pub/Sub, Cloud Datastore, Cloud Bigtable und BigQuery“ (Google Inc., 2017). Ein wichtiger Punkt bei GCD ist, dass das Programmiermodell Open-Source ist und somit jeder dieses erweitern kann und ein Pull-Request beim dem Projekt stellen kann.
In Bezug auf Apache Kafka lassen sich auch bei GCD Analogien herstellen. Hierzu zählt unter anderem das Programming Model, das mit den Data Prozessor bei Kafka in Verbindung zu bringen ist. Entgegen der Amazon Firehose liegt bei Dataflow aber der Fokus weniger auf dem Bereich der Datenquelle beziehungsweise der Datenakquise.

Name	**Widget installieren**
Beschreibung	Das Widget muss im Kontext der konkreten Anwendung zugänglich gemacht werden.
Beteiligte Akteure	Betreiber
Ergebnis	Das Widget arbeitet erwartungsgemäß.
Standardablauf	1. Das implementierte Widget liegt vor. 2. Das Widget wird für die Plattform installiert.

Tabelle 3.6: Anwendungsfall: Widget installieren

Konstatieren lässt sich allgemein das Folgende: Google ist von Gründungsbeginn an ein Unternehmen, das durch die Entwicklung der Suchmaschine und der damit verbundenen Verarbeitung, Analyse und Persistierung von Daten mit den grundlegenden Herausforderungen von Big Data (siehe 2.1) konfrontiert wurde. Das heißt, sie waren häufig die Ersten, die sich mit Problemen im Bereich der Verarbeitung und Analyse von Daten befassen mussten.

3.9.3 Apache Spark

Apache Spark[11] ist die Komponente, die als eine mögliche Processing-Engine zum Einsatz kommen kann. Dies ist, wie zuvor beschrieben, der Fall bei Amazon Kinesis (siehe 3.9.1). Unter dem Dach von Apache Spark werden mehrere Teilkomponenten zusammengefasst, die noch genauer unterschieden werden müssen. Spark subsumiert die Teilkomponenten Streaming, Application und Worker. Das Einsatzgebiet von Spark Streaming liegt in der Fähigkeit der Verarbeitung von Echtzeit-Datenströmen. Andere Möglichkeiten der Verarbeitung basieren auf Stapelverarbeitung, die einen großen und existenten Datenbestand voraussetzt. Das Ziel ist es, sich auf die Szenarien zu fokussieren, in denen in Echtzeit sichtbare Resultate erzielt werden sollen.
Die Spark-Master-Instanz ist der zentrale Knoten. Er dient der Verwaltung der Applications und Worker. Die genannten Begriffe werden im Weiteren noch genauer erklärt. Der als „zentral“ betitelte Master-Knoten widerspricht nicht dem Ansatz der Verteilung im Cluster. In einem Cluster übernimmt immer nur ein Knoten die Rolle des Masters (Status: *alive*). Alle weiteren Instanzen, die als Master definiert werden, reihen sich als sogenannte Slaves ein und schalten sich inaktiv (Status: *standby*). Sie werden erst zum Master, sofern der aktive Master-Knoten durch Absturz oder provoziertes Abschalten nicht mehr erreichbar ist.

[11] https://spark.apache.org/

Eine Spark-Slave-Instanz ist vom Master nur durch einen Punkt zu unterscheiden, den aktuellen Status. Wie zuvor beschrieben verhält sich diese Instanz bei Aktivierung genau wie ein Master-Knoten, wobei dann auch nicht mehr von Slave-Knoten, sondern Master-Knoten gesprochen wird. Die Orchestrierung der Master- und Slave-Instanzen übernimmt der Zookeeper (siehe 2.4.3). Dieser kann eine inaktive Slave-Instanz zum Master erwecken.
Die Application in der Terminologie von Apache Spark ist eine konkrete Implementation eines verarbeitenden Programms. Das Programm definiert sich aus dem zu lösenden Problem. Genauer heißt es, dass jede Application auf dem Cluster eine konkrete Verarbeitung von Daten beinhaltet und in entsprechender Form speichert. Das daraus resultierende Ergebnis entspricht dem gewonnenen Erkenntnisgewinn.
Die Spark-Worker sind jeweils eigenständige Knoten auf dem Spark-Cluster. Das Cluster bildet sich aus einer definierten Anzahl von Workern, die eine konkrete Arbeit übernehmen. Das dynamische Verhalten des Clusters erlaubt es auf diesem Weg, zur Laufzeit weitere Worker dem Cluster hinzuzufügen. Jeder Knoten muss sich, um als Worker definiert zu werden, initial einmal am Cluster als Worker anmelden. Auf diesem Weg entsteht der Pool an Workern. Sobald eine Application zur Ausführung gebracht wird, bedeutet das, dass eine Application in einem Worker-Knoten geladen wird.

3.9.4 Zusammenfassung

Die vorgestellten Ansätze in diesem Abschnitt (3.9) sind vor dem Hintergrund verschiedener Anforderungen beleuchtet worden. Dabei stand nicht originär Network-Analytics im Vordergrund, sondern das Stream Processing. Ein Grund hierfür ist, dass die Cloud Plattformen nicht nur speziell einen Bereich adressiert haben, sondern vielmehr generischer in ihrer Anwendung sind. Dabei lässt sich der Google Cloud Dataflow als Beispiel heranziehen. Hierbei hat Google auf einer noch niedrigeren Ebene der Abstraktion sogar eine Vereinigung der Handhabung von Daten Streams und Batches entwickelt. Das Schema des MapReduce, das in den Anfängen oft die Basis im Bereich von Big Data war, hat mittlerweile ausgedient.
Gezeigt hat sich aber auch, dass zumeist die Datensenke - oder genauer die Mechanismen für das Einleiten von Daten in die jeweilige Plattform - eine Herausforderung darstellt. Das heißt, jede Plattform benötigt auch ein eigenes Schema oder ein Datenformat, damit entsprechende Services überhaupt genutzt werden können. Im Fall von Amazon Kinesis lässt sich ein Beispiel dafür finden, dass die Plattformen aus vielen einzelnen Bestandteilen zusammengesetzt wurden. Apache Spark ist dabei eben eine Möglichkeit Anwendungen zu konzipieren und zu implementieren trotz der Verwendung einer proprietären Cloud.
Die vorgestellten Ansätze sind grundsätzlich nur eine Auswahl an Systemen beim Stream

Processing. Dabei wurden zwei Cloud-Services und eine Open-Source Variante vorgestellt. In der folgenden Tabelle sind einige weitere Systeme aufgelistet, die in der beschriebenen Domäne Verwendung finden können.

Name	**Beschreibung**	**Präsenz**
Amazon Kinesis	Vorgestelltes Amazon System zur Datenanalyse als Produkt der Amazon Webservices	aws.amazon.com
Google Cloud Dataflow	Vorgestelltes Cloud System von Google zum Stream- und Batch-Processing	cloud.google.com
Apache Spark	Vorgestelltes Open-Source Produkt innerhalb der Apache Foundation zur Datenanalyse im Kontext von Big Data	spark.apache.org
Microsoft Azure	Echtzeit Stream-Processing innerhalb der Microsoft Azure Cloud	azure.microsoft.com
Apache Storm	Eine Open-Source Plattform für verteilte Berechnung in Echtzeit	storm.apache.org
Apache Samza	Ein verteiltes Stream-Processing Framework mit der Verwendung von Apache Kafka[12] und Apache Hadoop YARN[13]	samza.apache.org
Apache Trident	Trident als Basis für Apache Storm[14]. Außerdem ist es für Echtheit-Berechnung konzipiert	storm.apache.org

Tabelle 3.7: Auflistung von Stream-Processing Systemen

Die Tabelle 3.7 ist keine umfassende Zusammenstellung von Systemen im Kontext des Stream-Processing, aber sie beschreibt neben den bereits vorgestellten Systemen weitere Produkte. Dabei wird eines aber deutlich. Unter dem Dach der Apache Software Foundation[15] sind viele verschiedene Systeme vorhanden, die sich in der Domäne des Stream-Processings zusammengefunden haben. Die Systeme fußen in Teilen auch wiederum auf weiteren Systemen von Apache.

[15] https://www.apache.org/

3.10 Zusammenfassung

Im Analysekapitel galt es zunächst, den Kontext der Plattform herauszubilden. Dafür wurde zuerst der Plattformkontext (Abbildung 3.1) genauer spezifiziert. Dabei wurden drei Akteure benannt, die in verschiedenen Konstellationen Beteiligte an der Plattform sind.
Anschließend wurden verschiedene Begriffe im Kapitel der Terminologie (3.2) genauer definiert. Hier formten sich schon erste Teilbereiche der Plattform heraus, auf die in den folgenden Abschnitten genauer eingegangen wurde. Gemeint ist somit die Datenquelle als eine initiale Komponente, die am Anfang einer jeden konkreten Anwendung steht. Daraufhin fand die Verarbeitung durch die Daten Prozessoren statt. Im Anschluss daran wurde die Visualisierung durch entwickelte Widgets erarbeitet.
Der Zweck und Einsatzbereich der Plattform ist ein wichtiger Abschnitt im weiteren Kapitel der Analyse. Hierbei wurden die Bereiche des Stream-Processings und der Network-Analytics eingehender behandelt. Es wurden die ersten Bezüge zu einigen verwendeten Frameworks, Plattformen und Systemen hergestellt. Im Wesentlichen ist dabei Apache Kafka hervorzuheben. Network-Analytics befasst sich eher mit dem Einleiten von Informationen in das System.
Die Anforderungsspezifikation gliedert sich an die Domänenanalyse sowie Modellierung. Herausgearbeitet wurden die Bereiche, die im Kontext der Plattform von Wichtigkeit waren. Auf Grundlage der Domänenanalyse wurden im Folgenden zwei Anwendungsszenarien beschrieben, die zum Ausdruck bringen, wie ein konkreter Einsatz auf Basis der Plattform stattfinden kann. Diese Szenarien sind auch als Vorbereitung auf die späteren Kapitel „Anwendung: Request Intersection“ und „Anwendung: Tor-Exit-Interceptor“ zu sehen.
Im weiteren Verlauf der Analyse wurden die Anforderungen, die an die Plattform zu stellen sind, genauer spezifiziert. Diese Anforderungen gliedern sich nach funktionaler, technischer und nicht-funktionaler Natur.
Des Weiteren wurden die Anwendungsfälle tabellarisch zusammengefasst und beschrieben. Diese sind anhand der vorkommenden Akteure strukturiert und sind auch im Anhang der Ausarbeitung ergänzend zu finden.
Die Analyse schließt mit der Auseinandersetzung verschiedener existierender Arbeiten ab. Hierbei finden auch die Teilbereiche der Plattform verstärkt Erwähnung, die in fast allen Arbeiten eine Gemeinsamkeit darstellen.

4 Konzeption

Dieses Kapitel beginnt mit der Architekturbeschreibung der Plattform in ihrer Gesamtheit. Es wird zuerst ein allgemeiner Überblick über die Plattform geschaffen. Daran anschließend wird dann detallierter auf die Plattform eingegangen. Die zentralen Bereiche werden behandelt und in der Art erläutert, dass ein feingranulares Bild über die Plattform entsteht. Das Konzeptionskapitel hat insgesamt das Ziel die Ausrichtung und den Nutzen der Plattform zu konkretisieren.
Anknüpfend an die Beschreibung der Plattform werden dann die Komponenten und Systeme im Abschnitt Softwarestack beschrieben, die von zentraler Bedeutung für das Vorhaben der Plattform sind. Es werden die jeweiligen Ziele der Komponenten beziehungsweise deren Zweck in den Vordergrund gestellt.

4.1 Architektur

Die Architektur der Plattform unterliegt den Anforderungen, die in der Analyse dargestellt sind. Das heißt, bei der Konzeption ist im Allgemeinen darauf zu achten, dass zum Beispiel die Echtzeitfähigkeit in der Analyse einzelner konkreter Anwendungen nicht durch eine falsche Wahl oder Implementation von Komponenten in Mitleidenschaft gezogen wird. In diesem Abschnitt sind aber nicht einzelne konkrete Komponenten im Fokus, sondern das Gesamtkonstrukt. Die Architektur in Abbildung 4.1 untergliedert sich in drei zentrale Bereiche: die zugrundeliegende Infrastruktur, die eigentliche Plattform sowie die konkreten Anwendungen auf Basis der Plattform. In den folgenden drei Abschnitten werden diese Teilbereiche von der Infrastruktur bis hin zu den Anwendungen erläutert.

4.1.1 Infrastruktur

Die zugrundeliegende Infrastruktur für den Betrieb der Plattform bedeutet allgemein sowohl eine entsprechende Netz- als auch eine Hardwareinfrastruktur. Im Fall der Netztinfrastruktur ist es für den Betrieb der Plattform wichtig, dass es keine Limitationen für die Plattform in

der Art gibt, dass zu geringe Upstreams oder Downstreams den Betrieb im Durchsatz der Daten beeinflussen könnten. Auf der Seite der Hardware ist weniger die konkrete Hardware im Einzelnen von Bedeutung. Das heißt auch, ob zum Beispiel durch Virtualisierung Ressourcen verfügbar sind oder ob wirkliche Root-Server eingesetzt werden. Vielmehr ist die Verfügbarkeit der Docker-Engine entscheidend, da die Plattform auf dieser konzipiert ist.

4.1.2 Plattform

Die Plattform ist der eigentliche Bestandteil dieser Arbeit und wurde sowohl in den Grundlagen als auch in der Analyse eingehender behandelt. Dabei wird in der Konzeption ein genaues Bild über die konkrete Ausprägung der Plattform und ihrer Schnittstellen in Abschnitt 4.2 geschaffen. Für die Architektur ist primär von Bedeutung, dass die zuvor beschriebene Grundlage weniger auf spezieller Hardware als auf der Containerisierung fußt. Dadurch lassen sich auch die Konzepte der Service-Oriented Architecture (2.5.3) beziehungsweise des Cloud Computing (2.2.1) realisieren. Für den Betrieb entwickelter Anwendungen auf Grundlage der Schnittstellen lassen sich die konkreten Anwendungen ebenso wie die Komponenten der Plattform durch die Containerisierung orchestrieren.

4.1.3 Anwendung

Konkrete Anwendungen, die auf Basis der Plattform entwickelt wurden, bilden die oberste Ebene der Architektur in Abbildung 4.1. Dabei sind zwei konkrete Anwendungen in der Abbildung symbolisiert und eine weitere, die optional durch eine gestrichelte Linie angedeutet ist. Als konkrete Beispiele für Anwendungen sind in den Kapiteln „Anwendung: Request Intersection“ (5) sowie „Anwendung: Tor-Exit-Interceptor“ (6) sowohl die Entwicklung als auch Analyseergebnisse einer vorhergehenden Konzeption beschrieben.
Die beiden Anwendungen stehen dabei exemplarisch für Möglichkeiten einer Entwicklung, stellen aber keine allgemeine Blaupause für Anwendungen dar. Das heißt, das Konzept der Plattform ist generisch im Sinne der Netzwerkdatenanalyse.

4.2 Plattform

Dieser Abschnitt beschreibt die Konzeption der Plattform. Die verschiedenen Ansätze der Architektur werden in Hinblick auf die Ausrichtung dieser Arbeit beschrieben.
Am Anfang steht die Systemarchitektur - oder auch Plattformarchitektur - im Mittelpunkt, die zuvor in den Grundlagen sowie später eingehender in der Analyse behandelt wurde.

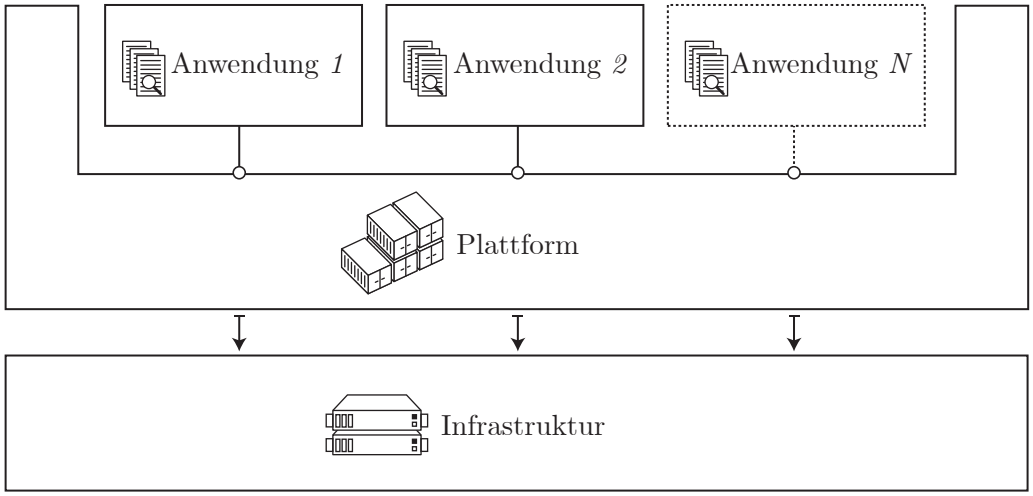

Abbildung 4.1: Architektur im Gesamtkontext

Ein rudimentärer Überblick wird geschaffen, der die Konzeption der verschiedenen Systemkomponenten und deren Zusammenwirken beschreibt. Weiterhin findet eine eingehendere Auseinandersetzung mit den jeweiligen Komponenten der Plattform statt. Es werden deren Aufgaben im Gesamtkonstrukt sowie die Schnittstellen beschrieben und konzipiert.
Die Plattform ist in der Gesamtheit kein abgeschlossenes Konstrukt. Das heißt, konkrete Analyseszenarien werden erst durch die Entwicklung von weiteren „Extensions“ ermöglicht. Dieses fußen auf der Plattform und definieren erst durch ihre Ausrichtung den Kontext des Analyseszenarios. Aus diesem Grund bedarf es einer Konzeption der Extensions im Allgemeinen. Hierzu zählen die Widgets für den Bereich der Visualisierung sowie die Datenprozessoren im Bereich des Back-Ends zur Verarbeitung konkreter Datenströme. Weiterhin werden die Möglichkeiten der Generierung von Datenstreams im Abschnitt der Extensions beschrieben.

4.2.1 Übersicht

Die Abbildung 4.2 zeigt eine rudimentär-schematische Darstellung der Plattform und ihrer einzelnen Bestandteile. Ziel der Abbildung ist es einen Gesamtüberblick über die Plattform zu erlangen. Die anfänglich in den Kapiteln Grundlagen und Analyse beschriebenen Systemkomponenten werden hierbei nicht umfassend in Hinblick auf dem Funktionsumfang dargestellt, sondern vielmehr im Zusammenwirken des Gesamtkontextes der Plattform gezeigt. Es existieren drei zentrale Bereiche der Plattform, die zu unterscheiden sind:

Akquise Datengenerierung und Anbindung an die Plattform

Verarbeitung Datenverarbeitung auf Basis der Plattform

Visualisierung Ergebnisvisualisierung von Daten der Verarbeitung

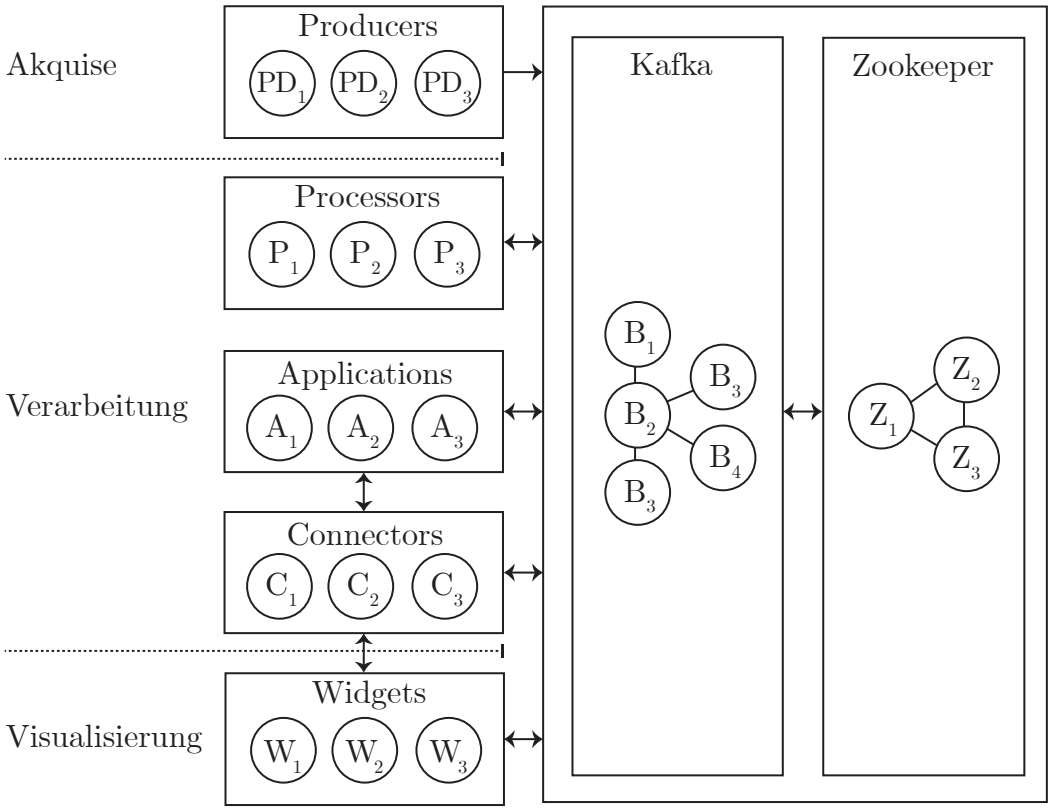

Abbildung 4.2: Systemarchitektur nach Komponenten

Wichtig in der Abbildung 4.2 ist der Grundzug der Strukturierung der Systemteile. In der Abbildung sind sowohl Kafka als auch Zookeeper als zwei getrennte Cluster visualisiert. Grundsätzlich sieht das Konzept der Plattform aber nicht die Verwendung mehrerer voneinander unabhängiger Cluster vor. Eine konkrete Konfiguration der jeweiligen Knoten eines Clusters hängt aber von der infrastrukturellen Verfügbarkeit von Server-Instanzen ab. Aus diesem Grund ist in der Darstellung eine Separation der Knoten visualisiert. In einer konkreten Belegung können aber durchaus Überlagerungen beziehungsweise Schnittmengen existieren.
Die Kommunikationspfade der Darstellung sind, wie einleitend erwähnt, rudimentär-schematisch.

Zum Beispiel ist hierbei die Bridge-API nicht näher spezifiziert, da in dieser Darstellung das Gesamtkonstrukt der Plattform ersichtlich werden soll.
Die *Applications* im Bereich der Verarbeitung sind im Allgemeinen Anwendungen, die mit der Plattform über die entsprechenden Schnittstellen kommunizieren können, die aber für das Gesamtkonstrukt keine direkte Bedeutung haben. Als Beispiel sind hier Anwendungen gemeint, die einen Service bereitstellen, der proprietär angeboten wird, der aber nicht durch eine Implementation eines eigenen Prozessors gewährleistet werden kann.
Die Producer sind die Systemkomponenten, die anhand einer von mehreren konzipierten API's Daten in das System beziehungsweise in die Plattform leiten. Sie sind über die Producer-API mit dem Kafka-Cluster konnektiert. Da es keine Limitation bezüglich der Anzahl von Producern gibt, variieren diese je nach Analyseszenario.
Die Data Prozessoren nutzen die Streams-API und dienen der Verarbeitung und Transformation entsprechender Datenstreams, die zuvor durch Producer generiert wurden. Ihr Kommunikationskanal ist bidirektional im Konzept der Plattform, da sie lesend und schreibend agieren können.
Connectoren bieten zum einen die Möglichkeit der Kommunikation mit etwaigen Applications, zum anderen eine Anbindung an das Kafka Cluster.
Da Widgets direkt Daten über eine entsprechende API beziehen, sie aber keine Aufgabe der Verarbeitung übernehmen, nutzen sie die Consumer-API über die Indirektion der *Bridge*. Die Bridge ist in Abbildung 4.2 nicht dargestellt. Abschließend arbeiten die Broker- sowie Zookeeper-Instanzen im Verbund auf dem Cluster. Da je nach Konfiguration der einzelnen Teilkomponenten eine Verbindung auch über die Zookeeper hergestellt wird, sind diese in der schematischen Darstellung strukturell zusammengefasst.

4.2.2 Instanzstruktur

Die Strukturierung der Systembereiche nach Komponenten diente der Veranschaulichung der Systemarchitektur. In einem konkreten Szenario wird die Strukturierung anhand der infrastrukturell verfügbaren Ressourcen durchgeführt. Eine beispielhafte Belegung ist in Abbildung 4.3 dargestellt. Die Abkürzungen der Buchstaben sind analog zur Abbildung 4.2 gewählt. Wichtig hierbei ist die Komponente *WS*. Es handelt sich um einen reinen Webserver, dessen Aufgabe es ist, den statischen Content des Front-Ends auszuliefern. Aus der Abbildung wird somit ersichtlich, dass er in demselben Netz beziehungsweise auf einem Knoten aus dem Netz ausgeführt wird. Die entsprechenden Widgets sind logisch somit dem Front-End zuzuordnen, in dem sie zur Ausführung gebracht werden.

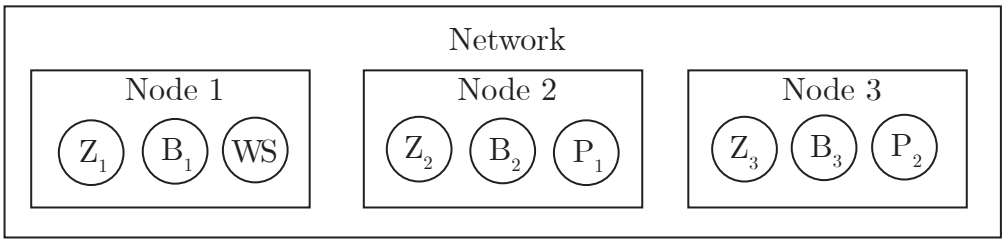

Abbildung 4.3: Instanzen nach Clusterknoten

4.2.3 Komponentenarchitektur

Der Abschnitt der Komponentenarchitektur gibt einen detaillierten Überblick über die einzelnen Komponenten der Plattform. Dabei werden die Aufgaben sowie Schnittstellen genauer beleuchtet, die in der Konzeption der Plattformarchitektur anfänglich behandelt wurden. Der Abschnitt Softwarestack gab einen ersten Überblick über die technischen Details im Kontext der jeweiligen Komponenten. Die Übersicht zeigt das Zusammenwirken der einzelnen Komponenten im Gesamtkontext der Plattform. Abschließend werden diese Komponenten in diesem Kapitel behandelt.

4.2.3.1 Content-Server

Beim Webserver handelt es sich, wie zuvor im Abschnitt Instanzstruktur beschrieben, um die Komponente zum Ausliefern von statischen Inhalten. Es sind im Allgemeinen die Formate Hypertext Markup Language (HTML), Java Script (JS), Cascading Style Sheets (CSS) sowie Mediaobjekte betroffen. Wie in den Kapiteln Analyse und Konzeption beschrieben werden die Widgets im Bereich des Front-Ends der Plattform implementiert und betrieben.

Aus diesem Grund wird die API zur Bridge (siehe 4.3.2.3, 4.2.3.2) nicht über den Server bereitgestellt. Aus Gründen der SoC sollten die API und der Content-Server getrennt voneinander sein. Im Zuge der Einfachheit und einer geringeren Komplexität der Plattform wird der Bridge-Node und Content-Server jeweils in verschiedenen Images gekapselt.

Web-Client Der konkrete Web-Client basiert auf AngularJS (siehe 2.4.4). Der Client ist per HTTP erreichbar. Nach dem Ausliefern der Daten initialisiert sich dann die Single Site Pages (SSP) von AngularJS. Im Zuge des Prozesses der Initialisierung werden auch vorhandene Widgets mit geladen, sodass diese im Client verfügbar sind. Grundsätzlich obliegt es dem Entwickler, in welcher Form eine Widget Daten visualisiert, sodass das Widget die anzusprechende

Schnittstelle selbst wählt.
Die Konzeption sieht vor, dass ein Widget sich über die Bridge-API des Bridge-Servers an die Plattform konnektiert um Datenstreams zu abonnieren. Die Bridge-API bietet zwar die Möglichkeit einer bidirektionalen Kommunikation zwischen der Bridge-Komponente und dem Kafka Cluster, doch im Regelfall wird die API unidirektional zum Einsatz kommen.

4.2.3.2 Bridge-Node

Der Bridge-Node ist das Bindeglied zwischen dem Ökosystem von Kafka und seinen Streams sowie dem Front-End. Dabei basiert das GUI auf einer SSP des AngularJS Frameworks. Dieser Web-Client kommuniziert dann auf Basis der Bridge-API mit dem Back-End.
Der Bridge-Node erfüllt die Kriterien einer *Application*, wie sie in Abbildung 4.2 dargestellt und in Abschnitt Übersicht (4.2.1) beschrieben sind. Dabei definiert sich die API des Bridge-Nodes wie folgt:

Methode	**URL-Schema**	**Beschreibung**
Subscribe	/v2/broker/?topics=*topicList*	Eröffnen eines Web Socket (WS) der Datenstreams von *topicList*. Dabei ist *topicList* eine kommaseparierte Liste der zu abonnierenden Topics.
Publish	/v2/broker	Erstellen und Publizieren neuer Events auf Basis des JSON-Formats - die Struktur ist im folgenden Codebeispiel (4.1) ersichtlich.

Tabelle 4.1: WS-Bridge-API

```
1 {
2     "topic": "topic-name",
3     "key": "partition-key",
4     "message": "message text"
5 }
```

Listing 4.1: WS-Message-Format

Der Bridge-Node sowie alle anderen direkten Komponenten der eigenen Plattform werden wie zuvor beschrieben in Containern verpackt. Der Bridge-Node basiert auf dem Projekt *kafka-websocket*[1], das unter der Apache License 2.0[2] bei GitHub veröffentlicht ist.

[1] https://github.com/b/kafka-websocket
[2] https://www.apache.org/licenses/LICENSE-2.0.html

4.2.3.3 Extensions

Der Abschnitt Extensions beschreibt die Schnittstellen der jeweiligen Komponenten, anhand derer eine Verbindung mit der eigenen Plattform hergestellt werden kann. Die drei Komponenten werden genauer beschrieben, die den Kontext einer jeden Anwendung beziehungsweise Analyse auf Basis der Plattform definieren.

Data Producer Der Data Producer wurde bereits anfänglich in Abschnitt 4.3.2.1 behandelt. Es wurde die API von Kafka beschrieben, die den Producern zugrundeliegt. Die zwei angebotenen Schnittstellen der Producer-Komponente sind in Abbildung 4.2 ersichtlich.

Typ	Beschreibung
WebSocket	Bei der Verwendung dieser Schnittstelle muss die Komponente mit zwei Parametern initialisiert werden.
Std-In	Die zweite Möglichkeit - Datenstreams der Plattform zugänglich zu machen - ist die über das Command Line Interface (CLI). Dabei kann die Producer-Komponente über Pipes beziehungsweise *StdIn* angesprochen werden.

Tabelle 4.2: Producer-API

Die Komponente verlangt zwei zentrale Konfigurationsparameter für die Initialisierung und den Betrieb. Diese sind: *topic* und *bootstrap-server*.

Parameter

- **topic**: Das Topic, über das die Daten in der Plattform publiziert werden. Somit ist dieser Datenstrom für die Daten Prozessoren darüber zugänglich.
- **bootstrap-server**: Eine Liste der aktiven Kafka-Broker

4.2.3.4 Data Processor

Die Daten Prozessoren wurden im Abschnitt Softwarestack beschrieben. Dabei fand das Konzept eines Prozessors in „Implementation von Daten Prozessoren“ (4.3.2.2) Erwähnung. In diesem Abschnitt der Extensions wird nun ein beispielhafter Prozessor anhand der Kafka-Streams-API skizziert. In der entsprechenden IDE müssen zuerst einmal die benötigten Klassen verfügbar gemacht werden. Dafür kann zum Beispiel Maven[3] genutzt werden. Die Kafka-Stream–Dependency kann wie folgt hinzugefügt werden:

[3] https://maven.apache.org/

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>0.10.x.x</version>
</dependency>
```

Listing 4.2: Kafka-Stream Maven Dependency

Der Versions-Knoten des XML-Eintrags ist auf die systemweit gleiche Version zu setzen. Das heißt im Fall der aktuellen Version: *0.10.1.0*. Im folgenden Code ist eine beispielhafte Implementation eines Daten Prozessors abgebildet. Definiert werden zuerst die entsprechenden Konfigurationsparameter. Über die Klasse *KStreamBuilder* wird der eigentliche Prozessor instanziiert. Entscheidend für die Implementation sind die Zeilen 12 und fortfolgende.
Im Beispiel 4.3 wird der eingehende Wert (Type Integer) mit *2* multipliziert. Der Eingangswert wird dabei aus dem Topic *in-topic* gelesen und im Anschluss an die Transformation des Wertes in das Topic *out-topic* geschrieben.

```
Map<String, Object> props = new HashMap<>();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "1");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
props.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, Serdes.Integer().
    getClass().getName());
props.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.Integer().
    getClass().getName());

StreamsConfig config = new StreamsConfig(props);

KStreamBuilder builder = new KStreamBuilder();

builder
  .stream("in-topic")
  .mapValues(value -> {
    return value * 2
  })
  .to("out-topic");

KafkaStreams streams = new KafkaStreams(builder, config);

streams.start();
```

Listing 4.3: Data Processor Beispiel

4.2.3.5 Widgets

Die zuvor beschriebenen Widgets in Abschnitt 4.3.2.3 sowie die Anwendungsfälle in Abschnitt 3.8 haben anfänglich auf die Notwendigkeit der Konzeption einer adäquaten Datenvisualisierung hingewiesen. Die Widgets sind keine eigenständigen Komponenten zur Datenvisualisierung, sondern sie sind erst im Verbund mit dem Content-Server (4.2.3.1) sowie mit der Bridge-API (4.2.3.2) einsetzbar. Die Klammer dieser Komponenten ist somit das Dashboard, das für den Akteur des Konsumenten gedacht ist.
In der Konzeption und Entwicklung von Widgets ist die Bridge-API das zentrale Bindeglied zwischen Front-End - in dem das Widget betrieben wird - und den Prozessoren auf Basis der Plattform. Der konzipierte Web-Client fand bereits Erwähnung in Abschnitt 4.2.3.1. Im Folgenden wird nun das *KafkaConsumer* Modul des Web-Client konzipiert. Das Protokoll, über das der Datenaustausch stattfindet, ist WS beziehungsweise Web Socket Secure (WSS). Das *KafkaConsumer* Modul setzt voraus, dass die JavaScript WebSocket API verfügbar ist. Dies ist so gut wie in allen aktuellen Webbrowsern gegeben. Das Modul baut auf die eventbasierte Ereignisverarbeitung, sodass zumeist die gängigen vier folgenden Events implementiert werden:

- onOpen
- onError
- onMessage
- onClose

Das *onOpen* Event wird bei einem erfolgreichen Verbindungsaufbau zum Bridge-Server ausgelöst. Das Pendant des *onError* Events wird hingegen beim Fehlschlagen eines Verbindungsaufbaus ausgelöst. Das Event *onClose* beschreibt einen „geordneten“ Verbindungsabbau ohne einen entsprechend aufgetretenen Fehler.
Das entscheidende Event *onMessage* betrifft somit das Auftreten neuer Nachrichten über einen Datenkanal (Stream). Konkret heißt das, die Widget-Logik ist um das Auftreten neuer Nachrichten aus den jeweiligen Topics zu konzipieren. Dabei ist in dem Codebeispiel 4.4 eine beispielhafte Implementation abgebildet. Das Beispiel gibt die empfangenen Nachrichten auf der Konsole aus. Ein entsprechendes Data-Binding im AngularJS Framework ist in diesem Beispiel aus Gründen der Einfachheit nicht enthalten.

```
var inst = new KafkaConsumer(["out-topic"]);

inst.onOpen(() => {
    console.info("Connected");
});

inst.onError((err: ErrorEvent) => {
    console.error(err);
});

inst.onClose(() => {
    console.info("closed")
});

inst.onMessage((msg: number) => {
    console.log("Message", msg)
});
```

Listing 4.4: Widget-Beispiel

4.3 Softwarestack

Der Abschnitt des Softwarestacks beschreibt die Plattformen, Frameworks und Bibliotheken, die für die entwickelte Plattform von Bedeutung sind. Hierbei werden die verschiedenen genannten Bestandteile im Softwarestack zusammengefasst. Das Ziel ist es, einen konzeptionellen Überblick zu geben, sodass ein inhaltliches Verständnis davon entsteht, welche technischen Verantwortlichkeiten durch welche Systeme gewährleistet werden. Die verschiedenen Systeme sind in ihrer Zuständigkeit den drei zentralen Bereichen der Plattform zugeordnet: Producing, Processing und Visualization.
Berücksichtigt werden die Bereiche der jeweiligen Systeme, die eine technische und strukturelle Relevanz für die Plattform darstellen. Das hat aber auch im Einzelfall zur Folge, dass in der Konzeption keine allumfassende Beschreibung etwaiger Systeme gewährleistet wird.
Das Kapitel Softwarestack ordnet sich von der Basis der Containerisierung (Docker) bis hin zu den High-Level Frameworks im Bereich der Front-End Widgets (Angular).

4.3.1 Docker

Das Konzept der Software-Containerisierung wurde zuvor in den Grundlagen behandelt (siehe 2.4.1). Die Kapselung einer Software innerhalb eines Containers ist dabei das primäre Charakteristikum. Dabei wird durch die isolierte Ausführung der Software innerhalb des Containers deutlich, dass die Architektur eines verteilten Systems anhand von Microservices ein geeignetes Muster ist.

Im folgenden Abschnitt wird die Docker-Plattform im Einsatz der eigenen Plattform zur Netzwerk-Datenanalyse beschrieben. Das Augenmerk liegt auf dem Betrieb einer im Container gekapselten Software. Das Container-Management ist eine der Aufgaben, die für den erfolgreichen Betrieb der Plattform wichtig sind. In dem Zusammenhang ist Kubernetes eines der Werkzeuge, die für den Produktionsbetrieb des Systems einsetzbar sind. Hingegen sind die Anforderungen bei der Entwicklung von Daten Prozessoren andere als in der Produktion. Aus diesem Grund muss die Plattform sowohl eine adäquate Möglichkeit des Einsatzes für die Entwicklung sowie für die Produktion bieten.

4.3.1.1 Entwicklung und Produktion

Ein häufig beschriebener Vorteil von Docker liegt im Betriebsverhalten einer entwickelten Software. Durch die Containerisierung findet sowohl eine Kapselung sowie eine Isolation der betriebenen Software statt. Dadurch erfolgt die Kommunikation nach außen einzig anhand der angebotenen Software-API. Daraus ergibt sich, dass Container einfacher und besser orchestriert werden können - sie also keine enge Kopplung zu der Laufzeitumgebung aufweisen.
Somit ergibt sich der Vorteil, dass verschiedene Umgebungen (z.B. Entwicklung und Produktion) für den Betrieb des Containers keine Probleme darstellen (Merkel, 2014; Liu und Zhao, 2014). Diese Gewissheit führt dazu, dass die eigene Plattform in verschiedenen Betriebsumgebungen eine einfache Möglichkeit der Integration gewährleisten muss. Im Folgenden werden nun die zwei wichtigsten Betriebsumgebungen genauer beschrieben.

Entwicklung Für die Entwicklung bietet Docker eine native Möglichkeit verschiedene Komponenten im Verbund zu betreiben. Das entsprechende Tool heißt Docker Compose[4]. Da die Konzeption der eigenen Plattform vorsieht, dass diese als verteiltes System entwickelt und betrieben wird, ist Docker-Compose für die Entwicklung von Daten Prozessoren prädestiniert. Die Basis des Einsatzes der Plattform mit Hilfe von Docker-Compose bildet eine auf Yet Another Markup Language (YAML) basierende Konfigurationsdatei - die *docker-compose.yml*.
Innerhalb dieser Datei werden die Rahmenbedingungen beschrieben, die für die verbundenen Container gelten. Die Container an sich basieren auf der Dockerfile (siehe 2.4.1.3). Für den gängigen Fall einer lokalen Entwicklung von Daten Prozessoren sind alle entsprechenden Teilkomponenten der Plattform betreibbar. Dabei sind zwei unterschiedliche Möglichkeiten vorstellbar. Zum einen kann die Plattform ohne entsprechende Prozessoren gestartet werden. Die konkrete Entwicklung geschieht mit Hilfe einer Integrated Development Environment (IDE). Beim Starten des Prozessors aus der IDE müssen die entsprechenden Schnittstellen der

[4] https://docs.docker.com/compose/

Plattform konfiguriert und konnektiert werden. Zum anderen kann aber auch eine JAR gebildet werden um diese anschließend der laufenden Plattform hinzuzufügen.

Produktion Anders als in der Entwicklung unterscheidet sich die Produktionsumgebung primär durch die physische Verteilung der jeweiligen Container der entwickelten Plattform. Beim Prozess der Implementation etwaiger neuer Prozessoren liegt das Augenmerk auf der Einfachheit des Vorgangs gegenüber der Performance im Produktionsbetrieb. Ein Betrieb für die Produktion ist grundsätzlich auch auf Basis von Docker-Compose möglich. Dies macht in den Fällen Sinn, wenn ein Service beispielsweise keine Verteilung der Komponenten anbietet oder nur auf wenigen Softwarekomponenten in Containern basiert. Im konkreten Fall ist dies aber nicht dem Ziel einer verteilten Plattform zuträglich. Ein Grund hierfür ist, dass beispielsweise Kafka-Streams so konzipiert ist, dass eine Verteilung einzelner Stream Prozessoren in der Architektur großer Anwendungen vorgesehen ist.
Die Herausforderung der Verwaltung und Orchestrierung verschiedener Container, die im Verbund betrieben werden sollen, begegnet Kubernetes[5]. Kubernetes zeichnet sich durch die folgenden Bereiche aus. Zum einen ist das die Möglichkeit des automatisierten Deployments von Containern. Dabei können neue Container einem existierenden Verbund (Plattform) unkompliziert hinzugefügt werden. Da das Ausrollen neuer Container häufig auch fehleranfällig ist, werden bei Kubernetes in diesen Fällen automatisch Rollbacks ausgeführt. Weiterhin wird Kubernetes damit beworben, dass das System durch das Hinzufügen weiterer Container trotzdem skaliert. Als Beispiel wird hierbei der Einsatz von Kubernetes bei Google genannt: „Designed on the same principles that allows Google to run billions of containers a week [...]" (The Linux Foundation, 2017).

Weitere Einsatzumgebungen Die beiden zentralen Einsatzumgebungen der Entwicklung und Produktion beschreiben vor dem Hintergrund der jeweiligen Anforderungen den konzeptionellen Einsatz der Plattform. Diese beiden Umgebungen bilden das Fundament. Je nach Anwendungskonzeption beziehungsweise Continuous-Integration-Flow können durchaus weitere Umgebungen hinzukommen. Dazu gehören etwa *Testing* und *Staging*. Somit lassen sich vier gängige Umgebungen spezifizieren, die bei einem gesamtheitlichen Workflow beachtet werden müssen. Bei dieser Kategorisierung lassen sich wiederum das *Development* und *Testing* sowie *Staging* und *Production* gruppieren. Durch die strukturelle Gemeinsamkeit der jeweiligen Umgebungen bietet sich die Verwendung derselben Werkzeuge an. Grundsätzlich möglich ist aber auch die Erwägung und der Einsatz von Docker Swarm[6]. Hierbei handelt es sich um

[5] https://kubernetes.io/
[6] https://docs.docker.com/swarm/

ein Werkzeug zum „[...] native clustering for Docker“ (Docker Inc., 2017). Das wird dadurch ermöglicht, dass durch Virtualisierung verschiedene Docker-Hosts als ein Verbund nach außen zugänglich gemacht werden: „It turns a pool of Docker hosts into a single, virtual Docker host“ (Docker Inc., 2017). Von Vorteil bei Docker Swarm ist, dass die Swarm-API mit der Remote-API kompatibel ist. Dadurch lassen sich die beiden Docker-API miteinander verbinden.

Konfigurations-Schema Eine beispielhafte Konfiguration ist in Abbildung 4.4 dargestellt. Die zuvor beschriebenen Unterschiede zwischen Entwicklung und Produktion werden schematisch skizziert. Außerdem wird das Zusammenwirken von Containern und den jeweiligen Umgebungen dargestellt.

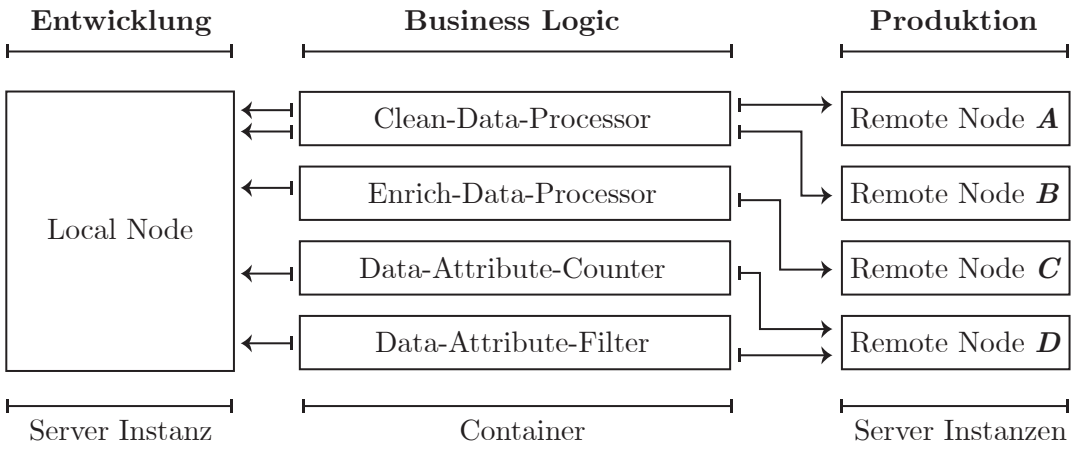

Abbildung 4.4: Beispielhafte Konfiguration der Entwicklung gegenüber der Produktion

Fazit Zusammenfassend lassen sich im Kontext von Docker, genauer der Containerisierung der Komponenten der Plattform, verschiedene Sachverhalte konstatieren.
Zunächst bedingen die infrastrukturellen Gegebenheiten direkt die etwaigen Umgebungen. Das heißt, bei nur einem vorhandenen Host-Knoten in der Produktion erscheint der Einsatz von Kubernetes überdimensioniert. Aus diesem Grund wäre in dem beschriebenen Fall auch an Docker Compose für den Produktionseinsatz zu denken.
Die eingangs beschriebene Isolation und Kapselung der Plattformkomponenten ist als ausschlaggebend für eine einfache Verteilung der Container (Komponenten) zu sehen. Aus der Verteilung ergibt sich dann unter anderem auch eine bessere Möglichkeit etwaigen System-Performance-Problemen begegnen zu können. Da dies für die Entwicklung noch zu vernach-

lässigen ist, wird es sich aber in der Produktion auszahlen. Daraus ergibt sich dann auch das Attribut der Echtzeitfähigkeit der Plattform. Im Fall von Performance-Problemen bezüglich des Durchsatzes von Daten sind die Daten Prozessoren skalierbar.
Die grundsätzliche Entscheidung, auf die Containerisierung von Docker zu bauen, ist der Konzeption eines Verteilten Systems gegenüber einer monolithischen Architektur geschuldet. Da jedes System zumeist einem stetigen Wandel unterworfen ist, trägt diese Entscheidung auch dazu bei, auf zukünftige Veränderungen entsprechend reagieren zu können.

4.3.2 Kafka

Apache Kafka ist die Komponente der eigenen Plattform, die metaphorisch als Herz bezeichnet werden kann. In den Grundlagen zu Apache Kafka wurden bereits erste Konzepte von Kafka-Streams vorgestellt. Dieser Abschnitt hat zum Ziel, einen tieferen Einblick in Kafka zu gewähren und die Verbindung zur eigenen Plattform herzustellen. Behandelt werden im Folgenden die vier verschiedenen Kafka-API:

- 4.3.2.1 : Producer-API
- 4.3.2.2 : Streams-API
- 4.3.2.3 : Consumer-API
- 4.3.2.4 : Connect-API

Zunächst ist eine Beschreibung des Zusammenwirkens der einzelnen Systemteile von Apache Kafka wichtig. Dabei lässt sich Kafka in drei Bereiche gliedern: Publish/Subscribe, Processing und Storing von Daten (siehe 2.4.2). Das Publish/Subscribe Konzept baut dabei auf den Einsatz von sogenannten Topics. Die Topics sind im Allgemeinen Data Pipelines. Das bedeutet, auf verschiedene Weise konnektierte Daten Produzenten können in entsprechende Pipelines schreiben. Diese Daten können dann von Prozessoren abgerufen, verarbeitet und analysiert werden.

Topics Die Kafka Topics dienen der Adressierung von Datenströmen innerhalb der Kafka Plattform. Das Konzept dabei ist, entsprechende Daten aus Topics zu lesen oder zu schreiben. Auf diese Weise ist es möglich, Verarbeitungsketten zu konstruieren. Ein Beispiel:
Am Anfang steht eine existierende Datenquelle. Diese wird mit der Plattform über ein Topic konnektiert. Dabei ist ein entsprechender Bezeichner zu wählen. Im Folgenden werden die Daten des Streams durch verschiedene Daten Prozessoren verarbeitet, wobei diese jeweils immer ihre Daten aus Topics lesen und schreiben.

Dadurch kann man den Ablauf mittels einer entsprechenden Pipeline-Struktur modellieren. Die Abbildung 4.5 stellt eine solche Verarbeitungskette dar. In dem Beispiel werden Netzwerkdaten durch Topic-Pipelines separiert, verarbeitet und analysiert. Das Ergebnis ist eine Übersicht über die physisch größten Knoten des Netzwerkverkehrs. Dabei entspricht der Workflow *A => B => C* der Summierung der Paketgrößen und *A => D => E* der Bestimmung der Geolokation der entsprechenden Internet Protocol (IP)-Adressen. Das Ergebnis ist dann über das Topic *E* abrufbar.

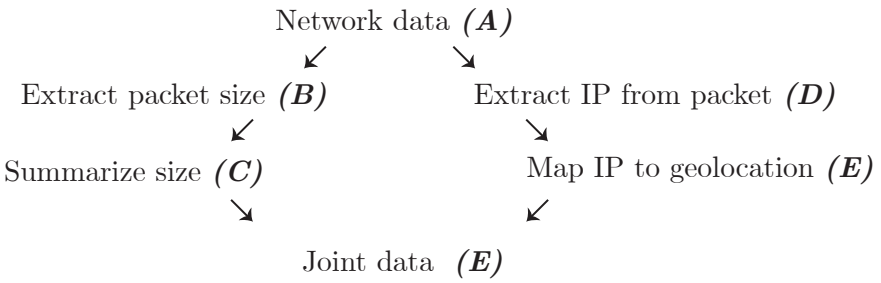

Abbildung 4.5: Beispielhafte Data-Pipelines auf Basis von Topics

4.3.2.1 Producer

Am Anfang einer jeden Anwendung steht die Wahl einer geeigneten Datenquelle. Hierbei ist es nebensächlich, wie und in welchem Umfang eine spätere Verarbeitung und Analyse stattfindet. Es gilt zu gewährleisten, dass ein Stream von Daten bereitgestellt wird. Dieser Stream sollte somit kein punktuelles Datenvolumen für das System bereitstellen, sondern kontinuierlich Daten liefern beziehungsweise transportieren.
Bei der eigenen Plattform lassen sich die folgenden Anforderungen zum allgemeinen Vorgang des Datentransports zusammenfassen. Zunächst gilt es, Daten unabhängig von der Quelle für die Plattform bereitstellen zu können. Zu erkennen ist, dass Netzwerkdaten nicht ausschließlich per CLI oder allgemeiner Netzwerk-Interface an die Plattform konnektierbar sein müssen. Daher ergibt sich bei heterogenen Datenquellen die Herausforderung, dass die Producer-Komponente bestenfalls mit diesen Unterschieden umgehen kann. Konkret heißt das, verschiedene Datenquellen sollten mit ein und derselben Producer-Komponente beziehungsweise API mit der Plattform konnektierbar sein.
Für die Konzeption der Producer lässt sich somit das folgende Verhalten definieren: Ein Producer ist eine eigene Instanz auf dem System, das eine oder mehrere Datenquellen für einen

zu analysierenden Kontext bereitstellt. Dabei ist ein Producer als unabhängige Komponente außerhalb des Betriebs der Plattform anzusehen. Die Kopplung zur Plattform besteht nur über die definierten Topics der konkreten Anwendung auf Basis der Plattform. Die Abbildung 4.6 stellt das beschriebene Zusammenwirken der einzelnen Bereiche dar.

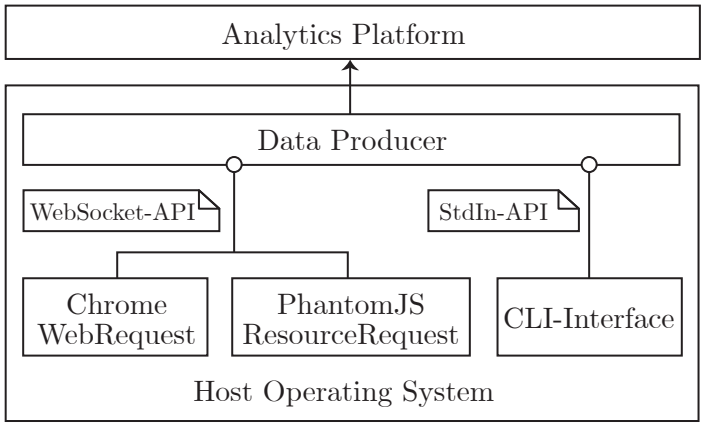

Abbildung 4.6: Schematisches Zusammenwirken von Data Producer und Datenquellen

Die Möglichkeiten, Daten an einen Producer zu senden, sollten vielseitig sein. Das bedeutet, dass wie zuvor erwähnt nicht nur über das CLI eine Möglichkeit bereitgestellt wird. Die Abbildung 4.6 zeigt zwei angebotene Schnittstellen, auf deren Basis eine Datenquelle mit dem Producer konnektiert werden kann. Im Einzelnen sollten die Schnittstellen so generisch sein, dass bei einer größeren Bandbreite an konkreten Analyseszenarien eine von mehreren Schnittstellen verwendet werden kann ohne etwa eine neue Schnittstelle implementieren zu müssen.

Datenformat Das Datenformat ist ein weiterer wichtiger Punkt in der Konzeption des Zusammenwirkens aller Komponenten. Hierbei sind je nach Datenquelle verschiedene Formate existent. Am Beispiel von CLI-Werkzeugen wie *tcpdump*, *tshark* oder *windump* ist häufig das Dateiformat PCAP[7] vorzufinden. Dabei handelt es sich um ein binär kodiertes Dateiformat. Alle genannten CLI-Werkzeuge wie auch Wireshark[8] bieten die Persistierung wie auch die Analyse (im Fall von Wireshark) entsprechender PCAP-Dateien an. Üblicherweise werden diese Daten aber zumeist direkt in ein entsprechendes Filesystem geschrieben.
Bei der konkreten Plattform widerspricht dieses Verhalten aber dem gesetzen Ziel der Plattform.

[7] http://www.tcpdump.org/manpages/pcap.3pcap.html
[8] https://www.wireshark.org/

Insofern müssen Datenstreams erstellt werden, indem ein Format verwendet wird, das durch die Data Prozessoren verarbeitet werden kann.
Im beschriebenen Fall sieht die Konzeption somit vor, dass der Producer grundsätzlich unabhängig vom Datenformat arbeiten kann. Die Deserialisierung ist Bestandteil der Daten Prozessoren, die mit Hilfe entsprechender Klassen diese Aufgabe zu bewältigen haben.

4.3.2.2 Streams

Der Bereich der Verarbeitung und Analyse von Daten beziehungsweise Datenstreams basiert im Allgemeinen auf der Verwendung der Streams-API von Apache Kafka. Diese Schnittstelle stellt das Bindeglied zwischen der Datengenerierung und der Ergebnisausgabe dar. Dabei nutzen die sogenannten Daten Prozessoren die Streams-API. Die Systemarchitektur von Apache Kafka baut beim Messaging auf eine Verbindung der beiden Modelle Queueing und Publish/Subscribe. Der Vorteil dieser Kombination ist die Möglichkeit der Verteilung der Datenprozessoren auf verschiedene Knoten trotz der Verarbeitung ein und desselben Topics.

Data Processor Die Konzeption der eigenen Plattform sieht vor, sich die beschriebenen Vorteile in der Art zunutze zu machen, dass jeder Prozessor in einem Container isoliert und durch die Containerverwaltung (siehe 4.3.1.1) einem konkreten Knoten zugeführt wird.
Ein konkretes Analysevorhaben definiert wegen der Komplexität der jeweiligen Verarbeitungsschritte die Struktur und Landschaft der jeweiligen Datenprozessoren in den Docker-Containern. Das bedeutet: Sofern am Anfang einer Verarbeitungskette schwergewichtige Transformationen an einem Datenstrom vorgenommen werden müssen, sollte die Anzahl der Datenprozessoren an dieser Stelle größer gegenüber der Anzahl der leichtgewichtigeren Transformationen sein. Auf diese Weise definiert die Anwendung auf Basis der Plattform die Struktur und Anzahl der Daten Prozessoren pro Topic.

Implementation von Daten Prozessoren Die Streams-API bietet grundsätzlich zwei verschiedene Möglichkeiten der Implementation von Daten Prozessoren. Die erste Möglichkeit ist die Streams-DSL. Diese bietet eine rudimentäre Möglichkeit der Datentransformation auf Basis zweier primärer Abstraktionen: KStream und KTable (siehe 2.4.2.1). Damit lassen sich gängige Transformationen der Daten in Streams implementieren. Diese DSL wird auch als „High-Level“ beschrieben.
Außerdem wird eine „Low-Level“ API angeboten. Mit Hilfe dieser muss das *Processor*-Interface

implementiert werden. Für die Verarbeitung kann eine der beiden Varianten genutzt werden um einen entsprechenden Daten Prozessor zu implementieren. Jeder Prozessor konnektiert sich mit einem Datenstrom über das entsprechende Topic. Der Name des Topics sollte variabel an den Docker-Container durch Environment-Variables übergeben werden und am entsprechenden Entrypoint als Parameter beim Starten des Datenprozessors gesetzt werden können.

4.3.2.3 Consumer

Die Verwendung der Consumer-API ist für alle Systemteile von Relevanz, die Daten aus Streams übernehmen oder relevante Verarbeitungsergebnisse ausgeben. Dabei fungiert diese Schnittstelle für die eigene Plattform als Bindeglied zwischen der Datentransformation und der Datenvisualisierung. Die Visualisierung von Verarbeitungsergebnissen soll über ein Web-Front-End gewährleistet werden. Bei diesem Prozess dienen Widgets (siehe 3.8) als kleine konkrete Front-End-Anwendungen, die entsprechende Topics abonnieren können.
Ein Vorteil der Daten Prozessoren ist es, Topics lesen und schreiben zu können. Da diese direkt mit der entsprechenden API von Kafka kommunizieren, existiert in diesem Fall keine Indirektionsstufe. Das eigene Konzept sieht aber vor, dass eine Kommunikation mit der Streams-API auch durch Benutzerinteraktion ermöglicht werden soll. Hierfür muss somit eine Schnittstelle geschaffen werden, sodass Widgets im Front-End trotzdem vom Message-Passing der Streams-API profitieren können. Dabei entsteht aber zwangsläufig eine Indirektionsstufe über ein Web-Front-End taugliches Protokoll.
Die Standardprotokolle HTTP und HTTPS sind für diese Aufgabe nicht geeignet, da keine bidirektionale Kommunikation gewährleistet werden kann. Aus diesem Grund bietet sich aber das WS beziehungsweise WSS-Protokoll an.

Web Socket Widgets Die zuvor beschriebene Notwendigkeit der Kommunikation mit dem Kafka Cluster ist eine Grundvoraussetzung für die zu entwickelnden Widgets. Die Aufgabe der Widgets besteht in erster Linie in der Repräsentation von etwaigen Verarbeitungsergebnissen der Daten Prozessoren. Für die beschriebene Stufe der Indirektion bedarf es einer Komponente im Back-End oder genauer im Verbund des Clusters, die als Bindeglied zwischen dem Web-Front-End und der Streams-API fungiert. Die Abbildung 4.7 stellt das Konzept der Verbindung der jeweiligen Plattformbereiche beispielhaft dar.

4.3.2.4 Connect

In der Analyse wurde bereits auf die Anforderung der Echtzeitfähigkeit der eigenen Plattform eingegangen. Fest steht auch, dass der Fokus auf dem Stream-Processing und nicht auf dem

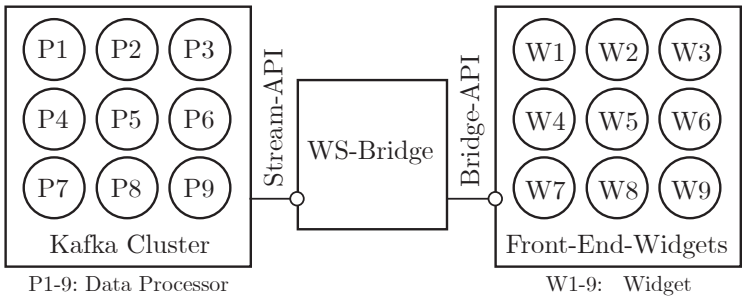

Abbildung 4.7: Konzeption Bridge-API für Front-End-Widgets

Batch-Processing liegt (siehe 3.5). Aus dieser Zielsetzung leitet sich bei der Datenakquise ab, dass keine existierende Datenbestände in die Plattform geleitet werden müssen. Die Connect-API der Kafka Plattform bietet aber grundsätzlich die Möglichkeit Daten anderer Systeme mit Kafka zu verbinden. Dafür ist die Connect-API konzipiert.
Auf Grundlage der eigenen Plattform, die unter anderem auf Kafka fußt, ist diese Schnittstelle zwar weniger wichtig für das beschriebene Einleiten von Daten in das System, sie ist aber trotzdem eine Möglichkeit der Kommunikation mit Datenbanken allgemein.
Dennoch kann mit Hilfe der Schnittstelle die Möglichkeit geschaffen werden, Widgets über die Bridge-API und Topics mit einem Connector zu verbinden um Daten dauerhaft zu persistieren. Diese Möglichkeit besteht natürlich auch ohne eine Indirektion über die Bridge-API.

4.3.3 Zookeeper

Der Apache Zookeeper ist das Werkzeug um die einzelnen Kafka-Instanzen (Knoten) zu orchestrieren. Der Zookeeper ist eine der Komponenten, mit der keiner der Akteure aus dem Kapitel Analyse direkt interagieren muss. Trotzdem ist der Zookeeper unabdingbar für den Betrieb der Plattform. Im Kapitel Grundlagen wurde bereits auf die Kernpunkte des Zookeepers eingegangen (siehe 2.4.3). An dieser Stelle der Konzeption ist primär der Fakt von Bedeutung, dass der Zookeeper als reine Laufzeitkomponente im Kontext der eigenen Plattform anzusehen ist. Das heißt, sie ist eine Komponente zur Verwaltung von Kafka-Brokern. Somit sind Zookeeper-Instanzen Bestandteil einer jeden konkreten Anwendung zur Analyse, bedürfen aber nach initialer Konfiguration keiner weiteren Implementation.

4.3.4 PhantomJS

In der Analyse wurden zwei mögliche Szenarien skizziert, in denen die eigene Plattform zum Einsatz kommen kann. Im Fall des „The onion router (Tor)“-Netzwerks ergibt sich aus dessen Konzeption ein hoher Grad an Kommunikationsverschlüsselung und Anonymisierung. Bedarf es der Entschlüsselung von Kommunikationsströmen, so ist dies auf der Ebene von CLI nur schwerlich möglich. Deswegen kann PhantomJS[9] zum Einsatz kommen. Dabei handelt es sich um eine programmierbare Browser-Engine, die im Betrieb aber „headless“ läuft. Das bedeutet, es existiert kein GUI, in dem Webseiten visuell gerendert werden.
Dieses Tool kann somit in Verbindung mit einer der beiden Data Producer API dazu verwendet werden, Netzwerkdaten direkt über eine verbundene PhantomJS Instanz mit Tor als Datenquelle zu nutzen.

4.4 Zusammenfassung

Das Kapitel Konzeption hatte zum Ziel die verwendeten Plattformen, Frameworks und Bibliotheken im Kontext der eigenen Plattform zu beleuchten. Zu Anfang wurde die Bedeutung von Docker beschrieben. Unter den Gesichtspunkten der Verteilbarkeit und Skalierbarkeit waren dies die zentralen Punkte, die im Kontext der Echtzeit-Datenanalyse von großer Bedeutung sind. Weiterhin fanden dann die verschiedenen Einsatzumgebungen Erwähnung bezogen auf die Bereiche der Entwicklung und Produktion.
Es wurden auch die Komponenten im Verbund zueinander beschrieben, da diese den Kern der eigenen Plattform bilden.
Im Kapitel Plattform wurde die Systemarchitektur und das Zusammenwirken der einzelnen Systembereiche behandelt. Die einzelnen Komponenten wurden dann im Kapitel Komponentenarchitektur erwähnt. Sie wurden detaillierter beschrieben wie auch deren Konnektivität zu anderen Komponenten über die entsprechenden API’s.

[9] http://phantomjs.org/

5 Anwendung: Request Intersection

Die Analyse, Konzeption und Realisierung der Plattform zur Echtzeit-Netzwerkdatenanalyse sind die zentralen Kapitel dieser Arbeit. Die wichtigen Eckpunkte der Plattform wurden in den genannten Kapiteln herausgearbeitet. In den folgenden beiden Kapiteln werden nun zwei Szenarien beschrieben, in denen die Plattform zum Einsatz kommt.
In dem Analysekapitel (siehe 3.6) wurden bereits zwei einleitende Szenarien skizziert, die als Ausgangspunkt der nun folgenden Anwendungen dienen beziehungsweise als eine thematische Einführung zu verstehen sind.

5.1 Einleitung

Die erste Anwendung zur Analyse von Netzwerkdaten befasst sich mit der Untersuchung von Website-Requests verschiedener Primärseiten und etwaiger Sekundärseiten. Dabei ist unter der Primärseite die Webseite zu verstehen, die direkt als Webadresse aufgerufen wird. Als Sekundärseite sind die Ressourcen der Webseite zu verstehen, die im Nachgang des initialen Primäraufrufs auf verschiedene Weise durch die Webseite angefordert werden und unter anderem auch für die Darstellung der Webseite relevant sind. Auf das genaue Verhalten wird in den Grundlagen (siehe 5.2) noch genauer eingegangen.
Es gilt in der Anwendung „Request Intersection“ (dt. Schnittmenge von Anfragen) zu untersuchen, ob und wie eine Deckung von Servern zwischen verschiedenen Webseiten besteht und ob sich diese in Kategorien gliedern lassen.
Aus den erzielten Ergebnissen lassen sich dann verschiedene Rückschlüsse ziehen. Zum einen werden Erkenntnisse über eine verteilte und gemeinsam genutzte Infrastruktur von verschiedenen Webseiten erlangt. Zum anderen lässt sich aber auch erkennen, welche „Drittanbieter“ über das Surfverhalten der Besucher ihrer Kunden Kenntnis erlangen können und wie sich das quantitativ beziffern lässt.
Dieses Szenario wird in diesem Kapitel auf ausgewählten Webseiten angewandt und befasst sich von vornherein nur mit dieser definierten Auswahl. Das Szenario ließe sich grundsätz-

lich aber auch generischer in der Hinsicht konzipieren, dass man nicht-definierte Webseiten, sondern einen History-Stream etwaiger Benutzer als Grundlage nimmt.

5.2 Grundlagen

In diesem Abschnitt werden technische Grundlagen beschrieben, da diese die Voraussetzung für das Verständnis der gewählten Datenanalyseverfahren sind.
Am Anfang wird das Request-Response-Verhalten eingehender erläutert. Der Grund dafür ist, dass jeder Request im Folgenden als ein Datensatz des Datenstreams fungiert und dadurch die Grundlage des Analyseverhaltens darstellt.
Im Anschluss an den Abschnitt Request-Response wird dann auf das Ladeverhalten externer Ressourcen von HTML-Dokumenten eingegangen. Diese werden im Weiteren auch als Request oder Target bezeichnet. Jede dieser Ressourcen verursacht somit einen Request an die Primär- oder Sekundärseite.

5.2.1 Request-Response

Das Protokoll HTTP sowie die Erweiterung HTTPS mit einer Transportverschlüsselung der Daten sind die beiden Protokolle im Open Systems Interconnection (OSI) Modell, die auf der Anwendungsschicht angesiedelt sind.
Das Protokoll weist die Eigenschaft der Zustandslosigkeit auf, sodass alle Requests unabhängig voneinander behandelt werden (Oyman und Singh, 2012). Dieses grundsätzliche Paradigma der Zustandslosigkeit ist auch bei Hypertext Transfer Protocol 2 (HTTP2) erhalten geblieben (Nwosu u. a., 2012).
Für die Analyse des Datenverkehrs von Webseiten sind alle Anfragen an Primär- oder Sekundärserver klar voneinander trennbar. Das Request-Response-Verfahren eines Clients an einen Server zeichnet sich bei Webseiten zudem durch einen initialen Request aus, wobei dieser weitere Requests verursachen kann. Die Kausalität ist dabei aber nicht im Zustandsverhalten des Protokolls zu suchen, sondern im Inhalt des initialen Response. Die Response auf einen Seitenaufruf liefert dabei ein HTML-Dokument, in dessen Quelltext weitere Ressourcen referenziert sind, die durch die Browser-Engine zum Zeitpunkt des Renderns angefragt werden.
Für die spätere Analyse lässt sich konstatieren: Ein initialer Request auf eine Adresse *domain.com* liefert eine Response mit einem HTML-Dokument. Darin können weitere Ressourcen definiert sein, die zu weiteren Requests führen. Diese Ressourcen sind aber nicht zwangsläufig bei dem Primärserver (hier *domain.com*) hinterlegt, sondern können auch auf anderen Servern (Sekundärserver) verortet sein.

5.2.2 External Resources

Eine einleitende Beschreibung des Request-Response-Verhaltens erfolgte bereits im vorherigen Abschnitt. Es lassen sich im Allgemeinen verschiedene Arten von Ressourcen klassifizieren, die in einem HTML-Dokument referenziert werden können. Dabei zählen die folgenden drei HTML-Tags wohl zu den gebräuchlichsten:

Tag	Attribute	Beschreibung
script	src	Referenziert externe Skript-Dateien: Diese werden zur Laufzeit geparsed und interpretiert.
link	href	Referenziert externe CSS-Dateien: Im Allgemeinen sind das Definitionen zur Darstellung von HTML-Dokumenten.
img	src	Referenziert ein externes Mediaobjekt: Das sind zum Beispiel komprimierte Bilddateien.

Tabelle 5.1: HTML-Resource-Types

Weitere Ressourcen können unter anderem durch Tags wie *audio*, *video*, *iframe* oder *object* deklariert werden. Dabei lässt sich eine Klassifikation anhand der Unterschiedlichkeit des Inhalts herstellen. Dazu gehören unter anderem Flow, Phrasing und Embedded-Content (w3.org, 2017).
In den späten neunziger Jahren wurden die verschiedenen Arten von Ressourcen zumeist auf nur einem Server gespeichert. Daraus folgte, dass es keine Unterscheidung in Primär- und Sekundärserver gab. Mit dem Fortschreiten der Entwicklung des World Wide Web fanden sich aber zunehmend unterschiedliche Ansätze, bei denen eine stärkere Verteilung der Ressourcen stattfand. Ein Beispiel sind hierfür Mediaobjekte, die zumeist ein deutlich größeres Datenvolumen aufweisen gegenüber den zugrundeliegenden HTML-Dokumenten. Somit ist der Faktor der Serverauslastung in Bezug auf Anfragen sowie Transport dieser Mediaobjekte wichtig in Hinblick auf die Performance von Webseiten. Außerdem ist ein Transport von Mediaobjekten von nur einer Quelle an verschiedene anfragende Clients zumeist auch eine Ursache für eine hohe Netzauslastung. Desswegen wurden unter anderem geolokations-basierte Systeme entwickelt, die diesen Problemen begegnen. Dabei sind Content Delivery Networks beispielhaft hervorzuheben.

5.3 Analyse

In diesem Analyseabschnitt werden zentrale Verhaltensweisen herausgestellt und beschrieben, die ursächlich für die Zielsetzung der Anwendung sind. Es werden zunächst die Bereiche Host-Detection sowie Content-Types näher erläutert und analysiert.

5.3.1 Host-Detection

Grundsätzlich geht es in der Anwendung um eine Identifikation und Gruppierung von Resource-Hosts. Das bedeutet, jede Ressource wird zunächst als ein unabhängiger Request betrachtet. Es ist vorab nur bekannt, dass eine externe Anfrage (Ressource) an einen Server (Host) gestellt wird. In Abbildung 5.1 ist der allgemeine Ablauf des Anfrageverhaltens von Ressourcen ersichtlich.

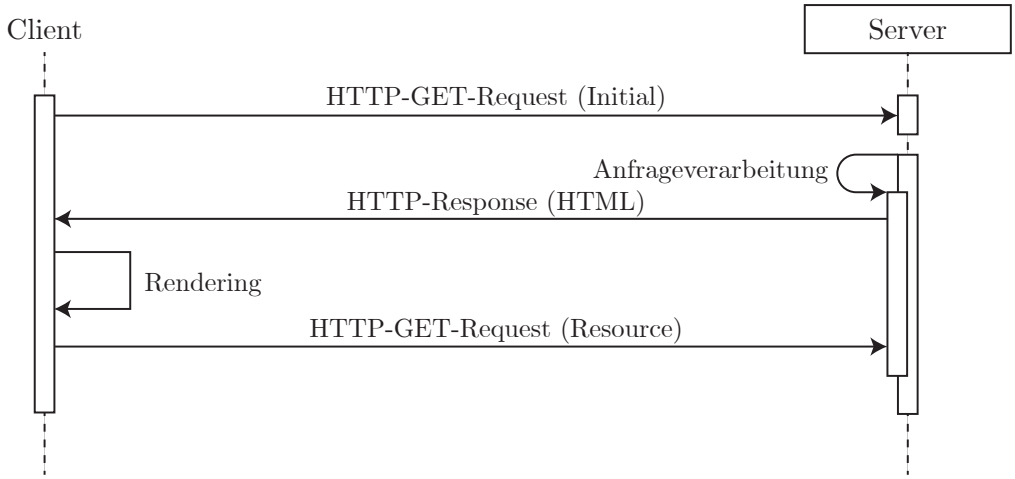

Abbildung 5.1: Sequenzdiagramm von Ressource-Anfragen

Das Ziel dieser Anwendung ist es, eine schlussendliche Gruppierung von Host-Servern und nicht der jeweils angefragten Ressourcen vorzunehmen. Dabei lässt sich eine unterschiedliche Granularität der jeweiligen Host-Requests erreichen.
Das Auflösen einer Uniform Resource Locator (URL) *www.domain.com* geschieht durch Domain Name System (DNS)-Server, wobei diese sich in Zonen beziehungsweise Ebenen untergliedern. Die Reihenfolge der Auflösung ist in Abbildung 5.2 ersichtlich.

. => com => domain => www

Abbildung 5.2: DNS-Namensauflösung *www.domain.com.*

Zu vernachlässigen sind bei der Anwendung die Auflösungen bis zur Top-Level-Domain. Von Interesse hingegen sind die Auslösung einer Domain selbst sowie der Sub-Level-Domains. Durch das DNS-Protokoll ist für *www.domain* im jeweiligen lokalen DNS ein eindeutiger Server mit entsprechender IP-Adresse definiert. Bei abweichender Sub-Level-Domain hingegen (zum Beispiel *img.domain*) kann wiederum eine andere IP-Adresse hinterlegt sein. Konkret heißt das für das Analysevorhaben: Zwei verschiedene Requests nach dem folgenden Schema in Tabelle 5.2 werden als zwei verschiedene Host-Resource-Anfragen behandelt.

URL-Request
http://www.domain.com/js/main.js
http://scr.domain.com/js/main.js

Tabelle 5.2: URL-Resource-Requests

5.3.2 Content-Types

Im Kapitel Grundlagen dieser Anwendung wurde bereits das Request-Response-Verhalten von Client und Server beschrieben. Im Anschluss daran wurden die wichtigsten Arten von externen Ressourcen behandelt, die in HTML-Dokumenten referenziert werden können. Zu beachten für die Möglichkeiten eines Drittanbieters (Sekundärserver) sind aber auch die verschiedenen Arten der Ressourcen.
Im Fall von referenzierten Mediadaten lassen sich auf Seiten der Sekundärserver nur bedingt Informationen über den Anfragesteller (Request an Sekundärserver) sammeln. So lassen sich zum Beispiel über die referenzierte URL auch kodierte ID's als Parameter übertragen, anhand derer eine Identifizierung des Anfragestellers geschehen kann. Dieses Verhalten lässt sich auch auf andere Arten von referenzierten Dokumenten übertragen, zum Beispiel auf Stylesheets.
Ein anderes Verhalten lässt sich aber bei Skriptdokumenten herbeiführen. Im Fall einer extern referenzierten Skriptdatei eines Sekundärservers werden auch Programmlogiken mit übertragen. Diese Skriptdateien sind zumeist in JavaScript geschrieben. Jede Webbrowser-Engine lädt dann zur Laufzeit diese Scriptdateien und lässt sie durch den JavaScript-Interpreter ausführen. Der Zweck von JavaScript in Webbrowsern ist es, eine clientseitige Sprache zur Benutzerinteraktion sowie auch Manipulation des Document Object Model (DOM) bereitzustellen. Auf diese Weise sind vielfältige Möglichkeiten vorhanden, die Inhalte des Primärservers zu modifizieren.

Im negativen Fall spricht man hierbei von Cross Site Scripting (XSS). Im positiven Fall sind es Services wie Google Analytics[1].

5.3.3 Anforderungen

In diesem Abschnitt werden die Anforderungen der konkreten Anwendung (5) spezifiziert. Die Anforderungen werden herausgearbeitet und nach funktionalen, technischen und nicht-funktionalen Eigenschaften untergliedert. Es finden dieselben Bezeichner wie in Abbildung 3.5 der Analyse (3) Verwendung. Das Akronym „RI“ bezeichnet die Anwendungen „Request Intersection“.

5.3.3.1 Funktionale Anforderungen

RI-FA-1
Die Daten werden mit Hilfe der *webRequest*-API von Chrome akquiriert.

RI-FA-2
Die Daten werden über Streams der Plattform zugänglich gemacht.

RI-FA-3
Die Daten werden verarbeitet und führen zu neuen Datenstreams.

RI-FA-4
Die verarbeiteten Daten werden durch entsprechende Möglichkeiten der Visualisierung dargestellt.

5.3.3.2 Technische Anforderungen

RI-TA-1
Trotz steigendem Datenaufkommen leidet die Verarbeitung nicht durch die zugrundeliegende Struktur.

5.3.3.3 Nicht-funktionale Anforderungen

RI-NFA-1
Innerhalb von fünf Sekunden sollten Verarbeitungsergebnisse erkennbar sein.

[1] http://analytics.google.com/

RI-NFA-2
Strukturinformationen über die Beziehung von Hosts zueinander sollen ersichtlich werden.

5.4 Konzeption

Die vorherigen Abschnitte haben die wichtigsten technischen Gebenheiten im Kontext des Vorhabens beleuchtet. Auf diesem Wege sind die Grundlagen für die Konzeption geschaffen worden, sodass das eigentliche Ziel konkretisiert werden kann.
Es gilt mit der Anwendung zu untersuchen, in welcher Art verschiedene Webseiten zueinander in Verbindung stehen. Das geschieht auf der Basis ihrer extern referenzierten Ressourcen. Ein weiteres Ziel ist es, eine visuelle Repräsentation zu erlangen, anhand derer man den Zusammenhang dargestellt bekommt. Die Echtzeitfähigkeit der Anwendung ermöglicht es somit, direkt die Verbindung von Primär- und Sekundärservern zu erkennen und daraus Rückschlüsse auf etwaige Services von Drittanbietern zu ziehen.

5.4.1 Untersuchungsumgebung

Das Renderingverhalten verschiedener Webbrowser kann zwischen verschiedenen Engines wie Blink, Gecko oder Trident variieren. Außerdem kann auch die JavaScript-Engine Einfluss auf das Webbrowserverhalten im Allgemeinen haben. Webbrowser-Extensions sind zumeist Erweiterungen, die nicht originär mit einem Webbrowser ausgeliefert werden, sie aber durchaus auch Einfluss auf Webseiten haben. Hierbei sind zum Beispiel Ad-Blocker zu nennen. Aus den genannten Gründen kommt der Webbrowser Google Chrome (Version 56.0.2924.87) auf macOS (Version 10.12.3) ohne eine Vorkonfiguration zum Einsatz. Einzig eine Extension zum Mitschneiden des Netzwerkdatenverkehrs wird aktiviert, die aber keinen Einfluss auf das Rendering hat.

5.4.2 Host-Request-Mapping

Die Unterscheidung in Primär- und Sekundärseiten wurde zuvor in der Einleitung erwähnt. Somit sind in dem konkreten Analyseszenario die zehn Webseiten der Nachrichtenportale die Primärseiten.
Jede dieser Nachrichtenseiten ist - wie eingangs beschrieben - von externen Ressourcen abhängig. Dabei sind die Arten der Ressourcen im konkreten Fall mannigfaltig. Es werden

Mediaobjekte wie Bilder geladen sowie Darstellungsanweisungen in Stylesheet-Dateien.
Von Interesse sind aber die externen Ressourcen, die von Serviceanbietern wie Google Analytics geladen werden. Der personalisierte Tracking-Code wird dabei zumeist als Parameter in der URL mit übergeben oder auch nach dem initialisierten Laden der Bibliothek. Somit unterscheiden sich in diesem Fall die Anfragen von *spiegel.de* und *welt.de* nicht im Bestandteil des Hostname.
In diesem konkreten Fall wäre das Laden der Google Bibliothek von einem Google Server eine Sekundärseite beziehungsweise ein Sekundärrequest. Auf dieser Grundlage lässt sich eine Datenstruktur definieren, anhand derer ein Host-Mapping vorgenommen werden kann.
Aus den beschriebenen Sachverhalten unterhält jeder Primärrequest eine Menge von Sekundärrequests. Von Interesse sind aber eigentlich die Sekundärrequests, beziehungsweise von welchen Primärrequests eine Schnittmenge mit Sekundärrequests besteht. Im Fall der zehn Nachrichtenportale kann somit zum einen angenommen werden, dass diese einzigartig und eindeutig sind. Die Sekundärrequests hingegen können von verschiedenen Primärseiten initiiert werden und können somit auf verschiedenen Primärseiten zustandekommen.

5.5 Anwendung

In diesem Abschnitt werden die drei zentralen Bereiche der konkreten Anwendung auf der Basis der Plattform beschrieben. Dabei werden die Implementationen partiell vorgestellt. Außerdem liegt der Fokus auf dem Ziel des Analyseszenarios. Beschrieben werden: Data Producers, Data Processors sowie Widgets zur Ergebnispräsentation beschrieben.

5.5.1 Data Producer

In dem konkreten Szenario geht es allgemein um die Analyse von Netzwerkdatenverkehr von Webseiten. Es wird der Chrome Webbrowser zum Einsatz kommen. Als entsprechende API bietet sich dabei die WS-API als Producer an. Auf der Seite des Webbrowsers bietet sich zum Mitschneiden die WebRequest-API[2] an. Diese ermöglicht es, jeden getätigten Request als ein JSON-Objekt zu erhalten. Die Objektstruktur ist in Abbildung 5.1 ersichtlich. Das abgebildete JSON-Objekt entspricht einem Record des Datastreams. Im abgebildeten Fall ist somit für die Primärseite *www.spiegel.de* ein Sekundärrequest an *stat.flashtalking.com* gestellt worden.

[2] https://developer.chrome.com/extensions/webRequest

```
{
    "frameId": 30,
    "method": "GET",
    "parentFrameId": 0,
    "requestId": "333",
    "statusCode": 200,
    "statusLine": "HTTP/1.1 200 OK",
    "tabId": 2,
    "timeStamp": 1487955719908.992,
    "type": "image",
    "url": "http://stat.flashtalking.com/reportV3/ft.stat
        ?76779349-2364708;1692066;4458055-310-0-333152BF353847
        -501502222-175x0x0x0",
    "responseHeaders": [
     {"name": "Server", "value": "Apache"},
     {"name": "Content-Type", "value": "text/plain"},
     ...
    ]
}
```

Listing 5.1: WebRequest-Object *www.spiegel.de*

5.5.2 Data Processor

Der Processor soll das Requestverhalten aller Primärseiten sowie Sekundärrequests verarbeiten und analysieren. Um diese Anforderung zu gewährleisten ist das Windowing (siehe 2.4.2.1) geeignet. Der Grund dafür ist, dass über einen definierten Zeitraum ein Mapping von Primär- sowie Sekundärrequests stattfinden muss. Im vorliegenden Fall beträgt die Zeitspanne 60 Sekunden. Durch das Windowing ist es in dieser Anwendung möglich, die angestrebte Übersicht der einzelnen Hosts zu erstellen.
Das Mapping der beiden Requestarten geschieht auf der Grundlage der Erkenntnisse aus der DNS-Namensauflösung (siehe 5.2). Als Ergebnis ist in diesem Szenario die Schnittmenge der Sekundärrequests zu den Primärseiten relevant. Desswegen eignet sich das Mapping aus Abbildung 5.3. Dabei wird ersichtlich, dass jeder Sekundärrequest eine Liste mit der Schnittmenge von Primärrequests führt. Hierbei ist der Prozessor so implementiert, dass nur Sekundärrequests mit einer minimalen Schnittmenge von zwei Primärrequests vorgehalten werden.

Sekundärrequest : [Primärrequest-*1* [, Primärrequest-*N*]]

Abbildung 5.3: Mapping im Daten Prozessor

5.5.3 Widget

Das Widget im Bereich des Front-Ends dient der visuelle Präsentation der Analyseergebnisse. Dabei ist in diesem Szenario eine Graph-Darstellung der Ergebnisse für die Verdeutlichung des Sachverhalts geeignet.
Ursächlich ist dafür, dass jeder Host eindeutig ist. Das heißt konkret, der komplette Hostname ist der Identifikator eines Knotens im Graphen. Zwei Arten von Knoten sind zu unterscheiden: die Primärknoten sowie Sekundärknoten. Es existieren ausschließlich Kanten zwischen Primär- und Sekundärknoten, aber keine Kanten zwischen Knoten des gleichen Typs. Auf diese Weise lässt sich jederzeit der aktuelle Stand und das Requestverhalten visuell als Graph abbilden, die Zusammenhänge der getätigten Anfragen werden dadurch ersichtlich.

5.6 Ergebnisse

In diesem Abschnitt werden die Ergebnisse dargestellt, die auf Grundlage der zuvor beschriebenen Producer, Processor und Widgets erlangt wurden. Primär steht die Ansicht im Dashboard im Vordergrund, da hierüber im Livebetrieb direkt die Verarbeitungsergebnisse ersichtlich sind.

5.6.1 Untersuchungskontext

In der Anwendung werden zehn große deutsche Nachrichtenwebseiten untersucht. Es werden die folgenden Webseiten als Datenbasis für die getätigten Requests der jeweiligen Startseiten herangezogen. Die Auflistung der Webseiten ist in Tabelle 5.3 ersichtlich.

5.6.2 Intersection Graph

Die Abbildung 5.4 ist ein Screenshot aus dem Livebetrieb der Anwendung „Request intersection“. Darin sind die zuvor beschriebenen Primär- und Sekundärrequests als Graph modelliert. Ersichtlich wird - unter anderem auch durch die Interaktivität des Graphen im Browser - welche Primärseiten untereinander eine Schnittmenge an Sekundärrequests aufweisen. Im konkreten Fall zeigt sich, dass eine benutzerdefinierte Mindestanzahl von fünf Primärrequests an einem Sekundärrequest vorliegen muss, damit die Verbindung im Graphen modelliert wird. Es zeigt sich im konkreten Beispiel, dass der Sekundärrequest *script.ioam.de* einen Deckungsgrad von zehn aufweist. Das heißt konkret, dass alle zehn definierten Primärseiten (siehe Tabelle 5.3) etwaige Inhalte über den Host *ioam.de* beziehen. Wenn man recherchiert, was sich hinter *script.ioam.de* verbirgt, stößt man auf die INFOnline GmbH. Diese beschreibt sich

#	Name	Einstiegspunkt
1	bild.de	http://www.bild.de/
2	faz.net	http://www.faz.net/
3	focus.de	http://www.focus.de/
4	handelsblatt.com	http://www.handelsblatt.com/
5	n-tv.de	http://www.n-tv.de/
6	spiegel.de	http://www.spiegel.de/
7	stern.de	http://www.stern.de/
8	taz.de	http://www.taz.de/
9	welt.de	http://www.welt.de/
10	zeit.de	http://www.zeit.de/

Tabelle 5.3: Auflistung der untersuchten Nachrichtenportale

selbst in dem Bereich des „Digital Audience Measurements" vertreten zu sein, wobei sie unter anderem „[...] Zugriffe auf Ihre Websites, Apps und mobile-enabled Websites." (INFOnline GmbH, 2017) messen.
Ein identischer Deckungsgrad von zehn lässt sich auch bei *securepubads.g.doubleclick.net* feststellen. Dabei handelt es sich um eine Marke von Google, die im Bereich des Online-Marketings angesiedelt ist. Primär ist das Aussteuern von Werbung auf Webseiten betroffen, wie sich im konkreten Fall der zehn Primärseiten zeigt.

5.6.3 Zusammenfassung

In Abbildung 5.5 werden alle weiteren Schnittmengen von Primärseiten über Sekundärseiten in einer Matrix zusammengefasst. Es gilt auch hier ein minimaler Deckungsgrad von fünf.
Im Ergebnis zeigt sich, dass zum Beispiel die Primärseiten *bild.de*, *focus.de*, *handelsblatt.com* und *zeit.de* bei neun der zehn abgebildeten Sekundärseiten Übereinstimmungen haben. *bild.de* und *focus.de* unterscheiden sich voneinander aber in dem jeweils fehlenden Service. Hingegen weisen *handelsblatt.com* und *zeit.de* diesbezüglich keinen Unterschied auf.

5.6.4 Kategorisierung der Drittanbieter

Ein einfacher Sekundärrequest an sich ist vor dem Hintergrund des Nutzens solcher Cross-Website-Requests wenig aussagekräftig. So bedarf es einer eingehenderen Kategorisierung

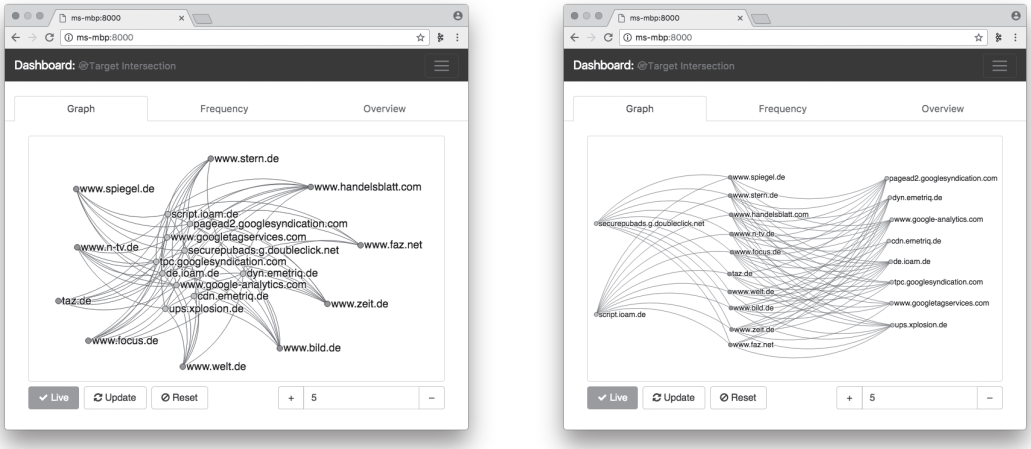

Abbildung 5.4: Intersection Graph: Primär- und Sekundärrequests

der jeweiligen Sekundärrequests. Die Requests an *doubleclick.net* sowie *ioam.de* weisen eine Deckung von 100% auf. Ihr Zweck wurde in 5.6.2 eingangs beschrieben. Im Bereich des Online-Marketings sind außerdem Requests an *googlesyndication.com* anzusiedeln. Über den Host Pay-Per-Click werden Ereignisse genauer ausgesteuert. Bei *emetriq.de* handelt es sich um einen ähnlichen Anbieter wie *ioam.de*. Dabei vereinen sie „[...] unterschiedlichste Daten in einem kollaborativen Datenpool und stellen diese allen Partnern aggregiert und veredelt zur Verfügung“ (emetriq GmbH, 2017). Bei *google-analytics.com* handelt es sich wohl um den bekanntesten Anbieter von Website-Analysen. Der Host *googletagservices.com* ist außerdem auch unter Google Analytics zu subsumieren. Dabei ähnelt das Schema dem von *ioam.de*.

5.7 Fazit

Allgemein lässt sich zuerst einmal erkennen, dass heutige Webseiten nicht mehr - wie in den Anfängen - Daten von nur einem Server beziehen. Es sei ausgeklammert, um welche Art von Daten es sich handelt.
Das Netz, also der Verbund von Servern in der Gesamtheit, bildet somit immer mehr die infrastrukturelle Basis der gesamten Nachrichtenportale, da diese auf verschiedene Drittanbieter bauen. Die Dienste verschiedener Primärseiten sind häufig auf die Bereiche der Werbung oder des User-Trackings zurückzuführen. Im Ergebnis sind in dem aktuellen Szenario keine Content Delivery Network (CDN)-Requests für Mediaobjekte der jeweiligen Primärseiten ersichtlich. Allerdings ist verstärkt davon auszugehen, dass diese Netze bei großen Nachrichtenportalen

	bild.de	faz.net	focus.de	handelsblatt.com	n-tv.de	spiegel.de	stern.de	taz.de	welt.de	zeit.de
securepubads.g.doubleclick.net	•	•	•	•	•	•	•	•	•	•
script.ioam.de	•	•	•	•	•	•	•	•	•	•
pagead2.googlesyndication.com	•	•	•	•	•	•	•	•		•
dyn.emetriq.de	•	•	•	•	•		•		•	•
www.google-analytics.com	•		•	•	•	•	•	•	•	•
cdn.emetriq.de	•			•	•				•	•
de.ioam.de	•		•	•	•	•	•	•	•	•
tpc.googlesyndication.com	•		•		•	•	•	•		
www.googletagservices.com			•	•		•	•	•		•
ups.xplosion.de	•	•	•	•	•		•		•	•

Abbildung 5.5: Intersection Matrix: Primär- und Sekundärrequests

zum Einsatz kommen beziehungsweise genutzt werden.
Die Verwendung von Tracking-Mechanismen ist bei den getesteten Nachrichtenseiten grundsätzlich auf den Wert der zu erlangenden Informationen zurückzuführen, die in den Daten direkt oder indirekt enthalten sind. In der Konsequenz lassen sich die Angebote der jeweiligen Portale auf Grundlage der gewonnenen Erkenntnisse verbessern. Bei dieser Betrachtung liegt der Mehrwert direkt bei den Verlagen der Nachrichten-Portale. Trotzdem lässt sich aber auch eine indirekte Folge der Schnittmengen an Requests erkennen. Die indirekte Folge ist das Wissen über verschiedene Primärseiten hinweg einen Weg von Anfragestellern (Benutzern) verfolgen zu können. Im beschriebenen Fall von *ioam.de* ist somit unabhängig von der konkreten Analyse des geladenen Scripts die eigene Person in allen zehn Nachrichten-Portalen identifizierbar. Da jeder Primärrequest zwar nur an eine der jeweiligen Webseiten gestellt wird, aber beim Nachladen des Scripts von *ioam.de* ein Sekundärrequest gestellt wird, ist das gesamte Anfrageverhalten für *ioam.de* bekannt, nicht aber für die jeweiligen Primärseiten.
Abschließend lässt sich konstatieren, dass das konkrete Szenario hier nur in der Domäne deutscher Nachrichtenseiten getestet wurde, in heterogenen Domänen aber wahrscheinlich ähnliche Resultate erzielen würde.

6 Anwendung: Tor-Exit-Interceptor

Die zweite konkrete Anwendung dieser Arbeit befasst sich mit der Analyse von Teilen des Tor-Netzwerks[1]. Für diese Anwendung sind verschiedene Voraussetzungen zu berücksichtigen, die in den folgenden Abschnitten erläutert werden. Dazu zählen neben infrastrukturellen auch softwaretechnische Voraussetzungen.
In den Abschnitten Einleitung und Grundlagen werden zum einen die Zielsetzung der Anwendung „Tor-Exit-Interceptor“ und die zugrundeliegende Idee beschrieben. Zum anderen finden einige wichtige technische Bestandteile des Tor-Netzes Erwähnung, da diese für das Verständnis der Anwendung von Bedeutung sind.

6.1 Einleitung

Die Anwendung „Tor-Exit-Interceptor“ hat zum Ziel eine Analyse von Teilen des Tor-Netzwerks auf der Basis der entwickelten Plattform zu durchlaufen. Die Formulierung „von Teilen des Tor-Netzwerks“ bedarf einer genaueren Erklärung. Vorab wurde ein Tor-Exit-Node eingerichtet um genau an dieser Stelle den Netzwerkdatenverkehr untersuchen zu können. Somit bezieht sich die Formulierung auf den Datenverkehr an dem aufgesetzten Tor-Exit-Node.
Eine eingehendere Erläuterung des technischen Verhaltens eines Exit-Nodes beziehungsweise eines Tor-Relay-Servers findet in den Grundlagen statt. Vorab ist an dieser Stelle aber die eigentliche Zielsetzung von Interesse. Die Anwendung soll eine Möglichkeit bieten direkt und unmittelbar (in Echtzeit) grundlegende Datenflüsse am Exit-Node sichtbar zu machen. Damit ist gemeint, dass die Datenflüsse in der Art sichtbar gemacht werden können, dass nachvollziehbar ist, welche anderen Server vom Exit-Node aus angesprochen werden. Daraus lassen sich Rückschlüsse darüber erlangen, woran das Interesse von einigen Nutzern des Tor-Netzwerks besteht. Außerdem wird eine Analyse von Webseiten-Domains durchgeführt. Dabei soll gezeigt werden, welche Seiten aktuell vom Exit-Node aus am häufigsten aufgerufen werden. Auf diesem Weg sind dann zu einem jeweiligen Zeitpunkt oder während eines definierten Zeitraums die häufigsten Anfragen erkennbar.

[1] https://www.torproject.org/

6.2 Grundlagen

In den Grundlagen werden einige wichtige technische Gegebenheiten des Tor-Netzwerks erläutert, die von Bedeutung für die Zielsetzung dieser Anwendung sind. Dabei wird eingangs das Verhalten der verschiedenen Tor-Relay-Nodes beschrieben sowie im Besonderen das der Tor-Exit-Nodes.

6.2.1 Tor-Relay-Nodes

Grundsätzlich dient das Tor-Netzwerk dem Ziel der Anonymisierung seiner Nutzer, sodass möglichst keine Rückverfolgung einzelner Personen oder Gruppen vollzogen werden kann. Im offenen Internet, das auf dem Transmission Control Protocol (TCP)/IP basiert, sind - sofern bei entsprechenden staatlichen Behörden eine Genehmigung vorliegt - Benutzer über ihren Internet Service Provider (ISP) identifizierbar. Das bedeutet in einem konkreten Fall natürlich nicht zwangsläufig, dass der Anschlussbesitzer auch der alleinige Nutzer dieses Anschlusses ist. Dieser Aspekt wird an dieser Stelle jedoch in den Hintergrund gestellt.
Das Tor-Netzwerk, das ein Overlay-Netzwerk ist, basiert auch auf TCP/IP. Durch seine entsprechende Architektur trägt es zur Anonymisierung seiner Nutzer bei. Im Allgemeinen sind die Server des Tor-Netzwerks in verschiedene Gruppen zu gliedern. Es werden anhand verschiedener Kriterien sogenannte Flags den entsprechenden Relays zugeordnet. Zuerst einmal ist jeder Server, der einen Tor-Client als Daemon laufen lässt, ein Tor-Relay.
Bezogen auf die Anonymisierung werden beim Tor-Netzwerk drei Server im Verbund für eine sogenannte Schaltung (engl. circuit) gebraucht. Über diese drei Server wird eine Verbindung mit dem offenen Internet hergestellt. Hierbei stellt ein Tor-Client, also die Seite, an der eine Person sich in das Tor-Netz einwählt, eine Verbindung mit dem Guard-Node her. Diese erste Verbindung wird dabei zum ersten Mal verschlüsselt. Im Anschluss daran wird diese Verschlüsselung durch Tor ebenfalls beim Middle-Node sowie Exit-Node fortgesetzt. Daher wird auch vom Onion-Routing gesprochen, da bei jedem Knoten eine weitere Schicht der Verschlüsselung hinzukommt. Ein Charakteristikum ist es, dass jede Stelle in dieser Schaltung nur seine direkten Nachbarn kennt, sie aber keine Kenntnis über die gesamte Route hat.
Am Tor-Exit-Node werden die Daten in das offene Internet geleitet. Auf diesem Weg ist die Quelle - also der Anfragesteller - nicht bekannt, sodass das Tor-Netz beziehungsweise die verschiedenen Knoten die Anonymisierung gewährleisten.

6.2.2 Tor-Relay-Flags

Die zuvor erwähnten Relay-Flags werden in Tabelle 6.1 beschrieben. Genannt werden die Flags, die der laufende Tor-Exit-Node vorweisen kann. Für die Zielsetzung dieser Anwendung ist das Flag „HSDir“ weniger von Bedeutung, jedoch in etwaigen weiteren Anwendungen ließen sich partiell die Anfragen an Hidden Services im Tor-Netz (Darknet) nachvollziehen.

Flag	Beschreibung
Running	Das Running-Flag beschreibt die grundsätzliche Verfügbarkeit der Server.
Fast	Dieses Flag wird Relays zugesprochen, sofern diese eine hohe Bandbreite vorzuweisen haben.
Valid	Das Valid-Flag wird nach einer erfolgreichen Validierung durch das Tor-Netz einem Tor-Server zugewiesen.
Stable	Sofern das Stable-Flag zugewiesen wurde, kann der Server für langlebige Circuits verwendet werden. Das Flag bemisst sich nach der Laufzeit des Servers.
Guard	Der Server fungiert als Guard-Node und kann bei der Wahl von Circuits als Guard gewählt werden.
Exit	Der Server fungiert als Exit-Node und hat eine definierte Exit-Policy.
V2Dir	Ein Hinweis auf die Implementation des V2-Directory-Protokolls
HSDir	Ein Server trägt zu den Hidden Services bei Tor bei, wenn er das HSDir-Flag zugewiesen bekommt. Dabei trägt er zur „Namensauflösung“ von *.onion* Adressen bei.

Tabelle 6.1: Auflistung der unterschiedlichen Tor-Flags

6.2.3 Tor-Exit-Node

Für die Anwendung wurde ein eigener Tor-Exit-Node aufgesetzt und betrieben. Bei diesem Tor-Exit-Node handelt es sich um einen virtuellen Server im Netz des „Verein[s] zur Förderung eines Deutschen Forschungsnetzes e.V.“. Das Betriebssystem des virtuellen Servers ist Debian in der Version 8 (jessie).

Auf dem Server ist der Tor-Client in der Version 0.2.8.9 installiert und zur Ausführung gebracht worden. Dabei läuft der Server seit dem Start fast ausschließlich als Relay, sodass keine sogenannte Exit-Policy definiert ist. Der Tor-Client hat nach dem Start kaum Flags vorzuweisen. Somit sind kurz nach dem Start nur die Flags *Running* sowie *V2Dir* vorhanden. Nach kurzer Laufzeit kommt dann das *Valid* Flag hinzu.
Für die Analyse der konkreten Anwendung sind primär die Flags *Stable*, *Fast* und *Exit* von Bedeutung. Das *Exit*-Flag ist aber grundsätzlich erst nach der Definition einer entsprechenden Exit-Policy erreichbar.
Das Vorhandensein beziehungsweise Zusammenwirken der entscheidenden Flags führt nach einer gewissen Laufzeit des Tor-Servers dazu, dass auch ein entsprechender Durchsatz an Daten über diesen Server stattfindet. Als Folge der zugewiesenen Flags durch die Algorithmen des Tor-Netzes sowie der zugrundeliegenden Infrastruktur weist der Exit-Node einen durchschnittlichen Durchsatz von 10 MB/s auf. Der Tor-Client läuft zu diesem Zeitpunkt bereits mehr als 140 Tage. Die statistische Entwicklung des Durchsatzes ist in Abbildung 9.1 des Anhangs nachvollziehbar. Die Grafiken entstammen dem Tor-Metric-Service Atlas[2]. Dabei ist zu beachten, dass zum abgebildeten Zeitpunkt keine Exit-Policy des Knotens bestand.

6.2.3.1 Exit-Policy

Die Exit-Policy definiert die IPv4 und IPv6 Adressen sowie Ports, die von einem aktiven Tor-Exit-Node genutzt werden können beziehungsweise freigeschaltet sind. Der Tor-Client wird zur Laufzeit über die entsprechende Stem-API[3] konfiguriert.
Im konkreten Analysefall wird der Standard-Port 80 freigegeben. Alle anderen Ports und Verbindungen sind nicht offen, sondern sie werden explizit vom Datenverkehr ausgeschlossen. Auf diesem Weg ist sichergestellt, dass ausschließlich Verbindungen über Port 80 in das offene Internet gelangen.
Für den reibungslosen Betrieb des Tor-Servers sind noch zusätzlich zwei Ports von Bedeutung. Zum einen ist dies der Port 9051, über den die Stem-Schnittstelle verfügbar ist. Von außen ist der Port aber nicht erreichbar, sondern ausschließlich über das Loopback-Interface. Weiterhin ist der Port 9030 für den Betrieb wichtig, da hierüber die eingehenden und ausgehenden Verbindungen zu den anderen Tor-Relay Servern aufgebaut werden.

[2] https://atlas.torproject.org/#details/1494ECFE6459C30A56C5096F17A4708B82E1DE41
[3] https://stem.torproject.org/

6.3 Analyse

In der Analyse wird der Kontext des Anwendungsszenarios genauer beschrieben und analysiert und die Faktoren, die von entscheidender Bedeutung für die Anwendung sind, werden hervorgehoben. Dabei wird auf das Zusammenwirken der Datengenerierung und Analyse eingegangen, die auf Basis der Plattform stattfinden.

6.3.1 Kommunikationsverfahren

In den Grundlagen wurde anfänglich das Verschlüsselungsverfahren von Tor-Servern untereinander im Kontext eines Overlay-Netzwerks erläutert. Es wurde ersichtlich, dass an keinem Knoten der Schaltung (circuit) der wirkliche Datenverkehr unverschlüsselt zugänglich ist, außer natürlich beim Anfragesteller (Initiator) und am Exit-Node.
Die Sichtbarkeit des Datenverkehrs am Exit-Node setzt aber voraus, dass die entsprechenden Netzwerk-Interfaces zugänglich sind beziehungsweise softwaretechnisch angesprochen und mitgeschnitten werden können.
Das grundsätzliche Verhalten eines Tor-Daemons, der über eine entsprechende Exit-Policy Datenverkehr unverschlüsselt ins Internet leitet, nutzt an der Stelle des Exit-Nodes das entsprechende Netzwerkinterface des Host-Systems, das eine Konnektivität mit dem Internet aufweist. Somit kann - wie bei jedem herkömmlichen Betriebssystem auch - der Datenverkehr an einem definierten Netzwerk-Interface mitgeschnitten werden. Die einzige Voraussetzung für das Mitschneiden des Datenverkehrs sind entsprechende Benutzerverwaltungsrechte beziehungsweise Root-Rechte auf dem zu analysierenden Betriebssystem.

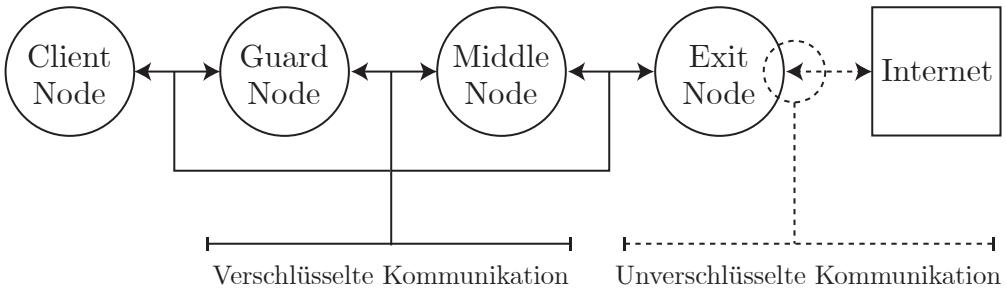

Abbildung 6.1: Tor-Circuit

6.3.1.1 Granularität der Daten

Die Analyse von Domain-Hosts beziehungsweise die Summierung der häufigsten Domains in einer Übersicht setzt voraus, dass die notwendigen Daten vorab bekannt sind. Somit könnte eine vorgeschaltete Selektion der Daten bereits auf der Ebene des Netzwerk-Interfaces durchgeführt werden. Dies widerspricht aber der Grundausrichtung der Plattform, in der eine Verarbeitung (hier: die Selektion) durch Daten Prozessoren vollzogen werden soll.
Das Analyseszenario ist in dem konkreten Fall der Anwendung klar definiert. Eine perspektivische Veränderung oder Verlagerung des Analyseziels ist aber bei gleichbleibender Datenbasis beziehungsweise Datenquelle ebenso vorstellbar. Daher ergibt sich im Rahmen der Analyse dieser Sachverhalte das Ziel die Datenquelle in der Art mit der Plattform zu konzipieren, dass auch bei Veränderung der Zielsetzung die Datenquelle gleich bleiben kann.

6.3.2 Datenintegrität

In den Grundlagen wurde anfänglich das Verhalten des Tor-Daemons beschrieben. Dabei wurde verdeutlicht, dass jeder Tor-Relay-Node (sowie in einer konkreten Form auch ein Tor-Exit-Node) etwaige Ports zur Kommunikation mit anderen Tor-Relay-Nodes benötigt. Ein Beispiel dafür ist das Zustandekommen von Circuits, die durch die jeweiligen Tor-Daemons untereinander verwaltet und initiiert werden müssen. Die Standard Ports dafür lassen sich in der entsprechenden Konfiguration vor dem Start des Daemons oder auch zur Laufzeit setzen. Trotzdem findet auch Datenverkehr auf den jeweiligen Ports 80 sowie 443 statt. Es handelt es sich bei diesem Datenverkehr nicht um Verkehr, der durch eine Exit-Policy spezifiziert wurde. Vielmehr geht es um Datenverkehr, der zwischen Tor-Relay-Nodes zur Verwaltung dieser stattfindet.
Es ergibt sich daraus, dass beim Mitschneiden eines entsprechenden Interfaces bei vorhandener Exit-Policy nicht nur Traffic aus den jeweiligen Circuits über das Interface mitgeschnitten wird. Dieser Traffic ist in der Zielsetzung der Anwendung aber nicht angestrebt, da er Datenstreams „verwässert“ beziehungsweise verfälscht.
Diese Erkenntnis ist ein Bestandteil der Entwicklung der Daten Prozessoren, die in der Konzeption zu berücksichtigen sind.

6.3.3 Anforderungen

In diesem Abschnitt werden die Anforderungen der konkreten Anwendung (6) spezifiziert. Die Anforderungen werden herausgearbeitet und nach funktionalen, technischen und nichtfunktionalen Eigenschaften untergliedert. Es finden dieselben Bezeichner wie in Abbildung

3.5 der Analyse (3) Verwendung. Das Akronym „TEI“ bezeichnet die Anwendung „Tor-Exit-Interceptor“.

6.3.3.1 Funktionale Anforderungen

TEI-FA-1
Die Daten werden am Interface des Betriebssystems mitgeschnitten.

TEI-FA-2
Datenstreams leiten die Daten in die Plattform.

TEI-FA-3
Daten werden um Geolokationsinformationen erweitert.

TEI-FA-4
Die verarbeiteten Daten werden durch entsprechende Möglichkeiten der Visualisierung dargestellt.

6.3.3.2 Technische Anforderungen

TEI-TA-1
Trotz steigendem Datenaufkommen leidet die Verarbeitung nicht durch die zugrundeliegende Struktur.

TEI-TA-2
Das Abrufen der Geolokationsinformationen ist von der Plattform entkoppelt.

6.3.3.3 Nicht-funktionale Anforderungen

TEI-NFA-1
Innerhalb von fünf Sekunden sollten Verarbeitungsergebnisse erkennbar sein.

TEI-NFA-2
Strukturinformationen über die Häufigkeit der Aufrufe entsprechender Hosts werden dargestellt.

6.4 Konzeption

Die Konzeption der Anwendung wird in diesem Abschnitt dargelegt. Dabei werden die zentralen Aspekte der Anwendung in der Art hervorgehoben, dass neben dem gesteckten Ziel auch

die vorhandenen und eingesetzten Techniken im Ansatz verdeutlicht werden.

6.4.1 Zielsetzung

Der Betrieb eines Tor-Exit-Nodes ermöglicht das Mitschneiden des Netzwerkdatenverkehrs am Knoten selbst. Dabei sind primär der Datenverkehr beziehungsweise die Datenpakete von Interesse, die den Traffic des HTTP-Protokolls betreffen. Es gilt zu untersuchen, welche HTTP-Host-Server am häufigsten frequentiert werden. Das heißt, es soll eine Übersicht darüber entstehen, welche Server am häufigsten aufgerufen werden. Aufgrund der Anonymisierung des Tor-Netzes ist es natürlich nicht möglich, direkt Rückschlüsse auf die Anfragesteller zu ziehen. Es lassen sich aber indirekt Informationen darüber erlangen, aus welchen Ländern oder Kontinenten vermutlich Anfragen gestellt werden.
Ziel der Anwendung ist es, eine visuell aufbereitete Darstellung der beschriebenen Sachverhalte sowie weitere Informationen über das entsprechende Dashboard dargestellt zu bekommen. Dazu zählt auch - da Rückschlüsse auf etwaige Geolokationen von HTTP-Servern gezogen werden können - eine Möglichkeit der geolokationsbasierten Visualisierung des Datenverkehrs zu schaffen.

6.4.2 Verarbeitungsketten

Die Datengenerierung geschieht auf der Basis des Mitschnitts von Netzwerkdaten an einem entsprechenden Interface. Die so aufkommenden Daten sind dadurch relativ grobkörnig. Das bedeutet, dass im Prozess der Verarbeitung die groben Daten in einer stufenweisen Transformation in feingranulare Daten überführt werden. In diesem Prozess produziert jeder Verarbeitungsschritt Artefakte, die je nach Untersuchungskontext weiter analysiert oder als Ergebnis gesehen werden können. Das heißt in der Konzeption, dass die Verarbeitungsstufen in sich so zu konzipieren sind, dass nicht ausschließlich die zuvor beschriebene Zielsetzung am Ende der gesamten Transformation stehen muss, sondern sich weitere Möglichkeiten der Analyse auf Basis von zwischenzeitlichen Artefakten eröffnen lassen.

6.4.2.1 Data Enrichment

Die Anreicherung von Daten um weitere Informationen ist ein passendes Beispiel für die Notwendigkeit einer weiteren Anwendung im Ökosystem der Plattform. Dabei ist die Aufgabe der Anwendung nicht originär die Transformation von Daten. Damit sind die Anwendungen gemeint, die im Kapitel der Konzeption der Systemarchitektur (4.2.1) zuvor erwähnt wurden.

Im konkreten Fall dieser Anwendung bedarf es der Notwendigkeit eines Mappings von IP-Adressen zu Geolokationen. Diese Notwendigkeit wurde zuvor in der Zielsetzung (6.4.1) sowie in den Verarbeitungsketten (6.4.2) erwähnt. Eine grundsätzliche Möglichkeit dieses Vorhabens ist auf Basis von IP-To-Geolocation-Datenbanken möglich. Das Unternehmen MAXMIND[4] bietet dafür eine freie Open-Source-Datenbank an.
Die Entwicklung einer Anwendung auf Basis der Datenbank ist für die Zielsetzung der Analyseanwendung notwendig. Dabei läuft die Anwendung - sowie alle anderen Komponenten des Systems auch - in einer isolierten Docker-Umgebung. Somit ist es möglich die entsprechenden Datenprozessoren, die die reine Verarbeitung der aufkommenden Netzwerkdaten übernehmen, an entsprechender Stelle auf die API der Geolokationsanwendung zugreifen zu lassen.

Destination-Mapping Für die Plattform muss schließlich ein Mapping in der Art konzipiert werden, dass eine Übersicht über die getätigten Requests am Tor-Exit-Knoten erstellt werden kann. Von Interesse ist grundsätzlich ein konkreter Host ohne etwaige lokale Namensauflösung. Das bedeutet in der Konzeption der Anwendung, dass jegliche Anfragen an einen Host summiert werden. Alle Sublevel-Domains werden dabei extrahiert.

Host-Domain : Count, Geolocation, [...]

Abbildung 6.2: Host-Domain-Mapping

Dieses Mapping wird erst am Ende der Verarbeitungskette auf Basis der feingranularen Daten vollzogen. Der restliche Datenverkehr zwischen zwei Tor-Relay-Servern fließt dabei nicht mit in das Ergebnis ein.

6.5 Anwendung

Dieser Abschnitt beschreibt einige Teilbereiche der konkreten Entwicklung der Anwendung zur Analyse des Netzwerkdatenverkehrs eines Tor-Exit-Nodes. Die drei zentralen Bereiche der Datengenerierung (Producer), Datenverarbeitung (Processor) und der Datenvisualisierung (Visualization beziehungsweise Widgets) werden beschrieben. Der Schwerpunkt liegt auf der Beschreibung der Bereiche, die in der Zielsetzung der Konzeption hervorgehoben wurden.

[4] https://www.maxmind.com/de/home

6.5.1 Producer

Die Daten für die Analyse der Anwendung werden am Standard-Interface des Betriebssystems gewonnen, auf dem der Tor-Daemon mit entsprechender Exit-Policy betrieben wird. Es findet an der Stelle keine Vorselektion der Datenpakete in der Art statt, dass über entsprechende Filter Datenverkehr ausgeschlossen wird. Zum Einsatz des Mitschnitts kommt in dieser Anwendung das Command-Line-Tool *tshark*. Hier handelt es sich, wie im Kapitel Grundlagen dieser Arbeit bereits erwähnt, um einen Network-Traffic-Sniffer. Die Producer-Komponente der Plattform zum Einleiten der Daten in das System bietet neben dem WS-Interface auch eine StdIn-Interface an (siehe 4.2.3.3). Auf dieses Interface baut diese Anwendung und nutzt somit eine Plattform-API zum Einleiten von Daten.
Die Konnektion des Producers an das Netzwerk-Interface auf dem Betriebssystem des Tor-Exit-Nodes ist die Datenquelle, über die die Daten der Plattform zugänglich gemacht werden.

6.5.2 Processor

Die Verarbeitung und Analyse der aufkommenden Daten ist gegenüber der ersten Anwendung (Request Intersection) komplexer. Die Komplexität entsteht durch die Notwendigkeit einer mehrstufigen Verarbeitungskette, die zuvor in Abschnitt 6.4.2 beschrieben wurde. Hinzu kommt außerdem noch die Anreicherung der aufkommenden Daten mit Geolokalisations-Informationen.
Konkret gliedert sich die Verarbeitung in fünf verschiedene Topics und vier verschiedene Daten Prozessoren. Im Folgenden werden diese verschiedenen Stufen eingehender beschrieben.
In Abbildung 6.3 ist die Verarbeitungskette der jeweiligen Prozessoren ersichtlich.

6.5.2.1 Datenaufkommen

Am Anfang der Verarbeitung steht die Herausforderung die mitgeschnittenen Daten zu bereinigen.
Jedes Tor-Relay, egal welcher Ausprägung (Exit, Middle, Guard), ist öffentlich bekannt. Das bedeutet, es lässt sich zur Laufzeit der Anwendung abrufen, ob ein bestimmter Knoten momentan ein Tor-Relay-Knoten ist. Daraus folgt, dass es öffentliche Listen bekannter Tor-Relay-Knoten gibt, die verlässlich sind. Auf dieser Basis funktionieren unter anderem auch die Tor-Metric Services[5] des Tor-Projekts.
Die Datenbereinigung der ersten Stufe ist somit ein reines Abgleichen des eingehenden Datenverkehrs mit einer zur Laufzeit akquirierten Liste bekannter Tor-Relay-Nodes. Als Resultat

[5] https://metrics.torproject.org/

steht am Ende des entsprechenden Datenprozessors ein bereinigter Tor-Exit-Datenstrom des entsprechenden Netzwerk-Interfaces.

6.5.2.2 Datenerweiterung

Das Erweitern der Daten geschieht mit Hilfe der Anwendung zur Bestimmung der Geolokalisation von IP-Adressen. Diese Grundfunktionalität ist nicht Bestandteil des eigentlichen Datenprozessors, da dieser der Transformation der eingehenden Daten begegnen soll. Das heißt in der konkreten Anwendung, dass der Prozessor als eingehenden Datenstrom die bereinigten Daten des Datenaufkommens nutzt. Im Mapping-Prozess der Daten wird dann die API der Geodaten-Anwendung genutzt um die Zieladresse im Datensatz des Prozessors abzufragen. Das Ergebnis wird dann dem Basisdatensatz hinzugefügt und erweitert diesen somit.
Die erweiterten Daten werden dann in ein neues Topic geschrieben, das wiederum von den folgenden Prozessoren verarbeitet werden kann. Notwendig ist das allerdings nur bei den Prozessoren, die zur Analyse auf die Geodaten bauen. In den anderen Fällen kann auch der Stream der bereinigten Netzwerkdaten herangezogen werden. In Abbildung 6.3 ist dies durch eine gestrichelte Linie symbolisiert.

6.5.2.3 Datenanalyse

Die Analyse der zuvor geschaffenen Datenströme erfolgt durch zwei unterschiedliche Daten Prozessoren. Zwar haben beide verschiedene Aufgaben, doch sie ähneln sich in ihrem technischen Verhalten. Die beiden Kollektoren dienen der Generierung der Übersichtsstatistiken in der Ergebnisdarstellung der Widgets.

Traffic-Collector Der Traffic-Collector basiert auf dem Windowing-Prinzip von Apache Kafka. Dabei generiert er innerhalb eines definierten Zeitfensters die auftretenden Daten in der Art, dass am Ende eine Statistik über die Ströme der Exit-Node-Daten entsteht. Dabei wird kein Mapping in Bezug auf HTTP-Requests vorgenommen, sondern es werden nur die Kommunikationsverbindungen summiert. Bei der Verwendung des erweiterten Datenstroms mit Geodaten lassen sich diese Daten in einer Weltkarte visualisieren. Im Fall der Konnektierung am bereinigten Netzwerkdatenverkehr entsteht eine minimalistische Statistik der Häufigkeit der Datenpakete pro IP-Adresse.

Host-Collector Der Host-Collector ist gegenüber dem Traffic-Collector in der Art verfeinert, dass eine Einschränkung auf das HTTP-Protokoll vorgenommen wird. Hier filtert der Prozessor

jeglichen anderen Datenverkehr aus dem definierten Input-Stream, der nicht dem HTTP-Protokoll zuzuordnen ist. Im Ergebnis lassen sich die nicht namensaufgelösten Daten so zusammenführen, dass ein Mapping, wie in Abbildung 6.2 beschrieben, ermöglicht wird. Technisch gesehen basiert der Host-Collector auch auf dem Windowing-Prinzip so wie der Traffic-Collector, wohingegen die Analysen verschiedene Ergebnisse erbringen.

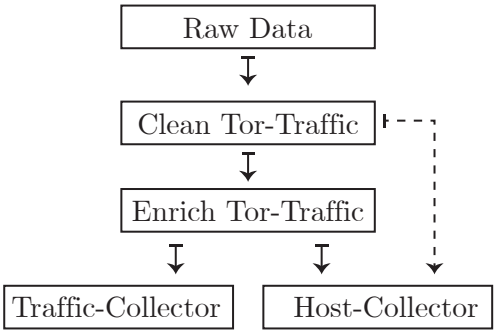

Abbildung 6.3: Tor-Exit-Processor

6.5.3 Visualization

Das Abrufen der entsprechenden Daten zur Visualisierung im Dashboard erfolgt über die beiden Streams des Traffic- und Host-Collectors. Da diese am Ende der konzipierten Verarbeitungskette stehen, sind sie über die Bridge-API im Front-End zugänglich.

6.6 Ergebnisse

In diesem Abschnitt werden die Ergebnisse sowie Erkenntnisse der durchlaufenen Analyse zusammengefasst. Es werden punktuell die Bezüge zu den technischen Grundlagen der Daten Prozessoren bis hin zur Visualisierung hergestellt und beschrieben. Im Einzelnen stehen aber die konkreten Resultate der Analyse im Mittelpunkt.
Vorab sind hier ein paar Eckdaten zu nennen, die das Verhalten und den Sachverhalt des Exit-Nodes genauer beschreiben. Der Relay-Server wurde zum Beginn des Tests mit einer Exit-Policy für den Port 80 (HTTP) konfiguriert. Ein konkretes Beispiel könnte die Verwendung des TorBrowsers[6] für HTTP-Anfragen sein.
Analysiert wurden Daten in einem Zeitraum von ca. zwei Stunden, in dem die Exit-Policy

[6]https://www.torproject.org/projects/torbrowser.html

definiert war. Innerhalb der zwei Stunden wurden über 12,5 Millonen Pakete am Exit-Node gemessen und analysiert. Das Zeitfenster des Knotens war zwischen 13:00 Uhr und 15:00 Uhr eines Donnerstags im März 2017 geöffnet.

6.6.1 Traffic-Mapping

Das Widget „Tor-Traffic-Interceptor“ bietet eine rudimentäre Übersicht über den punktuellen Traffic am Tor-Exit-Node. Ein Screenshot über den visualisiert aufbereiteten Traffic am Exit-Node ist in Abbildung 6.4 ersichtlich. Das Widget wurde auf Basis der Google-Maps-API[7] implementiert. Durch das Widget ist es möglich, direkt die Geolokationen der Server zu erkennen, die innerhalb eines gesetzten Zeitfensters aufgerufen wurden. Somit zeigt sich in Echtzeit, welche Server auf welchen Kontinenten, in welchen Ländern und partiell auch in welchen Städten vom Exit-Node aus aufgerufen werden.
Diese Übersicht ist insofern rudimentär, dass keine konkreten Inhalte in der Verarbeitung vorausgegangen sind, sondern vielmehr ein Eindruck darüber entstehen soll, wohin der aktuelle Traffic im Allgemeinen gerichtet ist.
Im Ergebnis zeigt sich, dass innerhalb der zwei Stunden, in denen der Server als Exit-Node fungierte, der überwiegende Traffic an Server gegangen ist, die der IP-Übersetzung nach entweder in Europa oder Nordamerika verortet sind. Aus dieser ersten Ansicht lässt sich keine prozentuale Angabe machen, da keine statistische Erhebung gemacht wurde, sondern nur visuell animiert wurde, wo gerade Datenverkehr stattfand.

6.6.2 Count-Exit-Hosts

Im zweiten Teilbereich der Analyse des Datenverkehrs wurden am Exit-Node getätigte HTTP-Requests untersucht. Wie in der Konzeption beschrieben wurden Erhebungen darüber gemacht, welche HTTP-Hosts am häufigsten frequentiert wurden. Außerdem wurde auch akkumuliert, in welchen Kontinenten, Ländern und Städten die angefragten HTTP-Server stehen.
Untergliedert werden die Ergebnisse in drei Abschnitte. Zuerst findet eine Beschreibung der Resultate innerhalb der ersten Stunde statt. Im Anschluss daran wird dann die zweite Stunde jeweils in zwei halbe Stunden untergliedert.
Die Analyse der ersten Stunde zeigt das Ergebnis in Tabelle 6.2. Dabei sind nur die zehn am häufigsten frequentierten Seiten abgebildet. Die Spalten City, Country und Continent beziehen sich dabei auf die Analyse der Ziel-Adresse. Die Spalte Requests beschreibt die Anzahl der gemessenen HTTP-Anfragen an den jeweiligen Host, wobei durch das Mapping (siehe 6.4.2.1)

[7] https://developers.google.com/maps/

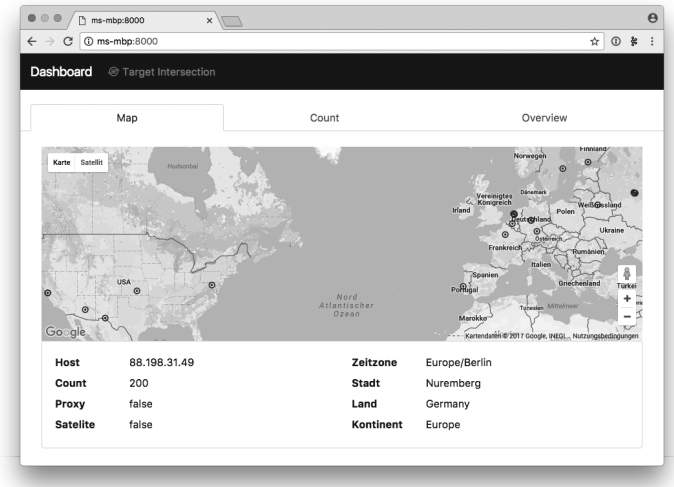

Abbildung 6.4: Tor-Exit Traffic-Map

alle Sub-Level-Domains insgesamt summiert wurden.
Im Ergebnis zeigt sich, dass der Host *doubleclick.net* am häufigsten frequentiert wurde, der sich in Kalifornien/USA verorten. Im vorherigen Test der „Request-Intersection“ zeigte sich auch, dass *doubleclick.net* auch dort in den analysierten Anfragen vorkam. Bei *doubleclick.net* handelt es sich um ein Unternehmen von Google, das seinen Kunden ermöglicht, Werbekampagnen zielgruppengerecht zu steuen.
Bei dem Host *steamcommunity.com* handelt es sich um eine Online-Vertriebsplattform für Computerspiele. Es zeigte sich, dass Steam verschiedene API's anbietet, anhand derer man Daten etwaiger Produkte ihres Sortiments abrufen kann.
An dritter Stelle steht das Portal der Europäischen Union, das unter *europa.eu* erreichbar ist. Hier handelt es sich um die offizielle Webpräsenz der EU, die ersten Erkenntnissen nach reinen Informationscharakter hat und keine konkreten Webservices anbietet.
In den weiteren sieben am häufigsten aufgerufenen Seiten finden sich neben *google.com* auch Portale mit pornographischen Inhalten. Dazu zählt unter anderem *xvideos.com*. Bei den Hosts *backpage.com* und *craigslist.org* handelt es sich um Anzeigenwebseiten, auf denen Kunden in verschiedenen Rubriken ihre Anzeigen schalten können. Das umfasst auch Rubriken wie Arbeitsstellen oder Partnersuche. Zu der Kategorie des Online-Marketings können die Hosts *vertamedia.com* sowie *vagex.com* gezählt werden, wobei letzterer (*vagex.com*) noch eingehender im Fazit erwähnt wird.

#	Host	City	Country	Continent	Count
1	doubleclick.net	Mountain View	United States	North America	7358
2	steamcommunity.com	–	–	Europe	6993
3	europa.eu	–	Luxembourg	Europe	2602
4	xvideos.com	–	Netherlands	Europe	1267
5	backpage.com	–	United States	North America	708
6	vagex.com	Provo	United States	North America	570
7	google.com	Mountain View	United States	North America	462
8	vertamedia.com	Piscataway	United States	North America	434
9	craigslist.org	San Francisco	United States	North America	429
10	list-manage.com	Amsterdam	Netherlands	Europe	402

Tabelle 6.2: Count-Exit-Hosts: Erste Stunde

In der ersten halben Stunde der zweiten Analysestunde sind die analysierten Ergebnisse in Tabelle 6.3 abgebildet. Dabei zeigte sich, dass sich die Tendenz aus der ersten Stunde fortsetzte. Es stiegen die Anfragen an *steamcommunity.com* jedoch stark an. Hinzugekommen ist außerdem eine weitere Seite mit pornographischen Inhalten *fap.to*, die in ihrer Häufigkeit der Requests die Seite *xvideos.com* abgelöst hat. Bei der IP-Adresse auf Rang zehn handelt es sich ebenfalls um Pornographie, wobei sich durch eine Reverse-DNS-Anfrage kein Eintrag finden lässt. Neben *google.com* ist in der untersuchten halben Stunde außerdem auch *bing.com* neu hinzugekommen.

In den Anfragen der zweiten hälfte der zweiten Stunde zeigt sich ein ähnliches Bild wie im vorherigen Zeitraum. Die Ergebnisse sind in Abbildung 6.4 ersichtlich. Dabei zeigen sich in diesem Zeitraum wieder neue Hosts unter den meist frequentierten Hosts. Die ersten Positionen sind aber über den gesamten Zeitraum der zwei Stunden relativ konstant.

6.6.3 Zusammenfassung

Die Unterteilung in drei unterschiedliche Zeitfenser hatte primär das Ziel immer nur ein punktuelles Aufkommen von Anfragen abzubilden. Für diesen Zweck wurde das erste Fenster größer gewählt, da anfänglich weniger Datenverkehr über den Exit-Node geleitet wurde. Die Konfiguration war nämlich erst nach einer gewissen Zeit im Tor-Netz bekannt.

#	Host	City	Country	Continent	Count
1	steamcommunity.com	Amsterdam	Netherlands	Europe	23220
2	doubleclick.net	Mountain View	United States	North America	7669
3	europa.eu	–	Luxembourg	Europe	4480
4	fap.to	Prague	Czechia	Europe	1384
5	bing.com	Redmond	United States	North America	1375
6	list-manage.com	Amsterdam	Netherlands	Europe	1136
7	amazon.de	Dublin	Ireland	Europe	1035
8	google.com	Mountain View	United States	North America	930
9	com.br	–	Brazil	South America	858
10	78.140.156.115	–	Netherlands	Europe	851

Tabelle 6.3: Count-Exit-Hosts: Erste halbe Stunde der zweiten Stunde

6.7 Fazit

Als Fazit der untersuchten HTTP-Anfragen am Tor-Exit-Node lassen sich verschiedene Ergebnisse konstatieren. Dazu zählt zunächst, dass unter den ersten zehn Plätzen relativ wenige Seiten mit pornographischen Inhalten vertreten sind. Bevor sich die beschriebenen Ergebnisse abzeichneten, war die Erwartung die, dass wohl zumindest ein Drittel der ersten zehn Plätze sich dieser Kategorie zuordnen ließen. Das Ergebnis zeigt aber, dass lediglich ein bis zwei derartige Seiten unter den ersten zehn Plätzen vertreten sind. Die Feststellung bezieht sich aber nur auf die ersten zehn Plätze und nicht auf die Gesamtheit aller angefragten Hosts.
Im Fall der relativ vielen HTTP-Anfragen an die Seite der Europäischen Union (*europa.eu*) lässt sich bei punktueller manueller Nachbetrachtung die folgende These aufstellen: Die Anfragen über den Exit-Node an *europa.eu* könnten Teil einer gesteuerten DDoS-Attacke gewesen sein. Bei einer manuellen Betrachtung der getätigten Anfragen im Einzelnen zeigt sich, dass häufig dieselbe Ressource angefragt wurde, sodass die Anfragen insgesamt dem Muster von DDoS-Attacken entsprächen.
Im Ergebnis der zweiten Stunde zeigt sich außerdem ein Fehler in der Konzeption des Mapping beziehungsweise der Implementation des Host-Mappings. So taucht *com.br* in den Ergebnissen auf Platz neun der ersten halben Stunde auf. Das Mapping hat an dieser Stelle außer Acht gelassen, dass Second-Level-Domains wie *co.uk* oder *com.br* existent sind.
Im Fall von *vagex.com* zeigt sich nach der Recherche des dahinterliegenden Unternehmens,

#	Host	City	Country	Continent	Count
1	steamcommunity.com	Amsterdam	Netherlands	Europe	11935
2	doubleclick.net	Mountain View	United States	North America	6505
3	amazon.de	Dublin	Ireland	Europe	4131
4	dalilyapp.com	Dublin	Ireland	Europe	3244
5	vk.com	–	Russia	Europe	2922
6	ebay.de	–	Netherlands	Europe	2914
7	amazon.es	Dublin	Ireland	Europe	2902
8	ameba.jp	Amsterdam	Netherlands	Europe	2803
9	66.135.211.96	Campbell	United States	North America	2388
10	xvideos.com	Phoenix	United States	North America	2002

Tabelle 6.4: Count-Exit-Hosts: Zweite Hälfte der zweiten Stunde

dass es sich dabei um eine Dienstleistung von „Klick-Kauf“ für YouTube-Videos handelt. Das heißt in konkreten Fällen, dass durch höhere Klickzahlen auf Videos entsprechend auch mehr Werbung geschaltet wird. Dadurch erhält der Eigentümer der Videos eine entsprechende Vergütung. Durch dieses Prinzip lässt sich somit Umsatz generieren.
Im Fall der Anfragen von *steamcommunity.com* scheint es sich um das Auslesen der vorhandenen Daten über die Steam-API zu handeln. Dabei ist die Absicht hinter diesem Vorgang nicht direkt ersichtlich, wobei das Schema bekannt ist. Es existieren immer wieder Fälle, in denen versucht wird an die Datenbestände von Unternehmen zu kommen. Im Fall von Steam handelt es sich um den wahrscheinlich größten Anbieter von Computerspielen, die sich über deren Plattform beziehen lassen.

7 Bewertung

In diesem Kapitel findet eine Bewertung der entwickelten Plattform statt. Dabei werden - mit dem Anspruch einer kritischen Auseinandersetzung - die getroffenen Entscheidungen sowie mögliche weitere Entwicklungsbereiche der Plattform behandelt.

Die Ausrichtung der entwickelten Plattform ist die Verarbeitung und Analyse von Netzwerkdaten. Hierbei galt es zu gewährleisten, dass der Anspruch sowie die Prämisse der Echtzeitfähigkeit der Plattform am Ende gegeben waren. In Hinblick auf die Architektur der Plattform zeigt sich die Komplexität dieses Anspruchs. Das heißt, es sind nicht originär entwicklungstechnische Faktoren für das Gelingen ausschlaggebend. Die zugrundeliegende Infrastruktur ist in nicht unerheblichem Maße mit dafür verantwortlich, dass das Ziel der Echtzeitfähigkeit gewährleistet wird. Dabei spielt sowohl die eingesetzte Hardware der Server eine Rolle, auf denen Teilkomponenten zur Ausführung gebracht werden, als auch die allgemeine Netzinfrastruktur der beteiligten Server der Plattform. Etwaige Probleme, die sich auf eine gegebene Infrastruktur zurückführen lassen, waren aber nicht inhaltlicher Bestandteil dieser Arbeit.

Eine Auseinandersetzung mit bereits existierenden Ansätzen zum Thema dieser Arbeit schließt das Analysekapitel ab. Es wurden neben proprietären Ansätzen von Unternehmen wie Google und Amazon auch quelloffene Technologien - zumeist unter dem Dach der Apache Foundation - behandelt und ein Bezug zur eigenen Plattform hergestellt. Den zuvor beschriebenen Limitationen auf der Ebene der Infrastruktur kann man bei großen Unternehmen besser begegnen. Außerdem bieten die Unternehmen ihre Produkte zumeist als Cloud Service an. Das heißt, im Vergleich zur eigenen Plattform muss lediglich die entsprechende Business Logic über die Schnittstellen der Cloud-Services implementiert werden, ohne konkret auf den Betrieb der Services achten zu müssen. Ein direkter Vergleich der eigenen Plattform zu diesen Cloud-Diensten ist somit nicht herstellbar, da ein höherer Grad der Abstraktion der eigenen Plattform als Cloud-Service nicht Teil dieser Arbeit ist.

Der Einsatz von Docker - beziehungsweise die grundsätzliche Verwendung aller Teilkomponenten des Systems in isolierten Umgebungen als Container - ist die Voraussetzung für den zuvor

beschriebenen Schritt der Entwicklung der Plattform als Cloud Service. Da von vornherein das Ziel eines Verteilten Systems ein Grundzug der Entwicklung war, bot sich Docker sowohl in der Entwicklung als auch für den Betrieb an. Diese Art der Abstraktion der Teilkomponenten ist die Basis für den reibungslosen Betrieb in linuxieden Betriebssystemumgebungen. Der Einsatz im Produktionsbetrieb wurde in der Auseinandersetzung mit Tools wie Kubernetes in der Konzeption behandelt. Dabei ist an dieser Stelle trotzdem die konkrete Infrastruktur mitverantwortlich für das Erreichen der Echtzeitfähigkeit.

Die Konzeption sowie die beiden konkreten Anwendungen nutzen zwei Schnittstellen der Plattform zum Einleiten beziehungsweise Generieren von Datastreams. Die konzipierten Schnittstellen sind generisch gehalten, sodass bei anderen Möglichkeiten des Mitschneidens von Netzwerkdatenverkehr diese Schnittstellen problemlos genutzt werden können. Ein anderer Weg an Stelle der Data Producer wäre aber auch denkbar. Das wäre zum Beispiel in der Art möglich, dass nicht die Generik der Schnittstelle im Vordergrund steht, sondern indem man spezielle Schnittstellen implementiert, die sich auf einer tieferen Ebene mit dem Betriebssystem konnektieren lassen. Hier wäre aus Sicht der Performance beziehungsweise des Durchsatzes an Daten gegenüber allgemeinen Schnittstellen noch weiteres Potential vorhanden.

In Tabelle 7.1 wird ersichtlich, welche Anforderungen der Anwendung „Request-Intersection“ durch die Plattform oder eine weitere Implementation von Komponenten gewährleistet wird. Bei der Betrachtung der gestellten Anforderungen an die Anwendung wird erkennbar, dass unter anderem entsprechende Anforderungen der Plattform die Basis für die Erfüllung der Anwendungsanforderung darstellen. Das ist bei den Anforderungen zum Verarbeiten von Daten (RI-FA-3) sowie bei der Visualisierung der Fall (RI-FA-4). In beiden Fällen bietet die Plattform die entsprechende Grundlage. Zum einen verarbeiten implementierte Prozessoren die Daten für das entsprechende Host-Mapping der Anwendung. Zum anderen arbeiten diese Prozessoren aber nur auf der Basis der Plattform beziehungsweise sie werden auf dieser betrieben. Für die Entwicklung der Anwendung „Request Intersection“ wird in der Tabelle aber auch ersichtlich, dass für die Anforderungen RI-FA-2 sowie RI-NFA-1 ausschließlich Basisanforderungen der Plattform verantwortlich sind. Das heißt, dass für das Einleiten der Daten (RI-FA-2) keine Implementationsarbeit auf Seiten der Anwendung geleistet werden muss. Die Grundfunktionalität stellt in diesem Fall die Plattform zur Verfügung. Auch im Fall der nicht-funktionalen Anforderung RI-NFA-1 stellt die Plattform die Grundvoraussetzung dar. Die Geschwindigkeit der Verfügbarkeit von Verarbeitungsergebnissen wird durch die Plattform gewährleistet und ihr muss bei der Entwicklung der Anwendung nicht begegnet werden.
Es zeigt sich bei dem Abgleich der Anforderungen aber auch, dass die reine Business Logic

nur durch eine Entwicklung auf der Seite der Anwendung gewährleistet werden kann. Die Plattform kann und soll diese konkreten Probleme nicht lösen. Gemeint ist zum Beispiel die Anforderung RI-FA-1. Diese definiert die Browser-API, die als Datenquelle verwendet wird. Außerdem ist das Ziel die Verarbeitungsergebnisse nur durch die Anwendung selbst und nicht durch die Plattform (RI-NFA-2) zu realisieren.

Bezeichner	**Plattform**	**Anwendung**
RI-FA-1	–	✓
RI-FA-2	✓P-FA-2	–
RI-FA-3	✓P-FA-3	✓
RI-FA-4	✓P-FA-5	✓
RI-TA-1	✓P-TA-2	–
RI-NFA-1	✓P-NFA-1	–
RI-NFA-2	–	✓

Tabelle 7.1: Anforderungsabgleich „Request-Intersection“

Die Tabelle 7.2 stellt den Anforderungsabgleich der Anwendung „Tor-Exit-Interceptor“ dar. Dabei zeigt sich im Allgemeinen ein ähnliches Bild wie bei der Anwendung „Request-Intersection“. Das Akquirieren der Daten wird durch die implementierte Anwendung abgedeckt (TEI-FA-1), da die Datenquelle zumeist in konkreten Anwendungen variiert. Hingegen wird aber auch ersichtlich, dass in dieser Anwendung die Plattform beziehungsweise deren entsprechende Komponente (Producer) für die Bereitstellung der Daten genutzt wird (TEI-FA-2). Im Fall der Anreicherung der Quelldaten in dem entsprechenden Datenstream kommt bei der Anwendung Tor-Exit-Interceptor hingegen eine plattformfremde Komponente zum Einsatz (TEI-FA-3). Da diese Anforderung aus der Notwendigkeit des Analyseszenarios zur Erweiterung der Daten um Geolokationen entstanden ist, kann sie auch nur durch die Implementation auf der Seite der Anwendung gewährleistet werden.
Insgesamt zeigt sich in der Anwendung Tor-Exit-Interceptor, dass Anforderungen der konkreten Business Logic auch hier sowohl auf der Seite der Anwendung als auch auf der Seite der Plattform angesiedelt sind. Das betrifft zum Beispiel die Anforderung TEI-FA-4 zur Visualisierung der am häufigsten aufgerufenen Hosts.

Die bei den Producern vorhandene Generik ist bei den Prozessoren hingegen nicht von Nachteil. Die verschiedenen Möglichkeiten der Implementation von Producern auf high- und low-level

Bezeichner	Plattform	Anwendung
TEI-FA-1	–	✓
TEI-FA-2	✓P-FA-2	–
TEI-FA-3	–	✓
TEI-FA-4	✓P-FA-5	–
TEI-TA-1	✓P-TA-2	–
TEI-TA-2	–	✓
TEI-NFA-1	✓P-NFA-1	–
TEI-NFA-2	–	✓

Tabelle 7.2: Anforderungsabgleich „Tor-Exit-Interceptor“

API's ist grundsätzlich von Vorteil. Es lassen sich Transformationen oder Manipulationen zumeist einfach und schnell implementieren, sofern das zu erreichende Ziel nicht von großer Komplexität ist. Im Beispiel der Anwendung Tor-Exit-Interceptor galt es, vorab eine einfache Filterung der Daten des Tor-Exit-Streams vorzunehmen. Das Resultat war schließlich ein Stream von reinem Tor-Exit-Traffic, der keine Spuren von Kommunikation zwischen Tor-Relay-Knoten beinhaltet. Für diese Aufgabe konnte durch die vielen Möglichkeiten der Formen der Implementation die leichtgewichtigste gewählt werden. Zuträglich ist aber auch das Paradigma des Publish/Subscribe der Topics von Kafka.

Die Möglichkeit der Strukturierung von Anwendungen nach dem Muster von aufeinanderfolgenden Schritten der Verarbeitung ist als positiver Faktor hervorzuheben. Es besteht keine technische Kopplung von konkreten Anwendungen der Plattform. Vielmehr existiert nur eine logische Kopplung von Prozessoren in der Art, dass etwaiges Wissen über den Gehalt von Daten vorhanden ist, sodass Prozessoren miteinander über die Topics verkettet werden können. Das heißt konkret, dass beim Prozess der Erstellung von Verarbeitungsketten an jedem Bindeglied neue Prozessoren hinzugefügt werden können ohne die Verarbeitungskette dabei zuvor öffnen zu müssen.

Im Allgemeinen lässt sich konstatieren, dass die gewählten Teilkomponenten, Systeme und Anwendungen in ihrer Gesamtheit den Ansprüchen der Aufgabe gewachsen sind. Punktuell lassen sich durch die Strukturierung der Plattform an verschiedenen Stellen Komponenten noch erweitern oder optimieren. Die Fähigkeit der Echtzeit hat sich in den konkreten Anwendungen an den Beispielen des Front-Ends gezeigt. Dabei wurden die erwarteten und verarbeiteten

Ergebnisse durch die implementierten Widgets adäquat visualisert. Bezogen auf die Echtzeitfähigkeit war aber weniger die Qualität der Darstellung von Bedeutung als vielmehr die Unmittelbarkeit, in der die Verarbeitungsergebnisse der Plattform verfügbar waren. Die visuelle Aufbereitung war schlussendlich mehr eine Frage des Verständnisses der Analyseergebnisse als eine Frage der technischen Machbarkeit.

8 Zusammenfassung

In dieser Zusammenfassung werden die zentralen Kapitel sowie deren Teilbereiche inhaltlich zusammengefasst. Dabei erfolgt ein Überblick über die behandelten Themengebiete von den Grundlagen bis hin zu den konkreten Anwendungen der letzten beiden Kapitel.

Zu Beginn dieser Arbeit wird die Motivation beschrieben, die dazu geführt hat, sich dem Thema dieser Arbeit zu widmen. Hierbei steht im Vordergrund, unmittelbare Ergebnisse aus Verarbeitungsprozessen zu gewinnen, die wiederum zu strategischen Entscheidungen führen können.
In der Zielsetzung wurde dieses Vorhaben in einer konkreten Form definiert, sodass daraus der große Rahmen des Arbeitsthemas ersichtlich wird.

In den Grundlagen (2) dieser Ausarbeitung werden die thematisch zentralen Bereiche behandelt. Im Allgemeinen steht hierbei im Vordergrund, die technischen sowie ökonomischen Sichtweisen zu nennen und zu beschreiben, die im Kontext von Big Data von Bedeutung sind. Beschrieben wird in dem Kapitel Big Data neben allgemeingültigen Definitionen wie dem 3-V-Modell auch die genannte ökonomische Sicht des Wertes von Daten.
Der zentrale Charakter der entwickelten Plattform wird in dem Abschnitt der Verteilten Systeme genauer behandelt.
Das Event Processing steht neben der Betriebsart als Verteiltes System im Mittelpunkt. Dabei wird in dem gleichnamigen Abschnitt auf die Echtzeitfähigkeit genauer eingegangen.
Neben den unterschiedlichen Technologien werden im Abschnitt Plattform die Basiskomponenten beschrieben. Die Grundlagen finden im Abschnitt der Systemarchitektur ihren Abschluss.

In dem Analysekapitel (3) steht im Vordergrund, die Bereiche und Probleme der Entwicklung der Plattform zu benennen und zu untersuchen. Am Anfang wird der Kontext der Plattform genauer spezifiziert und über ein entsprechendes Anwendungsfalldiagramm visualisiert. Dabei werden die wichtigsten Akteure in ihrer Interaktion mit dem System dargestellt.
Vorab bedarf es aber einer genauen Definition der zentralen Begrifflichkeiten der Arbeit, die in dem Abschnitt Terminologie zum Ausdruck gebracht werden.

Stream Processing und Network Analytics werden nach ihrer ersten Erwähnung in den Grundlagen genauer beschrieben.
Die Abschnitte Anforderungsspezifikation und Domänenanalyse bilden danach die Vorbereitung auf die konkreten Anforderungen an die Plattform.
Die Anforderungen und Anwendungsfälle auf der Basis des Plattformkontextes bilden den Abschluss der Plattformanalyse. Das Kapitel Analyse endet mit der Betrachtung und Auseinandersetzung verschiedener anderer Systeme.

In dem Kapitel Konzeption (4) werden die Erkenntnisse und eine erste Spezifikation einzelner Teilbereiche der Plattform konkretisiert. Dabei strukturiert sich das Kapitel von einer abstrakten Betrachtung von Teilbereichen hin zu einer feingranularen Sicht einzelner Bestandteile.
Zu Beginn der Konzeption wird der gesamte Kontext der Plattform im Abschnitt Architektur aufgezeigt. Vorab gilt es, eine allgemeine Übersicht über die unterschiedlichen Ebenen der Plattform zu geben mit folgendem Ziel: Die Bereiche, auf die die Plattform baut, und die Bereiche, die sie schafft, sollen ersichtlich werden.
Der Abschnitt Plattform behandelt im Anschluss an die Architektur eine feingranulare Sicht auf das Konzept der Zusammensetzung der Plattformkomponenten. Neben der Systemübersicht der Plattform wird auch eine mögliche Instanzstruktur beschrieben.
Der Aufbau und die Struktur der jeweiligen Komponenten steht im Abschnitt Komponentenarchitektur im Vordergrund. Es werden sowohl die Schnittstellen untereinander beschrieben als auch die Schnittstellen einiger Komponenten nach außen dargestellt, auf deren Basis konkrete Anwendungen entwickelt werden.
Die Konzeption findet ihren Abschluss mit der Beschreibung und Behandlung der wichtigen Anwendungen und Systeme, auf denen die Plattform fußt. Dabei findet eine kontextbezogene Beschreibung der Anwendungen statt.

Die Kapitel 5 und 6 beschreiben jeweils die Entwicklung bis zu den Analyseergebnissen konkreter Anwendungen auf Basis der Plattform. Es werden die entwickelten Prozessoren sowie Widgets beschrieben mit einer abschließenden Bewertung der Verarbeitungsergebnisse.

Die Anwendung der „Request Intersection“ (5) hat zum Ziel, einen definierten Input-Daten-Stream in der Art zu untersuchen, dass am Ende eine Übersicht darüber entsteht, welche Webseiten zueinander jeweils auf dieselben externen Ressourcen zugreifen.
Die Analyse führt dazu, dass in der Konzeption ein Mapping von Primär- und Sekundärrequests vorgenommen werden kann. Der Abschnitt Anwendung beschreibt die konkrete Realisierung der Daten Prozessoren, Daten Producer und Widgets.

Den Abschluss bildet das Kapitel Request Intersection mit der Beschreibung und Evaluation der Analyseergebnisse in Hinblick auf Auswirkungen auf eine hohe Dichte gemeinsamer Ressourcen unterschiedlicher Webseiten.

Die Anwendung „Tor-Exit-Interceptor“ wird im Kapitel 6 behandelt. In den Grundlagen werden vorab die wichtigsten Gegebenheiten des Tor-Netzwerks beschrieben. Besondere Erwähnung finden Struktur und Verhalten eines Tor-Exit-Knotens.
Das Ziel der Anwendung besteht in der Analyse des ausgehenden Netzwerkdatenverkehrs eines Tor-Exit-Knotens. Im Abschnitt Anwendung wird die Verarbeitungskette der Datenprozessoren beschrieben, die notwendig ist um die Ergebnisse aus dem Datenverkehr extrahieren zu können. Zum Abschluss der Anwendungen werden die unterschiedlichen Ergebnisse präsentiert.

Das Kapitel Bewertung (7) beinhaltet eine kritische Auseinandersetzung mit dem Thema dieser Arbeit sowie der erfolgten Entwicklung der Plattform. Dabei steht im Vordergrund, die Bereiche zu nennen, in denen noch Potential zum Ausbau des Themas und der Entwicklung liegt. Gelegt wird der Fokus aber nicht zu sehr auf Fragen der Implementation im Einzelnen, sondern eher auf das Konzept im Allgemeinen.

9 Ausblick

In diesem Kapitel werden - abschließend mit dem Thema dieser Arbeit - einige Überlegungen in Hinblick auf weitere Möglichkeiten oder nicht ausgeschöpfte Potentiale angestellt.

Eine Idee für die Weiterentwicklung wäre - wie in der Bewertung schon angeklungen - die Entwicklung der Plattform als Cloud-Service. Hierbei stünde primär im Vordergrund, die Systembereiche zu abstrahieren, die nur eine Schnittstelle der Plattform nach außen hin anbieten, sonst aber nicht weiter vom Akteur Entwickler verwendet werden. Dieser Service würde dann - verschiedener anderer proprietärer Services entsprechend - angeboten werden können. In diesem ersten Schritt würde nicht der monetäre Aspekt, sondern die Umsetzung des Service als Blackbox im Vordergrund stehen.
Bei diesem Vorhaben entstünden neue Herausforderungen für die adäquate Nutzung von Infrastrukturen, auf deren Grundlage der Betrieb der Plattform als Cloud-Service möglich gemacht werden könnte. Dafür in Frage kämen wohl auch die Anbieter Google oder Amazon, die selber Cloud-Dienste in diesen Themenbereichen anbieten.

Im Allgemeinen ließen sich auf der aktuellen Basis aber auch weitere konkrete Anwendungen entwickeln. Die Grundlage dafür ist durch die Entwicklung der Plattform so weit bereitet. Die beiden Anwendungen dieser Arbeit haben dabei mögliche Wege aufgezeigt. So würde die Verkettung einzelner Teilverarbeitungen mit neuen Analysezielen auch eine Möglichkeit der Erweiterung der vorhanden Anwendungen darstellen. Auf diesem Weg entstünde eine Art Framework aus Datentopics. Die Idee hinter diesem Framework im Rahmen der Echtzeitanalyse wäre, dass ein Schema konzipiert würde, dessen Struktur grundsätzlich mannigfaltige konkrete Analysen ermöglichen könnte.

Ein weiteres Analyseszenario könnte die Untersuchung von *.onion*-Adressen sein. Dabei handelt es sich um Adressen von Servern, die im Tor-Netz betrieben werden und auch nur von dort aus erreichbar sind. Vorausgesetzt wären ein Tor-Relay-Node mit entsprechendem Zugriff und ein zugewiesenes HSDir-Flag. Auf diesem Weg würden dann Client-Anfragen an entsprechende *.onion*-Adressen gemessen und Statistiken erhoben werden können.

Anhang

9.1 Inhalt der CD-ROM

Der Inhalt der CD-ROM ist in durch folgende Verzeichnisstruktur ersichtlich:

Thesis Die komplette Arbeit als PDF-Dokument

Literatur Abbildungen der Online-Quellen

Plattform Der Quellcode der entwickelten Plattform

9.2 Weitere Anwendungsfälle

Name	**Producer betreiben**
Beschreibung	Ein installierter Producer ist dem Zielsystem zugänglich.
Beteiligte Akteure	Betreiber
Ergebnis	Der Producer generiert einen Datenstream.
Nachbedingung	–
Standardablauf	1. Der Producer ist ordnungsgemäß installiert. 2. Der Producer ist einem Topic beziehungsweise Stream zugewiesen.

Tabelle 9.1: Anwendungsfall: Producer betreiben

Name	**Processor betreiben**
Beschreibung	Ein installierter Processor ist der Plattform zugänglich.
Beteiligte Akteure	Betreiber
Ergebnis	Der Processor ist mit einem Datenstream konnektiert.
Nachbedingung	–
Standardablauf	1. Der Processor ist ordnungsgemäß installiert. 2. Der Processor ist einem Topic beziehungsweise Stream zugewiesen.

Tabelle 9.2: Anwendungsfall: Processor betreiben

Name	**Widget betreiben**
Beschreibung	Widgets sind die Basis für die Visualisierung der verarbeiteten Daten.
Beteiligte Akteure	Konsument
Ergebnis	Der Konsument erlangt einen Mehrwert durch ein Widget.
Standardablauf	1. Das Widget ist der Plattform zugänglich gemacht worden. 2. Das Widget arbeitet wie durch den Entwickler implementiert. 3. Die Verarbeitungsergebnisse werden visualisiert.

Tabelle 9.3: Anwendungsfall: Widget betreiben

9.3 Tor-Exit-Node Statistiken

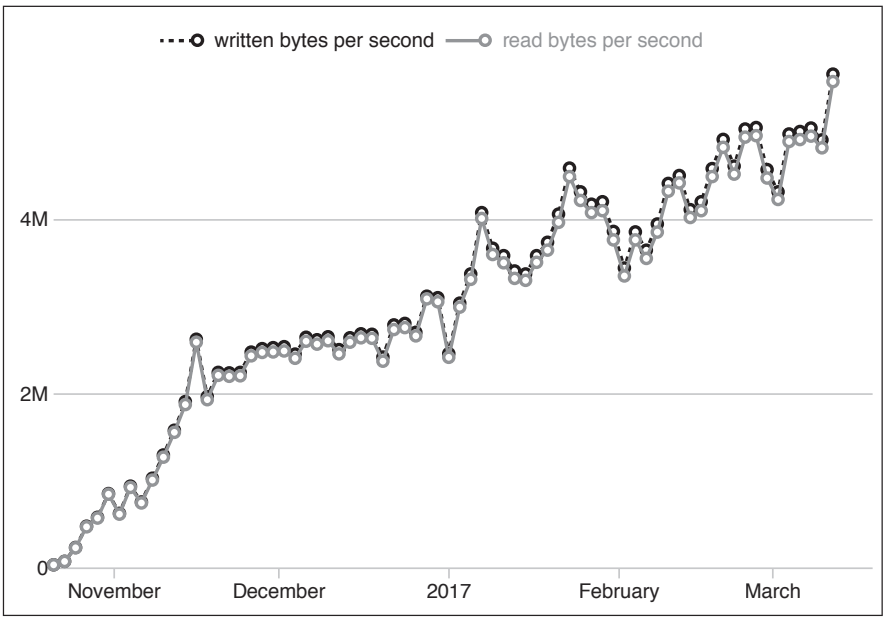

Abbildung 9.1: Tor-Relay: Written bytes per second

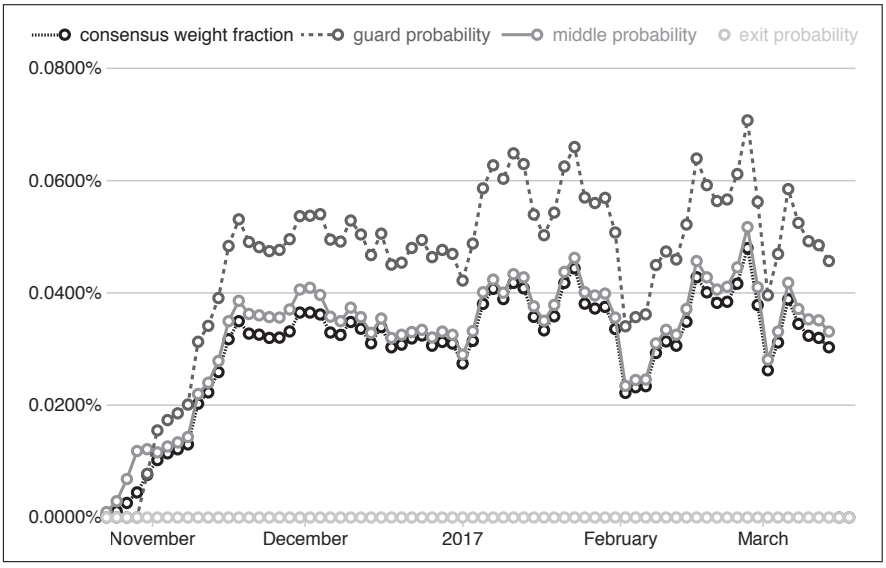

Abbildung 9.2: Tor-Relay: Consensus weight fraction

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

API Application Programming Interface

AWS Amazon Web Services

BDSG Bundesdatenschutzgesetz

CDN Content Delivery Network

CEP Complex Event Processing

CLI Command Line Interface

CSCW Computer Supported Cooperative Work

CSS Cascading Style Sheets

DNS Domain Name System

DOM Document Object Model

DSL Domain specific language

ESP Event Stream Processing

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HTTP2 Hypertext Transfer Protocol 2

IaaS Infrastructure as a Service

IP Internet Protocol

IDE Integrated Development Environment

ISP Internet Service Provider

JAR Java Archive

JS Java Script

JSON JavaScript Object Notation

LAPD Los Angeles Police Department

NASA National Aeronautics and Space Administration

NIST National Institute of Standards and Technology

OSI Open Systems Interconnection

PaaS Platform as a Service

PCAP Packet Capture

PDML Packet Details Markup Language

POC Proof of concept

SaaS Software as a Service

SDK Software Development Kit

SEP Simple Event Processing

SOA Service-Oriented Architecture

SoC Separation of concerns

SSP Single Site Pages

SQL Structured Query Language

TCP Transmission Control Protocol

Tor The onion router

UML Unified Modeling Language

URL Uniform Resource Locator

XML Extensible Markup Language

XSS Cross Site Scripting

YAML Yet Another Markup Language

WS Web Socket

WSS Web Socket Secure

Glossar

Amazon Web Services Unter den Amazon Web Services vereinigen sich verschiedene Online-Dienste des US-amerikanischen Unternehmens Amazon. 35, 110

Application Programming Interface Eine API ist eine Schnittstelle, die von einer Software angeboten wird, über die eine Anbindung an die Anwendung möglich ist. 16, 110

Bundesdatenschutzgesetz Das Bundesdatenschutzgesetz legt den Umgang mit personenbezogenen Daten in Deutschland fest. 8, 110

Cascading Style Sheets Eine Stylesheet-Sprache zum Zweck der visuellen Manipulation von Seiteninhalten bei HTML- sowie XML-basierten Dokumenten. 48, 110

Command Line Interface Eine textuelle Schnittstelle zur Interaktion kommandozeilenbasierter Anwendungen. 50, 110

Complex Event Processing Ein Teilgebiet der Informatik, das sich mit dem Erkennen und Analysieren von Daten anhand auftretender Ereignisse befasst. 12, 110

Computer Supported Cooperative Work Forschungsgebiet verschiedener Disziplinen über Gruppen und deren Zusammenarbeit mithilfe von Informations- und Kommunikationstechnologien. 9, 110

Content Delivery Network Ein CD-Netzwerk ist der Verbund physisch möglichst weit verteilter Server, die über das Internet erreichbar sind. Das Ziel ist es, Anfragen nach Ressourcen durch den nahe gelegensten im Sinne der physischen Distanz zu beantworten. 75, 110

Cross Site Scripting Die Beschreibung einer Methode des Zugriffs externer Ressourcen auf den internen Kontext von Webanwendungen. 69, 112

Document Object Model Das DOM ist die Schnittstelle für den Zugriff auf HTML- sowie XML-basierte Dokumente. 68, 110

Domain Name System Ein Dienst in IP-basierten Netzwerken mit der Aufgabe der Namensauflösung von textuellen Bezeichnern zu IP-Adressen. 67, 110

Domain specific language Eine domänenspezifische Sprache dient der Interaktion zwischen Mensch und Computern im Kontext spezieller Themengebiete (Domänen). 17, 110

Event Stream Processing Das ESP ist ein Teilbereich des CEP. Es beinhaltet primär Datenströme, deren Ordnung durch den Zeitpunkt des Auftretens eines Ereignisses definiert ist. 12, 110

Extensible Markup Language Eine textbasierte Auszeichnungssprache, durch die hierarchische Informationen beschrieben und gespeichert werden können.. 28, 112

Graphical User Interface Eine Schnittstelle zur Interaktion von Menschen mit Computern. 34, 110

Hypertext Markup Language Eine textuelle Auszeichnungssprache mit dem Zweck der Strukturierung von Informationen in hierarchischer Form. 48, 110

Hypertext Transfer Protocol Ein Netzwerkprotokoll zum Datenübertragen zwischen zwei Netzwerkteilnehmern. Es ist auf der Anwendungsschicht des OSI-Modells angesiedelt. 31, 110

Hypertext Transfer Protocol 2 Die Weiterentwicklung des HTTP-Protokolls in der Version 1.1. 65, 110

Hypertext Transfer Protocol Secure Die Erweiterung des HTTP-Protokolls mit dem Zweck der Transportverschlüsselung der Daten bei der Kommunikation zwischen zwei Netzwerkteilnehmern. 31, 110

Infrastructure as a Service Eine Art des Cloud Computings, in der eine technische Infrastruktur gebucht werden kann. 10, 110

Integrated Development Environment Eine Sammlung von Anwendungsprogrammen mit dem Zweck der nahtlosen Entwicklung von Software im Kontext spezieller Domänen. 54, 111

Internet Protocol Ein Netzwerkprotokoll auf der Vermittlungsschicht des OSI-Modells mit dem Zweck der Adressierungsmöglichkeit verschiedener Teilnehmer eines Netzes. 58, 110

Internet Service Provider Kurz: Internetdienstanbieter. Ein Anbieter, dessen Dienstleistung in der Bereitstellung von Internetzugängen liegt. 78, 111

Java Archive Ein komprimiertes Dateiformat (ZIP) mit dem Zusatz spezieller Metainformationen in textueller Form. 38, 111

Java Script Eine Script-Sprache, die anfänglich nur für eine Möglichkeit der dynamischen Interaktion mit Webseiten gedacht war, mittlerweile aber auch auf Servern eingesetzt wird. 48, 111

JavaScript Object Notation Eine aus JavaScript stammende textuelle Darstellung von Objekten mit dem Ziel der Möglichkeit der Serialisierung sowie Deserialisierung. 28, 111

Los Angeles Police Department Die Polizeibehörde der Stadt Los Angeles im Bundesstaat Kalifornien der Vereinigten Staaten von Amerika. 7, 111

National Aeronautics and Space Administration Die zivile Behörde für Raumfahrt und Flugwissenschaft der USA. 4, 111

National Institute of Standards and Technology Eine US-amerikanische Bundesbehörde mit der Zuständigkeit für Standardisierungsprozesse in der Informationstechnologie. 10, 111

Open Systems Interconnection Das OSI-Modell gliedert verschiedene Netzwerkprotokolle in die jeweiligen Schichten des Modells. Dabei werden sieben verschiedene Schichten unterschieden. 65, 111

Packet Capture Ein Dateiformat zur Speicherung von Netzwerkdatenmitschnitten. 28, 111

Packet Details Markup Language Eine auf XML basierende Auszeichnungssprache von Netzwerkdatenpaketen. 28, 111

Platform as a Service Eine Art des Cloud Computings, in der eine Plattform gebucht werden kann. 10, 111

Proof of concept Ein POC ist ein Artefakt einer Software, das die grundsätzliche Machbarkeit eines Vorhabens belegt. 17, 111

Separation of concerns Ein aus der Softwareentwicklung stammendes Paradigma, nach dem bei der Entwicklung der Software auf eine Trennung der Zuständigkeiten Rücksicht genommen werden sollte. 26, 111

Service-Oriented Architecture Ein Architekturmuster zur Strukturierung und zum Zusammenfügen verschiedener Komponenten zu einem gesamtheitlichen System. 21, 111

Simple Event Processing Eine Art von Prozessen in ereignisgesteuerten Architekturen. Hierbei lösen Prozesse direkt einfache Ereignisse aus. 20, 111

Single Site Pages Eine auf HTML basierte Webanwendung, deren Inhalte dynamisch nachgeladen werden ohne ein komplettes Neuladen der Ursprungsseite. 48, 111

Software as a Service Eine Art des Cloud Computings, in der eine Anwendung gebucht werden kann. 10, 111

Software Development Kit Ein SDK umfasst verschiedene Programme zum Zweck der Entwicklung von Software. 37, 111

Structured Query Language Eine Sprache in der Domäne der relationalen Datenbanken zum Ausführen der gängigen CRUD-Operationen (create, read, update, delete) auf der konkreten Datenbank. 36, 111

The onion router Ein Overlay-Netzwerk mit dem primären Ziel der Anonymisierung seiner Nutzer. 63, 111

Transmission Control Protocol Ein Netzwerkprotokoll, das für den Datenaustausch netzwerkfähiger Knoten konzipiert ist. Im OSI-Modell ist es ein Protokoll der Transportschicht. 78, 111

Unified Modeling Language „Die Sprache dient der Visualisierung, Spezifizierung, Konstruktion und Dokumentation der Artefakte eines softwareintensiven Systems“ Booch u. a. (2006). 111

Uniform Resource Locator Ein Schema der Identifikation und Lokalisation einer Ressource. 67, 112

Web Socket Ein Netzwerkprotokoll mit der Möglichkeit der bidirektionalen Kommunikation zwischen einem Web-Server und einer Web-Anwendung.. 49, 112

Web Socket Secure Die Erweiterung des WebSocket-Protokolls mit dem Zweck der Transportverschlüsselung von Daten zweier Teilnehmer. 52, 112

Yet Another Markup Language Eine an XML angelehnte Auszeichnungssprache zum Zweck der textuellen Serialisierung sowie Deserialisierung von Daten. 54, 112

Literaturverzeichnis

[Amazon Web Services Inc. 2017] Amazon Web Services Inc.: *Informationen zu AWS*. Januar 2017. – URL https://aws.amazon.com/de/about-aws/

[American Multinational Corporation 2011] American Multinational Corporation: *Datenwachstum verdoppelt sich alle zwei Jahre*. Juni 2011. – URL http://germany.emc.com/about/news/press/2011/20110628-01.htm

[Apache Software Foundation 2016a] Apache Software Foundation: *Apache Kafka*. Dezember 2016. – URL https://kafka.apache.org/intro

[Apache Software Foundation 2016b] Apache Software Foundation: *Apache Kafka*. Dezember 2016. – URL https://kafka.apache.org/documentation.html#streams

[Apache Software Foundation 2016c] Apache Software Foundation: *Apache Zookeeper*. Dezember 2016. – URL https://zookeeper.apache.org/doc/trunk/zookeeperOver.html

[Assunção u. a. 2015] Assunção, Marcos D. ; Calheiros, Rodrigo N. ; Bianchi, Silvia ; Netto, Marco A. ; Buyya, Rajkumar: Big Data computing and clouds: Trends and future directions. 79–80 (2015), S. 3 – 15. – URL http://www.sciencedirect.com/science/article/pii/S0743731514001452. – Special Issue on Scalable Systems for Big Data Management and Analytics. – ISSN 0743-7315

[Barbier und Recoussine 2014] Barbier, Franck ; Recoussine, Jean-Luc: Service-Oriented Architecture (SOA). (2014), S. 59–78

[Blau 2009] Blau, Danny: *Das Moore'sche Gesetz*. GRIN Verlag, 2009. – ISBN 978-3-640-44943-9

[Booch u. a. 2006] Booch, Grady ; Rumbaugh, James ; Jacobson, Ivar: *Das UML-Benutzerhandbuch - aktuell zur Version 2.0*. 1. Auflage. Pearson Deutschland GmbH, 2006. – ISBN 978-3-827-32295-1

[Bruns und Dunkel 2015] Bruns, Bruns ; Dunkel, Dunkel: *Complex Event Processing - Komplexe Analyse von massiven Datenströmen mit CEP*. 1. Auflage. Springer-Verlag, 2015. – ISBN 978-3-658-09899-5

[Buyya u. a. 2000] Buyya, Rajkumar ; Baker, Mark ; Hyde, Daniel C. ; Tavangarian, Djamshid: *Cluster Computing*. S. 1115–1117. In: Bode, Arndt (Hrsg.) ; Ludwig, Thomas (Hrsg.) ; Karl, Wolfgang (Hrsg.) ; Wismüller, Roland (Hrsg.): *Euro-Par 2000 Parallel Processing: 6th International Euro-Par Conference Munich, Germany, August 29 – September 1, 2000 Proceedings*, Springer Berlin Heidelberg, 2000. – URL http://dx.doi.org/10.1007/3-540-44520-X_158. – ISBN 978-3-540-44520-3

[Buyya u. a. 2016] Buyya, Rajkumar ; Calheiros, Rodrigo N. ; Dastjerdi, Amir V.: *Big Data - Principles and Paradigms*. 1. Auflage. Morgan Kaufmann, 2016. – ISBN 978-0-128-09346-7

[Conconi 2015] Conconi, Luca: *Das Mooresche Gesetz. Eine ewige Konstante?* 1. Auflage. GRIN Verlag, 2015. – ISBN 978-3-668-10707-6

[Cox und Ellsworth 1997a] Cox, Michael ; Ellsworth, David: Application-controlled Demand Paging for Out-of-core Visualization. In: *Proceedings of the 8th Conference on Visualization '97*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1997 (VIS '97), S. 235–ff.. – URL http://dl.acm.org/citation.cfm?id=266989.267068. – ISBN 1-58113-011-2

[Cox und Ellsworth 1997b] Cox, Michael ; Ellsworth, David: Managing big data for scientific visualization. In: *ACM Siggraph* Bd. 97, 1997, S. 146–162

[Docker Inc. 2016] Docker Inc.: *COMPARING CONTAINERS AND VIRTUAL MACHINES*. Dezember 2016. – URL https://www.docker.com/what-docker

[Docker Inc. 2017] Docker Inc.: *Docker Swarm overview - Docker*. Januar 2017. – URL https://docs.docker.com/swarm/overview/

[emetriq GmbH 2017] emetriq GmbH: *Vision und mission*. März 2017. – URL https://www.emetriq.com/vision-und-mission/

[Freiknecht 2014] Freiknecht, Jonas: *Big Data in der Praxis - Beispiellösungen mit Hadoop und NoSQL. Daten speichern, aufbereiten, visualisieren*. Carl Hanser Verlag GmbH Co KG, 2014. – ISBN 978-3-446-44177-4

[Gilbert und Lynch 2012] Gilbert, Seth ; Lynch, Nancy: Perspectives on the CAP Theorem. 45 (2012), Nr. 2, S. 30–36

[Google Inc. 2017] Google Inc.: *Cloud Dataflow - Batch & Stream Data Processing | Google Cloud Platform*. Januar 2017. – URL https://cloud.google.com/dataflow/

[Henze 2014] Henze, Mirko: Web 3.0: Daten sind das Öl des 21. Jahrhunderts. (2014)

[Hilbert und López 2011] Hilbert, Martin ; López, Priscila: The world's technological capacity to store, communicate, and compute information. 332 (2011), Nr. 6025, S. 60–65

[Hillebrand und Finger 2015] Hillebrand, Rainer ; Finger, Lars: Einkaufen in der Zukunft: Wie die Digitalisierung den Handel verändert. In: *Digitales Neuland*. Springer, 2015, S. 89–101

[Hirsch 2013] Hirsch, Dennis D.: Glass House Effect: Big Data, the New Oil, and the Power of Analogy, The. 66 (2013), S. 373

[INFOnline GmbH 2017] INFOnline GmbH: *Leistungsspektrum INFOnline GmbH*. März 2017. – URL https://www.infonline.de/unternehmen/leistungsspektrum/

[International Organization for Standardization 2016] International Organization for Standardization: *ISO/IEC 2382:2015 - Information technology - Vocabulary*. 2016. – URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=63598

[Irfani 2015] Irfani, Irfani: *Implementasi High Availability Server Dengan Teknik Failover Virtual Computer Cluster*. 2015

[Kreps u. a. 2011] Kreps, Jay ; Narkhede, Neha ; Rao, Jun u. a.: Kafka: A distributed messaging system for log processing. In: *Proceedings of the NetDB*, 2011, S. 1–7

[Laney 2001] Laney, Doug: 3D data management: Controlling data volume, velocity and variety. 6 (2001), S. 70

[Langheinrich und Mattern 2003] Langheinrich, Marc ; Mattern, Friedemann: Digitalisierung des Alltags. Was ist Pervasive Computing. 42 (2003), S. 6–12

[Liu und Zhao 2014] Liu, Di ; Zhao, Libin: The research and implementation of cloud computing platform based on docker. In: *Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference on* IEEE (Veranst.), 2014, S. 475–478

[LL.M. u. a. 2014] LL.M., Thomas S. ; Dechamps, Catherine ; LL.M., Henning F. ; LL.M., Wolfgang F. ; Funk, Axel ; Heinbuch, Holger ; Eur., Michael Schmidl L. ; Schrey, Joachim: *Handbuch IT-Outsourcing - Recht, Strategien, Prozesse, IT, Steuern und Cloud Computing*. 4. Auflage. C.F. Müller GmbH, 2014. – ISBN 978-3-811-43805-7

[Luckham 2006] Luckham, David: *What's the Difference Between ESP and CEP?* August 2006. – URL http://www.complexevents.com/2006/08/01/what%E2%80%99s-the-difference-between-esp-and-cep/

[Mell u. a. 2011] Mell, Peter ; Grance, Tim u. a.: The NIST definition of cloud computing. (2011)

[Merkel 2014] Merkel, Dirk: Docker: lightweight linux containers for consistent development and deployment. 2014 (2014), Nr. 239, S. 2

[Michelson 2006] Michelson, Brenda M.: Event-driven architecture overview. 2 (2006)

[Miller und Mork 2013] Miller, H G. ; Mork, Peter: From data to decisions: a value chain for big data. 15 (2013), Nr. 1, S. 57–59

[Mouat 2016] Mouat, Adrian: *Januar - Software entwickeln und deployen mit Containern*. dpunkt.verlag, 2016. – ISBN 978-3-960-88037-0

[Mozilla Developer Network 2016] Mozilla Developer Network: *Network Monitor - Firefox Developer Tools | MDN*. Dezember 2016. – URL https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor

[Nwosu u. a. 2012] Nwosu, Kingsley C. ; Thuraisingham, B. ; Berra, P. B.: *Multimedia Database Systems - Design and Implementation Strategies*. Springer Science & Business Media, 2012. – ISBN 978-1-461-30463-0

[Oyman und Singh 2012] Oyman, Ozgur ; Singh, Sarabjot: Quality of experience for HTTP adaptive streaming services. 50 (2012), Nr. 4

[Papazoglou 2003] Papazoglou, Mike P.: Service-oriented computing: Concepts, characteristics and directions. In: *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on* IEEE (Veranst.), 2003, S. 3–12

[Pippal u. a. 2014] Pippal, Sanjeev ; Singh, Shiv P. ; Kushwaha, Dharmender S.: Data Trasfer From MySQL To Hadoop: Implementers' Perspective. In: *Proceedings of the 2014 International*

Conference on Information and Communication Technology for Competitive Strategies. New York, NY, USA : ACM, 2014 (ICTCS '14), S. 79:1–79:5. – URL http://doi.acm.org/10.1145/2677855.2677934. – ISBN 978-1-4503-3216-3

[Porter 1985] Porter, Michael E.: *Competitive Advantage - Creating and Sustaining Superior Performance*. 1. Auflage. Simon and Schuster, 1985. – ISBN 978-1-416-59584-7

[Retter 2011] Retter, Sascha J.: Architektur und Implementierung ereignis-und situationsgetriebener Workflows. (2011)

[Robins 2010] Robins, D: Complex event processing. In: *Second International Workshop on Education Technology and Computer Science. Wuhan* Citeseer (Veranst.), 2010

[Schill und Springer 2012] Schill, Alexander ; Springer, Thomas: *Verteilte Systeme: Grundlagen und Basistechnologien*. Springer-Verlag, 2012

[Schlegel 2013] Schlegel, Mario: *Konzeption und prototypische Entwicklung einer CEP-Anwendung im Bereich E-Commerce* -. 1. Auflage. Diplomica Verlag, 2013. – ISBN 978-3-842-89259-0

[Schulte und Bülchmann 2016] Schulte, Alexandra ; Bülchmann, Oliver: *Wie Big Data die Rolle des Controllers verändert*. S. 54–60. In: Schäffer, Utz (Hrsg.) ; Weber, Jürgen (Hrsg.): *Controlling & Management Review Sonderheft 1-2016: Big Data - Zeitenwende für Controller*, Springer Fachmedien Wiesbaden, 2016. – URL http://dx.doi.org/10.1007/978-3-658-13444-0_7. – ISBN 978-3-658-13444-0

[Siegel 2016] Siegel, Eric: *Predictive Analytics - The Power to Predict Who Will Click, Buy, Lie, or Die*. 2. Auflage. John Wiley and Sons, 2016. – ISBN 978-1-119-15365-8

[Sill 2016] Sill, Alan: The Design and Architecture of Microservices. 3 (2016), Nr. 5, S. 76–80

[The Linux Foundation 2017] The Linux Foundation: *Kubernetes - Production-Grade Container Orchestration*. Januar 2017. – URL https://kubernetes.io/

[Thönes 2015] Thönes, Johannes: Microservices. 32 (2015), Nr. 1, S. 116–116

[Uskali und Kuutti 2015] Uskali, Turo I. ; Kuutti, Heikki: Models and streams of data Journalism. 2 (2015), Nr. 1, S. 77–88

[w3.org 2017] w3.org: *4.7 Embedded content — HTML5*. Februar 2017. – URL https://www.w3.org/TR/html5/embedded-content-0.html#embedded-content-0

[Waldrop 2016] Waldrop, M. M.: MORE THAN MOORE. (2016), Februar, Nr. 530, S. 144–147

[Ward und Barker 2013] Ward, Jonathan S. ; Barker, Adam: Undefined by data: a survey of big data definitions. (2013)

[Weichert 2013] Weichert, Thilo: Big Data und Datenschutz. (2013)

[Weiss 2009] Weiss, Integrating Real-Time Predictive A.: into SAP Applications [online], Dec. 30, 2009 [retrieved on Sep. 29, 2011]. (2009)

[Wirtschaftslexikon 2016] Wirtschaftslexikon, Gabler: *Gabler Wirtschaftslexikon*. November 2016. – URL http://wirtschaftslexikon.gabler.de/Archiv/-2046774198/big-data-v3.html

[Yu und Guo 2016] Yu, Shui ; Guo, Song: *Big Data Concepts, Theories, and Applications* -. 1. Auflage. Springer, 2016. – ISBN 978-3-319-27763-9

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 8. Mai 2017 Micha Severin