FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Classification and Monitoring of Incidents in Cloud-based Big Data Analytics

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Information Computing

eingereicht von

Manfred Halper, Bakk.rer.soc.oec.
Matrikelnummer 0007821

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Privatdoz. Dr.techn. Hong-Linh Truong

Wien, 25. Jänner 2018

Manfred Halper	Hong-Linh Truong

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Classification and Monitoring of Incidents in Cloud-based Big Data Analytics

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Information Computing

by

Manfred Halper, Bakk.rer.soc.oec.
Registration Number 0007821

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dr.techn. Hong-Linh Truong

Vienna, 25th January, 2018

Manfred Halper	Hong-Linh Truong

Erklärung zur Verfassung der Arbeit

Manfred Halper, Bakk.rer.soc.oec.
Millergasse 29/17, 1060 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. Jänner 2018

Manfred Halper

Kurzfassung

Cloud Computing bildet die Basis für hochgradig verteilte und ressourcen-intensiven Big Data Anwendungen. Deren Aufgabe ist es die von intelligenten Objekten, Multimedia, sozialen Medien, Forschung und Unternehmen erzeugten Datenmengen zu verarbeiten. Jeder in der Datenverarbeitung involvierte Akteur hat seine eigenen Methoden um Störungen zu verwalten. Diese isolierte Betrachtungsweise ist im Falle von Cloud-basierten Big Data Analyse Anwendungen hinderlich. Der Ausfall eines einzigen Elements der verteilten Anwendung hat Auswirkungen auf den gesamten Analyse Prozess.

Das Ziel dieser Diplomarbeit ist es Grundlagen zu erarbeiten, die die Cloud-basierte Big Data Analyse Anwendung als Ganzes und unter Einbeziehung ihrer spezifischen Bedürfnisse betrachtet. Realisiert wird dieses Ziel durch die Durchführung einer Stakeholder Analyse inklusive der Identifikation von Rollen und Akteuren, der Identifikation und Beschreibung generischer Anwendungsfälle, einer Studie typischer Störungen, der Entwicklung einer Klassifikation von Cloud-basierten Big Data Analyse Störungen und dem Erfassen von Metriken, die für die Identifikation und Klassifizierung von Störungen benötigt werden.

Wir haben eine generische Architektur eines Verwaltungssystems für Störungen entwickelt, die den unterschiedlichen Bedürfnissen von Cloud-basierten Big Data Analyse Anwendungen angepasst werden kann. Die Basis dieser Architektur bilden Stakeholder, Anwendungsfälle, Klassifikation und Softwarekomponentendiagramme. Merkmale bestehender Architekturen wurden erfasst und in den Kontext von Cloud-basierten Big Data Anwendungen übersetzt. Ein Prototyp samt Testszenario bewertet die Architektur und belegt deren Realisierbarkeit.

Abstract

Cloud Computing delivers the resources for highly distributed and resource intensive Big Data applications that analyses the massive amount of data produced by Internet of Things (IoT), multimedia, social media, operation of enterprises, trading of enterprises and scientific research. In the data transformation process various stakeholders add their knowledge and expertise. The problem is that each of them has its own methodologies and routines for incident management but none of them sees the process as a whole. A failure of single elements in the data transformation process can lead to a domino effect that impacts the whole process.

This thesis aims to deliver groundwork regarding an incident management process by applying an end-to-end approach that considers the specific needs of Cloud-based Big Data analytics. We present a stakeholder analysis including roles and actors. We describe the necessary generic use cases that need to be implemented. An incident survey details the characteristics of incidents of Cloud-based Big Data analytics. We develop a classification of cloud hosted Big Data analytic incidents from the survey. Metrics are added to describe the important elements in the data transformation process.

We introduce a generic architecture of an incident management system that can be adapted to the specific needs of Cloud-based Big Data analytics applications. By delivering common building blocks like stakeholders, use cases, classifications and software component diagrams, the essentials of existing architectures are captured and put into the perspective of the Cloud-based Big Data analytics application. We deliver a prototype for the evaluation of the architecture and we illustrate with a test scenario the viability of the generic architecture.

Contents

List of Figures

List of Tables

Listings

Acronyms

API Application Programming Interface. 41

BTS Base transceiver station. 19, 25, 27, 36, 43, 45, 73, 75

CERT/CC Computer Emergency Response Team Coordination Centre. 2

CS Cloud Service. xv, 2, 3, 7, 8, 10, 11, 13, 20, 22, 25, 41, 64

CSC Cloud Service Customer. 1, 2, 8, 10, 20, 41

CSN Cloud Service Partner. 1, 8, 10

CSP Cloud Service Provider. 8, 10, 12, 20, 21, 60

ENISA European Union Agency for Network and Information Security. 2

GFS Google's file system. 10, 11

GPS Global Positioning System. 29, 32

HDFS Hadoop Distributed File System. 10, 11, 76, 81

IaaS Infrastructure as a Service. 1, 7, 20, 22

IoT Internet of Things. ix, xv, 1–3, 8–13, 15, 19, 21, 22, 28–30, 32, 39, 40, 42–44, 46, 47, 73, 87

ISO International Organization for Standardization. 2, 7

IT Information Technology. 12

ITIL IT Infrastructure Library. 2, 13, 49, 50

NIST National Institute of Standards and Technology. 2, 7

NoSQL Not only SQL. 10, 11

PaaS Platform as a Service. 2, 7, 20, 22

QoA Quality of Analytics. 38, 40, 42–45, 47, 55, 57, 60, 63, 64, 87

QoS Quality of Service. 40–43, 46, 47, 55, 57, 60, 63, 64, 87

RCA Root Cause Analysis. 13, 14, 16, 27

SaaS Software as a Service. 2, 7, 22

SLA Service Level Agreement. 24–27, 41, 47, 56, 57, 60, 63, 64

SLI Service Level Indicator. 24, 25, 47, 60, 64

SLO Service Level Objective. 24, 25, 27, 43, 47, 55, 60, 64

CHAPTER 1

Introduction

Cloud Computing, Big Data and IoT are emerging technologies that are often considered individually but in reality they are interconnected. IoT produces vast amounts of data, Big Data's main purpose is the analysis of enormous unstructured and structured data sets and Cloud Computing delivers a platform that can flexibly satisfy high demands on storage technology, computing power and Internet throughput required by Big Data. With the move away from self hosted infrastructure and into the cloud, the way incidents are identified changes significantly and has to be adapted to the requirements of the different stakeholders. This thesis abstracts the Cloud-based Big Data analytics application to establish a generic architecture for incident management.

1.1 Problem Description

The three technology paradigm Cloud Computing, Big Data and IoT are linked in their emergence. Cloud Computing at the peak of the Gartner hype cycle in 2010[43] forms the basis for resource intensive applications like Big Data. Big Data emerged at the peak of the hype cycle in the year 2013[44] fully embracing the potential and abilities of Cloud Computing. Enabled by the maturity of Big Data IoT entered the hype cycle in 2014 [45]. IoT stayed there for 2015[46] and re-emerged in form of IoT platforms in 2017[47]. Cloud Computing is the key enabler for Big Data and Big Data is the key enabler for the analysis of IoT data. But to this point and partly caused by the fast emergence of Cloud Computing, Big Data and IoT, they are separated regarding incident management. To the best of our knowledge there are no tools to support a coherent end-to-end view of the incident detection, classification and management of Cloud-based Big Data analytics. Several difficulties depend on the deployment model of the cloud. Infrastructure as a Service (IaaS) typically provides more events, monitoring data and alerts then Software as a Service (SaaS) and Platform as a Service (PaaS) since in this deployment model the Cloud Service Customer (CSC) usually has more control over

the underlying operating infrastructure[32]. The large number of involved parties (e.g. other CSCs, Cloud Service Partner (CSN),...) complicates cloud incident handling compared to conventional incident management, where only one particular organisation is involved[87].

Incident handling for one particular organisation is a mature and understood topic[63] [64] [65] [90] [91]. Incident handling in Cloud-based Big Data analytics is complicated by the key characteristics of Cloud Computing, Big Data and IoT. Key characteristics for Cloud Computing include accessibility of resources over the network (broad network access), services consumed by multiple users (multi-tenancy), fast decrease and increase of resources (rapid elasticity and scalability) and no control or knowledge over the location of resources (resource pooling)[35]. Some key characteristics for Big Data[62] [40] are the size of data (Volume), the speed of incoming and outgoing data (Velocity), the sources and types of data (Variety) and the messiness and trustworthiness of data (Veracity). Key challenges of IoT regarding data generation are real-time processing and the enormously large amount of produced data[95][2].

The thesis will progress the field of incident management in Cloud-based Big Data analytics by:

1. Supporting stakeholders establishing an incident management process in Cloud-based Big Data analytics applications
2. Identifying missing support regarding key incidents

There are several approaches and recommendations for Cloud Computing incident management [88][58] of different organisations like Computer Emergency Response Team Coordination Centre (CERT/CC), National Institute of Standards and Technology (NIST) [57], International Organization for Standardization (ISO)[34], European Union Agency for Network and Information Security (ENISA), SANS Institute and IT Infrastructure Library (ITIL) but there is no study regarding a generic end-to-end model and architecture for incident management in Cloud-based Big Data analytics.

This thesis aims to answer the following research questions:

RQ1: What is the current state of the art of development of incident management/detection in Cloud-based Big Data analytics and what are the next steps to improve it?

RQ2: What are current challenges that organisations experience when they are trying to establish incident management in an environment where Big Data applications analyse IoT data on top of hosted CSs?

Currently there is no consistent definition of incidents in Cloud-based Big Data analytics. There are no certifications for tools, practices and trainings related to Cloud-based Big

Data analytics incident handling. The first research question addresses the problem that, while Cloud-based Big Data analytics popularity is increasing, the incident management capabilities of the various stakeholders involved in Cloud-based Big Data analytics do not evolve fast enough and are often isolated from applications and cloud-hosted external systems.

The different deployment models of CSs add complexity to the challenge of an incident management system. Service level agreements are the core of any incident management endeavour in Cloud-based Big Data analytics and need to be integrated in a thorough incident management system.

The major aim of the thesis is to provide support for incident management in cloud hosted Big Data systems. This includes the following objectives:

O1: A systematic study of the three involved technologies and their specific needs regarding incident management.

O2: Finding stakeholders and use cases specific to the scenario

O3: Conduct a survey of incidents specific to Cloud-based Big Data analytics with IoT data input

O4: Find metrics applicable for incident detection

O5: Introduce a generic architecture for an incident management system

O6: Illustrate the feasibility of the architecture by implementing a proof-of-concept with already existing software

The sixth objective is a practical approach on the topic of incident management. This objective explains how to access and collect relevant data for the incident management process from the systems involved in Cloud-based Big Data analytics. The purpose of the sixth objective is to prove that the generic architecture derived from the second, third, fourth and fifth objective conceptually works. The practical approach either analyses live - incident relevant - data from IoT or simulates incidents to prove the feasibility of the generic architecture.

1.2 Methodological Approach

The first step is a study of traditional incident management as well as new developments that are specific to Cloud Computing, Big Data and IoT. The study emphasises how the split responsibility of the different stakeholders changes the current incident management process. The outcome of the study is a list of generic stakeholders and corresponding use cases that are necessary for a generic incident management system. The next step is an incident survey that finds and describes incidents that are specifically tailored to the

system seen as a whole. The study identifies the set of incidents that - if not managed - negatively impact the progress of Cloud-based Big Data analytics. The set of incidents delivers an information basis for a classification that enables the design of a generic incident management architecture. The components of the architecture are implemented in a proof-of-concept:

The proof-of-concept consists of different parts, conceivable at the moment:

P1: A raw data collector which uses the interfaces of the services to gather relevant data for the incident management process. The raw data collector is extensible so that future manageable applications can be added to the incident management system.

P3: A rule-set that performs incident detection and classification on the clean data set to identify distinct incidents.

The proof-of-concept is very extensible to satisfy the diversity of Cloud-based Big Data analytics. The last step provides conclusions and future work regarding incident management of Cloud-based Big Data analytics.

1.3 Contributions

This thesis introduces a new generic architecture for incident monitoring and classification in Cloud-based Big Data analytics. An intensive related and background research delivers the basis for the following contributions:

(i) Stakeholder analysis, identifying generic use cases, roles and actors tailored to the requirements of Cloud-based Big Data analytics.

(ii) Incident survey identifying generic high-level incidents specific to Cloud-based Big Data analytics.

(iii) Generic classification scheme for incidents of Cloud-based Big Data analytics including a set of exemplary key-value metrics.

(iv) Describing software components of a generic architecture for incident management of Cloud-based Big Data analytics.

(v) Implementing a proof-of-concept of the generic architecture for incident management of Cloud-based Big Data analytics. The software used in the proof-of-concept is released as open source under `https://github.com/rdsea/bigdataincidentanalytics`.

1.4 Thesis Structure

The rest of the thesis is structured as follows:

- Chapter 2 present the intensive related work and background research.
- Chapter 3 shows the stakeholder analysis and relevant use cases for the generic incident management architecture.
- Chapter 4 the results of the incident survey are presented and the classification is described with the exemplary set of key-value metrics.
- Chapter 5 specifies the monitoring requirements and presents the software components of the generic architecture.
- Chapter 6 presents the proof-of-concept.
- Chapter 7 summarises the findings of this thesis and the future work is outlined.

CHAPTER 2

Background and State Of The Art

2.1 Cloud Computing

Cloud Computing is a term that emerged in the last few years, there are mainly two definitions of Cloud Computing one from NIST [76] and one from ISO. [37] *"Cloud Computing is a paradigm for enabling network access to a scalable and elastic pool of shareable physical or virtual resources with self-service provisioning and administration on-demand"* [37].

Cloud Computing touches several research fields and it is difficult to define the various areas of application. The most common classification from NIST divides the service models in (1) SaaS (2) PaaS and (3) IaaS. The ISO standard adds an additional layer by differentiating between cloud capabilities (application, infrastructure and platform) and cloud service categories. The later represents a group of CSs that possess a common set of qualities. The complexity and growth of cloud services is best illustrated by the vast amount of as-a-Service offerings and categories: Sugam Sharma in [95] from 2016 lists 78 new as-a-Service models. This illustrates how fast the as-a-Service modality of Cloud Computing is evolving and how narrow the basic classification into three service models (SaaS, PaaS, IaaS) is. The explosive growth of service models is partly due to the fact of the big growth of data and the need of ubiquitous information access. Another driving factor is that data as well as information have emerged as one of the most valuable resources in the 21st century. The growth of data has fuelled the need for new computation models and raised the overall available data that is in demand globally across all disciplines[95]. The key characteristics of Cloud Computing influencing incident management are *"broad network access, measured service, multi-tenancy, on-demand self service, rapid elasticity and scalability, resource pooling and payment on a as-need basis"*[35].

Cloud Computing and Big Data are connected. The emergence of Cloud Computing provides solutions for the storage and processing needs of Big Data[19]. Cloud Computing

has to handle the incoming flow of information of IoT while delivering the basis for the Cloud-based Big Data analytics application that generates value out of the collected data.

The International Organization for Standardization defines in [35] three major roles in Cloud Computing, Cloud Service Provider (CSP), CSC and CSN. The CSC uses CSs from the CSP and/or the CSN. The CSC performs business administration and manages the used CSs. The CSN supports the CSC and/or CSP in their activities. The CSP makes CSs available. The CSP focus lies on the provision, maintenance and delivery of CSs to the CSC. The extensive set of activities the CSP delivers includes but are not restricted to providing services, deploying services, monitoring services, providing audit data and managing the business plan.

2.2 Internet of Things

IoT combines aspects and technologies of ubiquitous computing, pervasive computing, Internet Protocol, sensing technologies, communication technologies and embedded devices. The building block of this paradigm is the smart object, an everyday object infused with intelligence that collects data from the environment, interacts/controls the physical world and is interconnected to other smart objects to exchange data and information[16].

An important concept regarding the management of incidents related to IoT are the three different phases that illustrate the physical-cyber world interactions of smart devices[16]. During the collection phase the device uses procedures for sensing the physical environment. It collects real-time physical data and constructs a general perception of it. The IoT device delivers the collected data to applications and servers during the transmission phase. During the processing, managing and utilisation phase information flows are analysed and processed, data is forwarded to applications and services and feedback is provided to control the applications[16].

IoT devices produce an enormously large amount of data that needs to be stored, processed, analysed and visualised efficiently to create value from it. The concepts of Cloud Computing and Big Data offer solutions to the demands of IoT by elastically providing the necessary resources and robust services to handle the data needs of IoT. The data challenges of IoT, the necessity of robust but affordable computing resources and the automation in humans routine tasks promised by IoT push the agenda of as-a-Service Cloud Computing models and Big Data[95].

Data generated by IoT pose several challenges. The data collection in the context of sensors often exhibit natural errors and incompleteness that lead to pollution in the stored data and in succession to errors in the processing of the whole data set[2]. IoT applications often require real-time processing that put high demands on the underlying systems since real-time processing needs significantly more resources than near real-time and/or batch processing. Real-time processing also changes the sequence of activities in

the Big Data abstraction regarding several pre processing steps that cause even more stress on computing resources.

Another challenge posed by IoT are the large volumes of data that subsequently stress storage and processing capabilities of the underlying system[2]. The generation and acquisition phase of the Cloud-based Big Data analytics application is simulated with collected real life samples. Regarding analytics IoT can be viewed as a sensor network that needs real-time analysis or as a way to collect massive amounts of data for later analysis in a batch based system.

This duality leads to a split in incident management. Incidents that occur during (near) real-time analysis and incidents that can manifest during the batch based analysis. Real-time data processing is used in web-analytics, security event monitoring, optimising devices (IoT devices based on behaviour and usage) and control system related tasks (SmartGrid)[74].

IoT, Big Data and Cloud Computing are interdependent technologies and therefore incident management has to view these systems as a whole. IoT has a large volume of data traffic, involves diverse data from heterogeneous sources and produces inherently messy unstructured data at high velocity [3]. This data characteristics of IoT lead to challenges in terms of processing and storage of the data[2]. These problems are only solvable with the techniques employed by Big Data. Big Data needs flexible and foremost vast amounts of resources. This amount of resources can only be supplied by Cloud Computing that enables Big Data applications with a scalable and elastic pool of shareable physical or virtual resources[37].

2.3 Big Data

Initially Big Data was defined by the 3V model from Doug Laney where the 3V stand for Volume, Velocity and Variety[62]. This definition however has been extended by IDC which added Value[40] and by Microsoft and IBM which added Veracity or Variability as a fourth characteristic. The trend to use words starting with a "V" continued and currently some of the key characteristics associated with Big Data are volume, velocity, variety, veracity, value, variability, volatility and validity.

There are various definitions of generic frameworks for Big Data but one that has been tested against established Big Data applications is from Pääkkönen and Pakkala [83]. They developed a reference architecture that is mapped to several use cases including Facebook, LinkedIn, Twitter and Netflix. Big Data sources are IoT, multimedia, social media, operation of enterprises, trading of enterprises and scientific research[19].

Big Data acquisition includes data extraction, loading, pre-processing and data storage where it is kept for further processing. During data collection raw data is acquired from specific data sources. Data transportation transfers the raw data to the storage infrastructure. Data pre-processing reduces noise, redundancies and inconsistencies,

improves data quality and integrates data from different sources to improve analysis accuracy and reduce storage expense[19][103].

The data storage system is divided into hardware infrastructure and data storage methods or mechanism. The storage methods are classified into file systems, databases and programming models. File systems constitute the foundation of the applications notable here are Google's file system (GFS) and the Hadoop Distributed File System (HDFS). Database technology used for Big Data is mostly Not only SQL (NoSQL) since relational databases cannot meet the challenges on categories and scales that are accompanied by Big Data. The database programming model employed in Big Data has to support large-scale parallel programs. This improves the performance of NoSQL and mitigates performance differences compared to relational databases[19] [103].

Another important Big Data concept is the shift of the data life cycle process. The common process steps consist of collection, preparation, analysis and action. Big Data shifts this process because of dataset characteristics and the time window for the end-to-end data life cycle. One example is the difference between the Extract-Load-Transform and the Extract-Transform-Load model. Dataset characteristics in Big Data applications with high volume need the data stored in a raw state before the data is cleaned and organised. This concept is described as schema-on-read where the preparation steps like cleaning, transformation and integration are performed when the data is read from storage[77].

Throughout this thesis the following definition of [77] will be used: *"Non-relational models, frequently referred to as NoSQL, refer to logical data models that do not follow relational algebra for the storage and manipulation of data"*[77]. The development and enhancement of information technologies lead to the generation of large amounts of data at a rapid rate. The efficient analysis of the data needs specialised analysis methods[19] [103] [52].

The goal of this thesis is to develop incident monitoring, detection, identification and classification process for Cloud-based Big Data analytics, with the primary function of processing vast amounts of IoT generated data. The first step here is the abstraction of the Big Data system to its core elements. The most fitting representation for this purpose is the generic framework from [83]. The possible set of data sources is restricted by the fixing the origin of data to IoT. This includes transportation and logistics, healthcare, smart environment, personal and social domain, wireless sensor networks, smart society, supply chain management and industry[89][16].

As soon as one element of the architecture is realised with a CS some stakeholders may change like the CSC, the CSP or the CSN. These roles are split into several other roles. It is important to notice that a party can occupy more than one role. For example a CSC can consume and offer cloud services and is therefore also a CSP for a different CSC [36].

The responsibility and the competence for the incident management process depends on the element of the Big Data architecture it is associated with. Every part of the Big Data architecture can be realised with cloud services. The Big Data Acquisition can be realised with protocols like AMQP and Java Message Service and tools like Storm, S4,

Kafka, Flume or Hadoop[17]. The Big Data Analysis can be realised with methods like web mining, data mining, stream mining, machine learning and tools like MapReduce, Microsoft Dryad, Greenplum and Hana[103][19]. The Data Storage can be realised with distributed file systems like HDFS or GFS, database systems like NoSQL or NewSQL and programming models like MapReduce or Dryad. The Big Data Usage includes MapReduce, Hive, Pig and Jaql for Hadoop-based approaches, Scope for Dryad and Dremel or Sazwall for GFS[17]. The list above gives an impression of the building blocks and their corresponding realisation via CSs.

The Big Data analysis process uses analytical tools or methods to inspect, transform and model data to extract value[52]. The possible methods differ regarding the latency requirements of the application. Data is characterised according to the time span it needs to be analysed, e.g. real-time, near real-time or batch. This leads to the distinction between stream processing (real-time or near real-time) and batch processing[74].

Figure 2.1: Quality of Analytics

Analytics stands for the discovery of meaningful pattens in data. It is used to refer to the methods, their implementations in tools, and results of the use of the tools as interpreted by the practitioner. Therefore Big Data couples analytics implementations design and data storage design[77]. Analytics from stream data processing are stream pattern matching, complex event processing, use of linked data, semantic approaches, entity summarisation, data abstraction based on ontologies, communication work-flow patterns and mobile analytics[52][22]. Analytics for batch based processing can be distinguished regarding the degree of structure of the data from the IoT devices such as structured, semi-structured or unstructured[52].

There are various possibilities to group Cloud-based Big Data analytics processes together in phases. The National Institute of Standards and Technology orders them in [78] regarding the value of the information. Demchenko et al. define a Big Data life-cycle [23]

that is embedded in a corresponding framework. Both offer possibilities for classification. Curry et al. describe a Big Data value chain in [22]. Kumaresan distinguishes [10] regarding platform layers and components. Another possibility is the distinction regarding the analytics workflow from [11]. Sharma's distinguishing factor are aspects of Big Data [95].

2.4 Stakeholder

The definition of a stakeholder is a person who has a stake in the success of the system[14]. But before identifying the relevant stakeholders the relevant parts of the system have to be identified[97]. Stakeholders in generic architectures are different because the application and design takes place in a broader, less-defined context which leads to a larger and less-defined stakeholder base[6]. It is important to identify the key stakeholders instead of generating an exhaustible list of possible stakeholders. The 4+1 model[60] is a well-known application-class framework that only has one dimension and exactly 4 views, logical, development, process and physical view. These views relate to different stakeholders and their concerns[48]. The four basic stakeholder classes users, developers, integrators and system engineers provide a very generic guidance regarding important stakeholders for a generic architecture.

Stakeholders for a generic architecture have to be identified. Basic and generic types of stakeholders are used as a starting point to identify the relevant stakeholders in the generic architecture. The next step, in defining the relevant stakeholders, is the identification of the underlying architecture and of the involved technologies and their interactions. It is important to notice that the roles are informative in nature, they are generalised and don't intend to be normative or imply any business or deployment model[78].

A generic architecture has to account for the interplay between Cloud Computing, IoT and Big Data. Therefore the reference frameworks are combined and reduced to the relevant stakeholders for incident monitoring, detection and classification. The data provider or data source is represented by the IoT devices. The deployment of the Big Data framework can either be directly on physical resources or on top of an Cloud Computing framework. In this case the framework is realised via a CSP.

2.4.1 Incident Management

"Incident management is the process responsible for managing the life-cycle of all incidents. Incident management ensures that normal service operation is restored as quickly as possible and the business impact is minimised"[65]. Incident management is a subsequent operation of event management. Event management monitors all events that occur through the Information Technology (IT) infrastructure, if the event is significant it may initiate anyone or a combination of incident-, problem- or change tasks. An event represents any change of state that is significant for the management of an IT service. In a general sense, an event is a discrete change of state or status of a system or device[86].

This implies that without the proper classification of an event no incident will be generated. ITIL defines an incident *"as an unplanned interruption to an IT service or reduction in the quality of an IT service or a failure of a Configuration Item that has not yet impacted an IT service"*[64]. The main difference between incidents, events and service requests is that events and service requests do not represent or are related to a disruption[64]. Following an approach from [87] the incident handling process is reduced to four main phases, preparation, detection and analysis, incident response and post-incident.

The incident management needs to be automatic and manual. The incoming data is filtered - inspected via stream processing for any known problems - and stored. The data residing in storage is used for analysis to identify incidents manual that the automatic monitoring is not yet aware of[101].

Monitored services and configuration items have to be measured if they are logging event data into the monitoring system. Records have to be analysed, the logged events have to be aggregated and filtered for incidents and the incidents have to be classified. Analyses are only feasible if they result in an action taken, the identified incidents have to be resolved and their root cause identified in the post-incident or post-mortem analysis[96]. A post-mortem is a description of an incident, its impact, the actions needed to mitigate or resolve it, the root cause and the actions undertaken to prevent the incident from reoccurring[90].

This thesis focuses on the identification of incidents in Big Data systems processing IoT data hosted on CSs over the identification of an disruptive or possible disruptive event. The preparation includes the definition of incident models. This includes time and location of the incident (When) and an incident description. The incident description provides information what failed (Where) , which areas are affected (What) and what caused the incident (How). The what, where, when and how are the context dimensions of the incident. The incident cause can be divided into the direct cause (the immediate event that resulted in an outage), the root cause (the key problem) and contributing factors[31] [20]. Root Cause Analysis (RCA) is beyond the scope of this thesis.

There are different ways incidents are created in the incident management system. The event system can automatically create incidents based on predefined rule sets that matches certain events and/or correlation of events to incidents. The rules are based on knowledge from already known event-incident-correlations. Stakeholders can create incidents manually. This only applies to incidents that are not known to the automatic correlation and filter system of the event management. Domain experts can develop new event-incident-correlations from the logged event data and add them to the event system for the automatic detection of future incidents[64].

Incident matching is the procedure that matches incident classification data against known problems, incidents, events and known errors. This procedure is necessary since many incidents are experienced regularly and appropriate resolution actions are well known. The incident matching process needs a known-error-database that can match incident classifications against known errors. This process can solve redundant incidents

effectively by finding resolutions quickly where possible[64].

Post-incident analysis and post-mortem culture is the final phase of the incident management process and starts after the incident has been resolved. Information and results from this phase deliver feedback for the improvement of the other phases. The collected information from other phases assists in learning and improving the incident management processes from a technical and managerial perspective[87]. Blameless post-mortem culture evaluates what happened, how effective was the response, what needs to be done different in the future and how to prevent this incident from happening in the future[90]. Some aspects of post-incident analysis is connected with problem management and RCA which is not the focus of this thesis. The important aspect here is that the processes of incident detection, classification and matching are influenced by the post-incident analysis in the regard that generated knowledge from post-incident is used to improve all these processes.

2.5 Software reference architecture

Architecture can be seen as the high-level structure of a system that describes fundamental aspects of the system and gives guidance to anyone who actually has to design and build the system. Architecture frameworks provide guidance but the sheer amount of available architectures complicate the search for the fitting architecture to a specific problem. Greefhorst et al. proposed a set of base dimensions as a distinction criteria between different architectures [48].

Bass et al. defined a reference architecture as a reference model mapped onto software elements that cooperatively implement the functionality of the reference model and the data that flows between them. They defined a reference model as the decomposition of a problem into parts that cooperatively solve the problem[13].

"A software reference architecture is a generic architecture for a class of system that is used as a foundation for the design of concrete architectures from this class"[6]. The generic nature of reference architectures leads to uncertainty regarding the architecture goal definition and design.

Concrete architecture has a specific context. It reflects the business goals of stakeholders. The identification of key architecture elements is well-studied and extensively published. A reference architecture facilitates system design and development for multiple projects. It designs applications in broader and less-defined context. The stakeholder base of a reference architecture is larger and less-defined.

Angelov et. al. propose a model that is based on three dimensions and corresponding sub-dimensions: the goal dimension, the context dimension and the design dimension[6]. A reference architecture balances these three dimensions.

The generic nature of a reference architecture makes them applicable in multiple, different contexts where every context represents a set of stakeholders and their requirements. The design of a generic architecture has to be on a high level of abstraction to guarantee its usage in different contexts[6].

Reference architectures offer a way to systematically reuse architectural knowledge when developing new software systems. They are the common denominator of a collection of similar systems. This collections always share a common technology domain, application domain or problem domain. The design of a reference architecture is usually done by capturing the essentials of existing architectures of a group of products and by taking into account future needs and variability[39].

Regarding the design strategy of a reference architecture there are two approaches, designing the reference architecture from scratch or designing it from existing architecture artefacts. This refers to the distinction of research-driven and practice-driven design of reference architectures[7]. Research-driven architectures provide a futuristic view of a class of systems while practice-driven builds on already existing architecture artefacts [39].

The type of generic architecture proposed in this thesis is based on a classical, standardisation architecture to be implemented in multiple organisations. The design strategy is practice-driven since there are available artefacts for Big Data systems, Cloud Computing, IoT and incident monitoring, detection and classification. The acquisition of data collects information about available artefacts. The construction of the basic structure of the generic architecture is derived from the collected information. The construction includes the documentation of the generic architecture in architecture views and consists of common building blocks like stakeholders, use cases and software component diagrams. Variability is guaranteed since the design of the generic architecture is on a high level of abstraction. The evaluation of the architecture is done with a proof of concept.

2.6 Taxonomies

"A taxonomy is a classification scheme that partitions a body of knowledge and defines the relationship of the pieces"[86]. Classification processes use taxonomies for separating and ordering.

Most of the references apply to the security incident management field, nonetheless the same criteria are applicable in any incident context if and only if they are generic and don't specify a concrete scenario[5] [71]. Since the most research in this area is done in the research field of security the methods and definitions postulated there are used to identify the basic requirements of taxonomies and subsequently classifications.

List of terms is a list of single defined terms. This is a very simple straightforward approach. This method tends to have problems regarding mutually exclusivity. An exhaustive list is unmanageable long and difficult to apply. Another shortcoming is that this kind of list ignores relationships between different types of incidents. Any type of incident taxonomy using list of terms usually is a product for a very specific case with a narrow focus[51].

Lists of categories are a variation of lists of terms by grouping several terms in broader categories to counter the problem regarding the unmanageable amount of individual

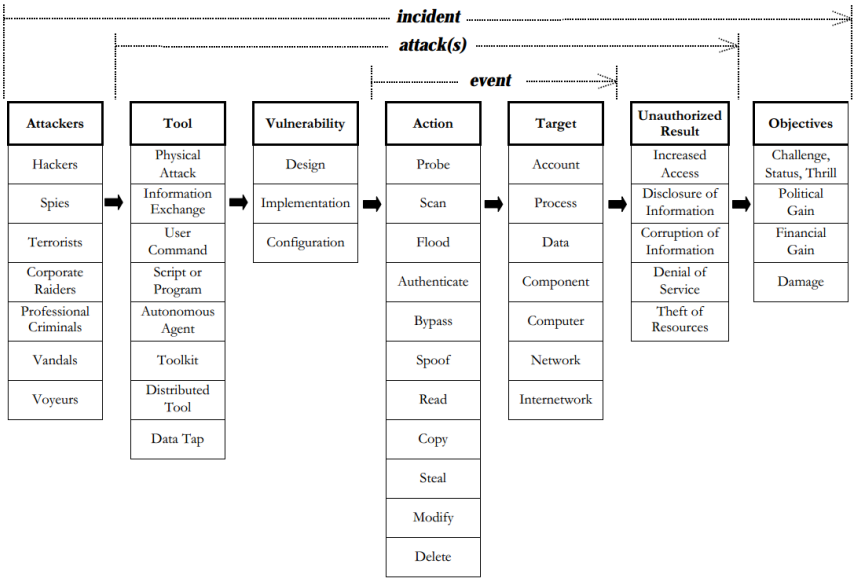

Figure 2.2: Computer and Network Incident Taxonomy taken from [51].

terms. The problem is that this list suffers from most of the same problems as one large list of terms[51].

Result categories group all incidents in basic categories that describe the result of the incident. An example is the incident of loosing one node in a cluster which leads to a significant reduction in the overall performance. The incident is classified regarding the corresponding result of the incident. This approach reduces the amount of categories but it limits the possible insight and information[51].

Matrixes classify based on dimensions. This allows categorisation of incidents with individual cells that represent a combination of different incident categories and incident types. One of the simplest ways is illustrated in [33] where the matrix has two dimensions. The additional dimensions in comparison with lists has advantages but the terms inside the matrix cells and the connections between them have to be intuitive and logic[51].

Elnagdy et al. [28] proposed a incident classification using ontology-based knowledge representation.

2.7 Monitoring

Monitoring is the basis of incident management. Monitoring forms the most basic layer of service reliability and incident management that enables more advanced elements including, incident detection, incident classification, incident response, "post-mortem" and RCA. The hierarchy of this elements is illustrated in Figure 2.3[90].

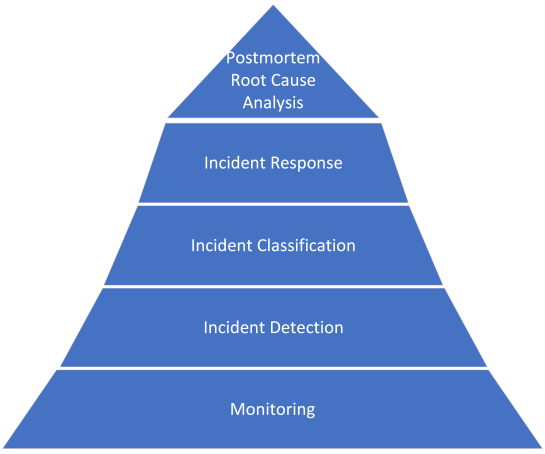

Figure 2.3: Incident Management Hierarchy taken from [90].

Incident detection and classification for services relies heavily on monitoring data collected at runtime. This includes service-level logs, performance counters, and machine/process/service-level events. In general various types of monitoring data has to be collected since each type of data usually only reflects specific aspects and components of the monitored system. Lou et al. sorted the monitoring data for a software service in three main types[68]:

(i) Performance counters

(ii) Events from the underlying operating system

(iii) Logs created by various components

There are different approaches when it comes to the identification of metrics that have to be monitored. Lou et al. [68] employ a data driven incident management that uses class association rules to identify incident beacons for monitoring. This method has been tested in large-scale online services at Microsoft™. The solution had two driving factors, first most incidents in the monitored system lasted less then 2 hours which led to over-fitting problems when using other algorithms. Second, when compared with classification-based techniques that order every incident in a single model, the false negative ratio was lower with the class association rules model[68].

Another viable approach regarding incident identification is complex event processing, where flowing information items are filtered and combined to form higher-level events. This model detects occurrences of particular patterns of low-level events, filters them and decides if they are relevant for the complex event processing systems[21].

Murphy et al. describe in detail the application Borgmon that monitors the large-scale cluster manager Borg[100] at Google™. Borgmon is a distributed time-series monitoring solution for the job scheduling infrastructure Borg from the year 2003. To facilitate

mass collection of required metrics, a standardised format was developed that enables instrumentation of the application. The generated time-series are stored in in-memory databases and regularly check pointed to disk. The rule evaluation is kept so simple that it practically resembles a programmable calculator[90].

CHAPTER 3

Use Cases and Challenges

3.1 Motivating Scenario

Let us consider a case of Base transceiver station (BTS)s providing wireless communication. BTSs consist of different equipment working together like transceiver, power amplifier, filter, low noise amplifier, duplexer, antenna system and power module. The BTS is usually equipped with alarm management that is capable of detecting equipment and environment alarm [98]. The various equipment produces data that is fed to an IoT gateway. For further processing, the data is forwarded to a message broker then the data is streamed into a data store with a non relational model and processed. Figure 3.1 shows a high level system architecture of a Big Data application processing IoT log data from BTS.

The BTSs form a large-scale, geo-distributed system providing network services to millions of users. The system delivers data to a Cloud-based Big Data analytics application for further processing consisting of the following parts:

(i) The data source such as IoT or data collectors that deliver the data assets

(ii) Data extraction Software such as message brokers, that deliver connectivity

(iii) Data loading and preprocessing that push the data assets into data stores

(iv) Data stores that save the data assets

(v) Data analysis structures

(vi) Interfacing and visualisation

The owner of the different parts of the system can vary. Analysis and/or visualisation of data is of interest for the data end user. The owner of the IoT, the message broker, the

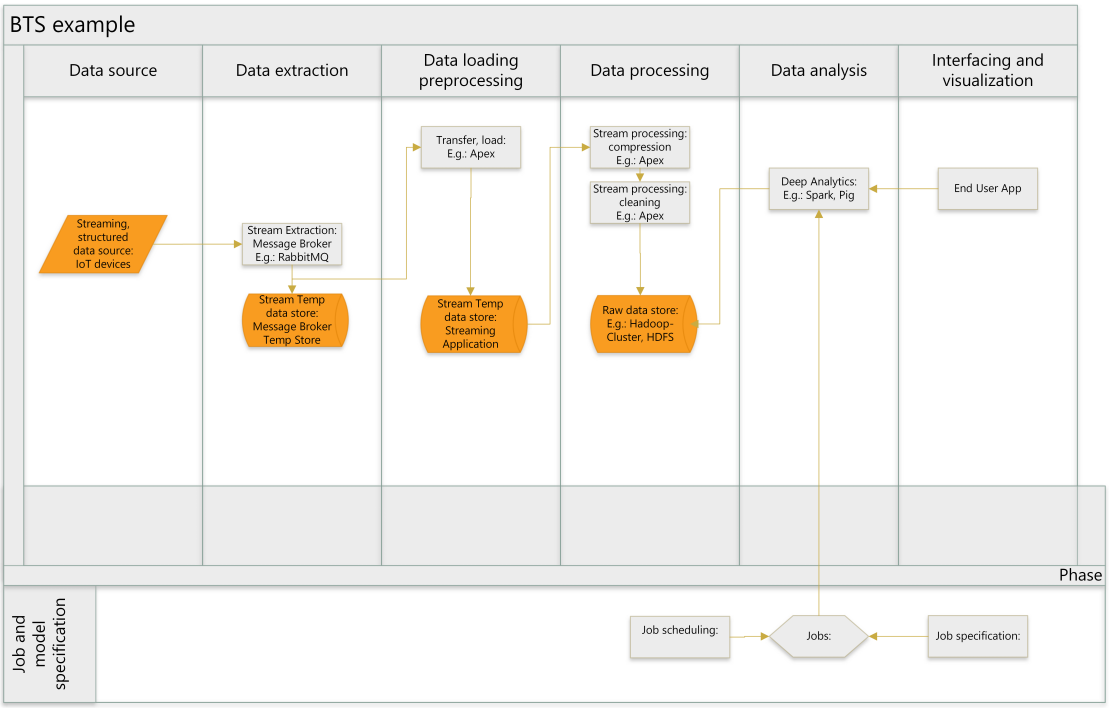

Figure 3.1: Base Transceiver Station Processing of Log Files.

processing platforms and the underlying physical infrastructure do not have to be the same.

The CSP role varies with the degree of outsourcing that in turn varies for different deployments. Every service can be realised by a CS. The incident classification uses the abstraction of the cloud based system so that there is no distinction between self hosted parts or CSs regarding the collection of monitoring data. Therefore the distinction is not between different deployment models of CSs (IaaS, PaaS,...) but instead it focuses only if CS can provide the necessary data for the analysis of an incident or not.

Access to data needed to identify an incident has to be provided for any service in the scenario by its owner. The owner of the system has to be able to identify any incidents as if the complete scenario is self hosted. The stakeholders of the CSs are superimposed with the roles of the cloud based system abstraction regarding the perspective of the incident analysis. For example it does not matter if an incident is caused by a role from the cloud based system abstraction or from a role of the CS that superimposes a cloud based system role. The basis of the analysis is provided by monitoring data from the services.

The loss of control of self hosted infrastructures due to usage of CSs makes monitoring cloud deployments of Big Data frameworks paramount. Since the CSC gives away control, the mechanism for monitoring and in succession for the collections of incident relevant

event data need to be mature. The CSP needs to deliver all necessary data to guarantee the same efficient incident management as a self hosted system.

There is no adhered standard regarding monitoring and incident management and this can lead to a famous Cloud Computing specific problem, vendor lock-in. CSP typically employ incident management systems and monitoring solutions that comply with the needs of their companies and without standards this can lead to vendor lock-in where the costs of switching to a different provider and creating the incident management process from scratch pose a significant obstacle. The various stakeholders of the scenario must have a common understanding over standard procedures regarding the mutual administered incident process.

Traditional incident management approaches do not cover all properties of Big Data systems processing IoT data, the amount of processed data, the geographical distribution and the complex interaction between the different system components pose problems. As a result the incident management takes place in the separate parts of the Big Data system isolated from each other whereas it has to view the system as a whole. This way it can identify root causes of incidents whilst ignoring ownership of singular components.

The combination of the different architecture models, the deployment models and the focus on incident monitoring, classification and analysis leads to the following stakeholders:

1. System Orchestrator orchestrates the system the data scientist is one actor of this stakeholder
2. Data provider creates data and pushes them into the Big Data application
3. Application provider provides algorithms and methodologies
4. Framework or Service Provider, CSP offering services, platforms, software and infrastructure
5. Data Consumer interfaces with the application and visualises deliverables

The system orchestrator integrates the Big Data applications into an operational vertical system. This stakeholder defines the requirements of the system including policy, governance, architecture, resources, and business requirements. The system orchestrator can be divided into actors like the data scientist and the architect. The actor data scientist for example is responsible for the selection of the data source and defining the requirements for data collection, storage, preparation, analysis and visualisation and the choice of the analytical model[79].

The data provider introduces new data into the Big Data system. The data provider stores captured data. The data in storage waits for further processing. Mapped to the motivating scenario, the data provider collects data from IoT and transfers it into message brokers.

The application provider executes the manipulations of the data life cycle meeting the requirements of the system orchestrator. The activities can be split up into different phases. Regarding the example this stakeholder establishes the mechanism that is necessary to meet the requirements postulated by the system orchestrator[79].

The framework provider offers services or resources for the Big Data application. Regarding the motivating example this is the owner of the CS that offers IaaS, PaaS and SaaS[79].

The data consumer receives the value output from the Big Data system. The data from the data provider gets enriched by the data application provider who is orchestrated by the system orchestrator. The data consumer searches, visualises and analyses the IoT processed data[79].

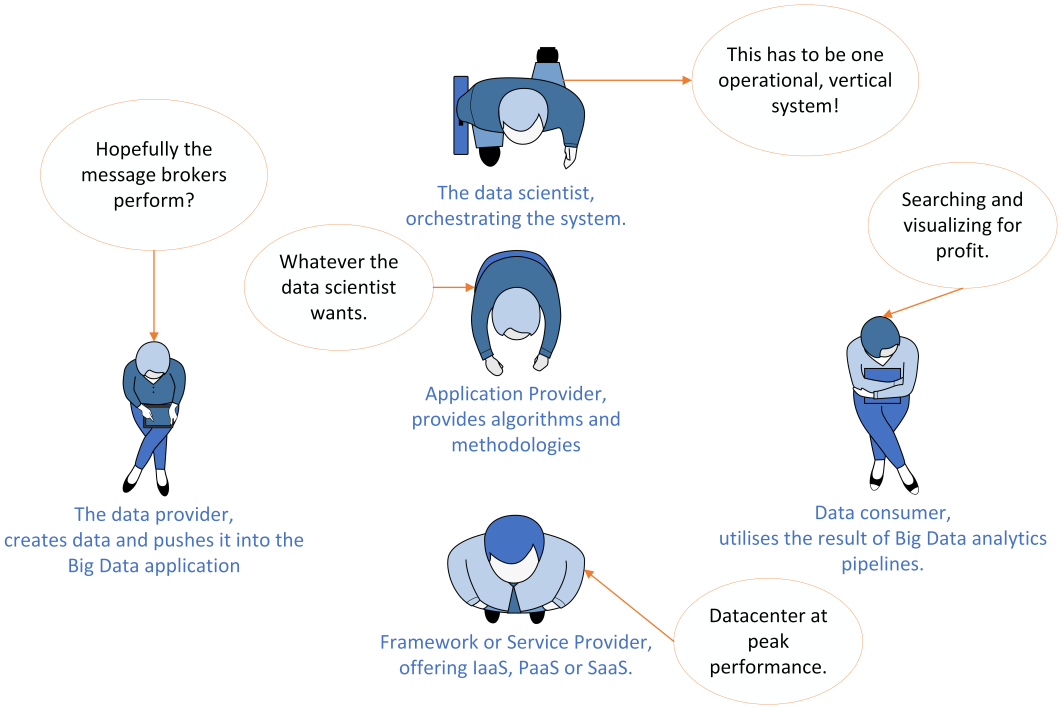

Figure 3.2: Stakeholders.

Big Data also extends the simple one-phased data pipeline pattern to a multiphase data pipeline where the output of one program becomes the input of the next. Any incident in one of the phases of the pipeline renders the whole output of the pipeline worthless. The design of a data pipeline refers to a program under the control of a periodic scheduling program. This kind of pipeline is generally stable as long as there are sufficient resources available. In the motivating scenario incidents have to be identified in both kinds of data pipelines. This includes jobs that exceed their run deadline, resource exhaustion, hanging processing chunks and trouble caused by uneven work distribution. The multiphase data

pipeline also has different requirements regarding monitoring. It needs to have real-time information instead of metrics reported only on completion[90].

3.2 Use Cases

3.2.1 Monitoring

Data analytics Monitoring

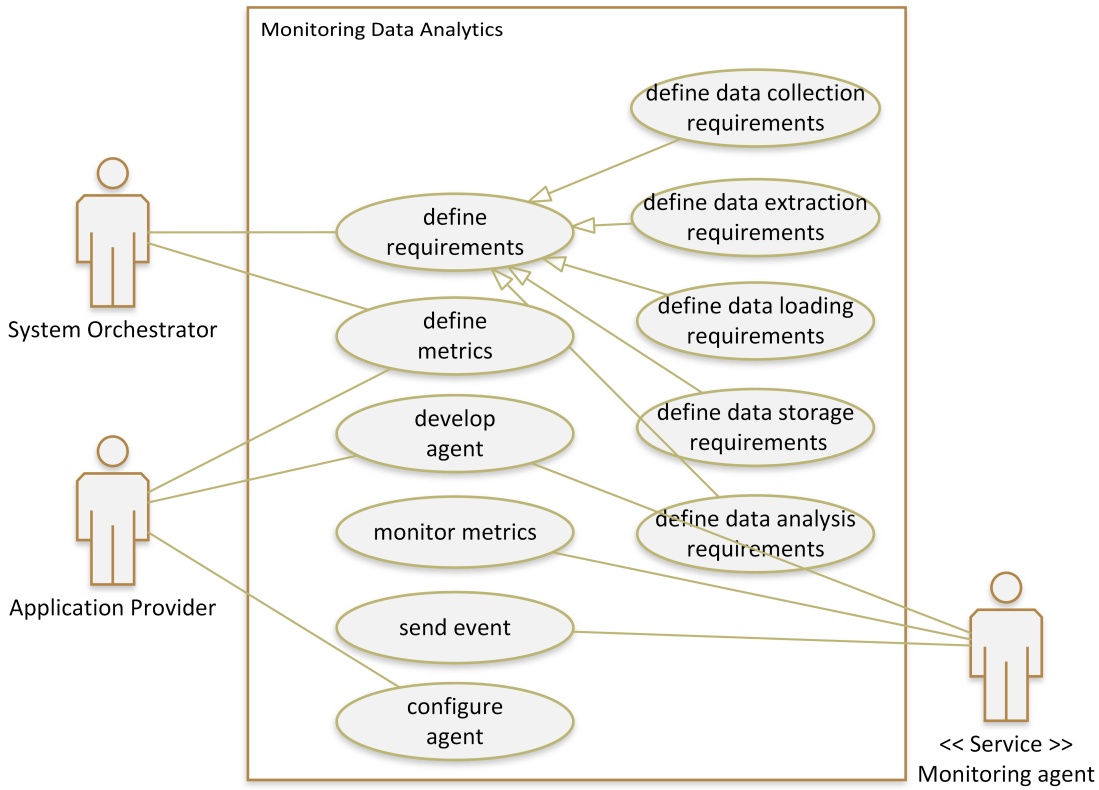

Figure 3.3: Use Case Monitoring Data Analytics.

Requirements:

1. The system orchestrator defines data analytics requirements
2. The application provider and the system orchestrator develop metrics that reflect on the beforehand defined requirements
3. The application provider builds monitoring agents corresponding to the defined metrics
4. The monitoring agents watch over the compliance of the defined metrics

5. If a metric is not met the monitoring agent sends an event

Without events there is no detection of incidents. It is impossible to anticipate all monitoring needs for a specific use case or domain beforehand. The monitoring layer has to be generic enough to accept new monitoring agents from the application provider. This monitoring tools have to be implemented in a way that the detected events are stored with all the other events. This guarantees that every event can be processed in the incident system independent from the source of its creation.

An example scenario for this use case is the data scientist who needs to save a certain amount of merged data logs per minute into the storage system. Therefore the data scientist defines metrics with the provider of the data extraction tool and the provider delivers a monitoring layer that monitors the amount of saved data logs per minute.

Service Level Agreement (SLA) monitoring

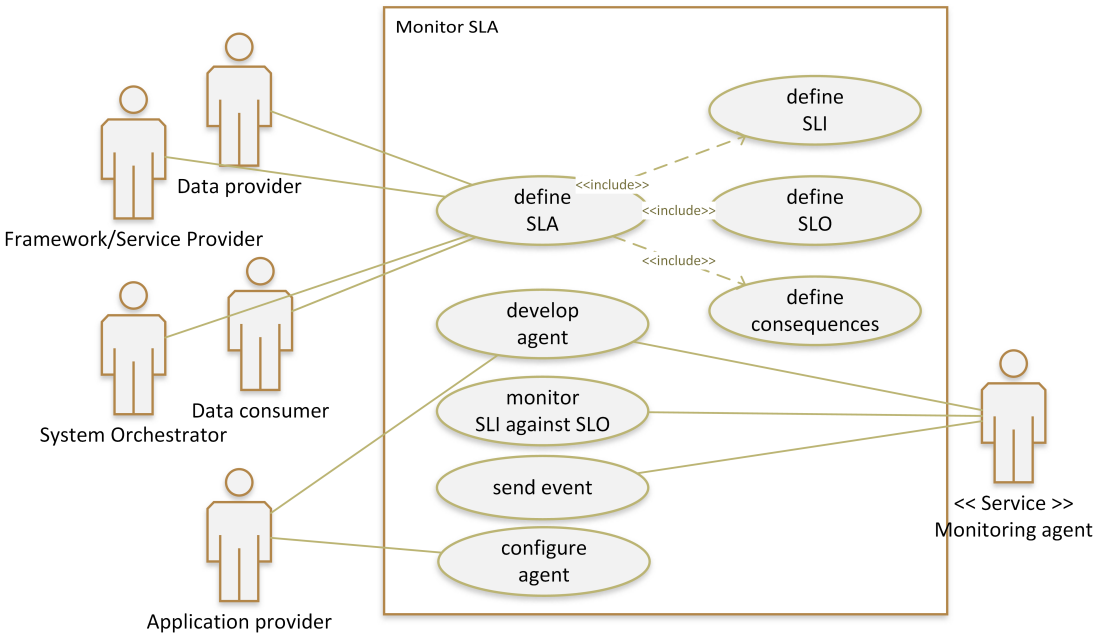

Figure 3.4: Use Case Monitoring SLA.

Requirements:

1. The system orchestrator, the data user and the data provider define SLAs including Service Level Indicator (SLI), Service Level Objective (SLO) and consequences if an SLO is not met

2. The application provider develops monitoring agents corresponding to the needs of the SLO

3. The monitoring agent watches over the compliance to the SLO of the SLI

4. If a metric is not met the monitoring agent sends an event

The incident system needs to support SLOs, SLIs and SLAs. SLO are a very important tool to define acceptable values for monitored qualities of the Big Data application and the CS. These values are managed directly by the stakeholders[90].

An example scenario for this use case is a data provider who has liabilities to the system orchestrator and the system orchestrator has liabilities to the data user. In the BTS scenario the extracted and merged data is subject to SLAs triggering consequences for the data provider if the data is highly unreliable and in consequence raises costs for the data scientist.

System Architecture monitoring

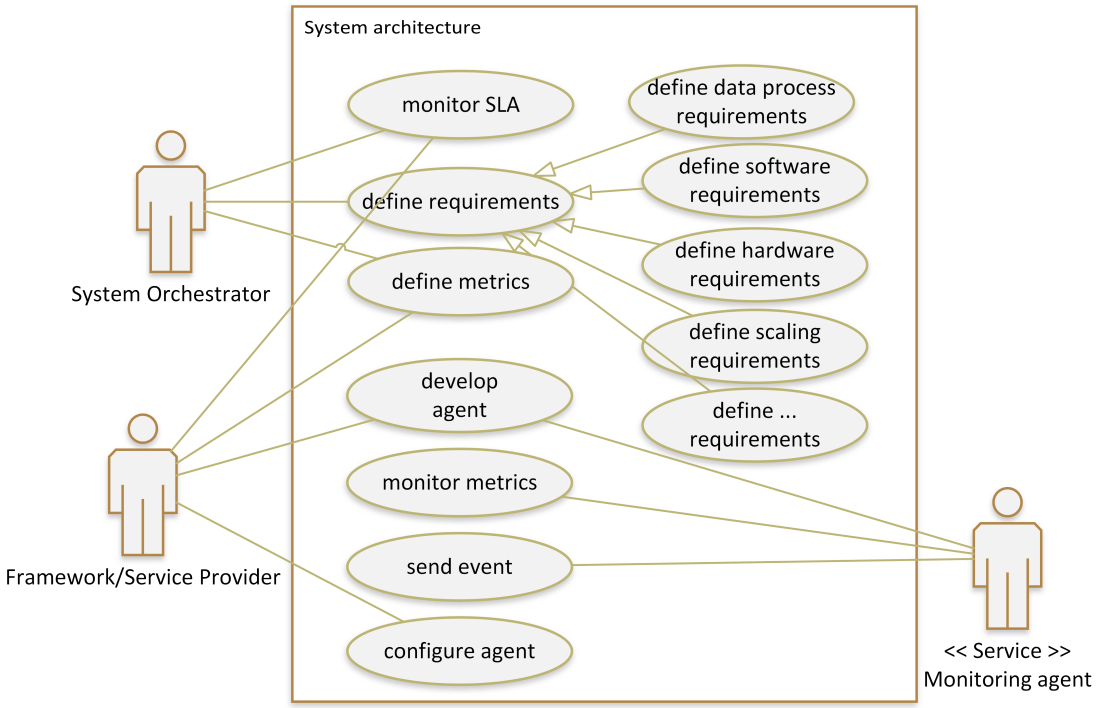

Figure 3.5: Use Case Monitoring System Architecture.

Requirements:

1. The system orchestrator defines system architecture requirements

2. The framework/service provider and the system orchestrator develop metrics that reflect on the beforehand defined requirements

3. The framework/service provider builds, supplies or configures monitoring agents corresponding to the defined metrics
4. The monitoring agent service watches over the compliance to the defined metrics
5. If a metric is not met the monitoring agent sends an event

An example scenario for this use case is a messaging infrastructure that is hosted on Amazon Web Services. The architects of the Big Data application specify requirements regarding the scaling of the messaging service for peak times. This requirements have to be monitored so that if Amazon Web Services fails to meet this metrics possible SLAs are used for consequences.

Multiphase Pipeline Monitoring

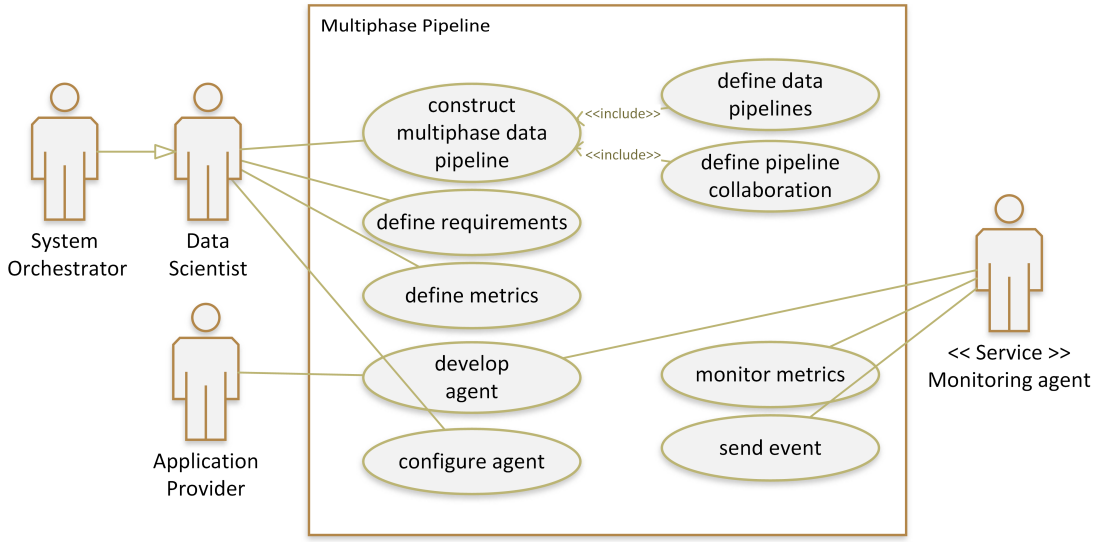

Figure 3.6: Use Case Monitoring Multiphase Pipeline.

Requirements:

1. Define data pipelines
2. Define collaboration between data pipelines
3. Define requirements regarding the collaboration of data pipelines
4. Define metrics that reflect the defined requirements
5. Develop an monitoring agent that is able to monitor the defined metrics
6. The Agent has to be flexible configurable since monitoring requirements change constantly

An example scenario for this use case is voluminous log data from the different systems that is extracted and merged together by the data provider, sent to Hadoop[49] and analysed. An error during the merging process leads to stored values that negatively influence the result of the analysis. Every data pipeline that contributes to the analysis has to be monitored in a way that enables RCA to determine where in the chain of data pipelines the incident happened.

Regulation Monitoring

This use case is similar to that of SLA monitoring. The SLA is a regulation, SLOs are necessary objectives to achieve regulatory compliance and the consequences are defined by the legislation of the regulator. The data provider stakeholder is the first stakeholder in the data value chain that has to comply with regulations. The system orchestrator stakeholder needs to monitor regulatory compliance throughout every data pipeline.

An example scenario for this use case is a data provider extracting and pushing data from BTSs. The data provider has to comply to regulations regarding data processing of telecommunication networks. The system orchestrator monitors the compliance of the data provider to all regulations from the state, regarding the processing of data from telecommunication networks.

3.2.2 Use Cases Detection

Automatic Incident Detection/Identification

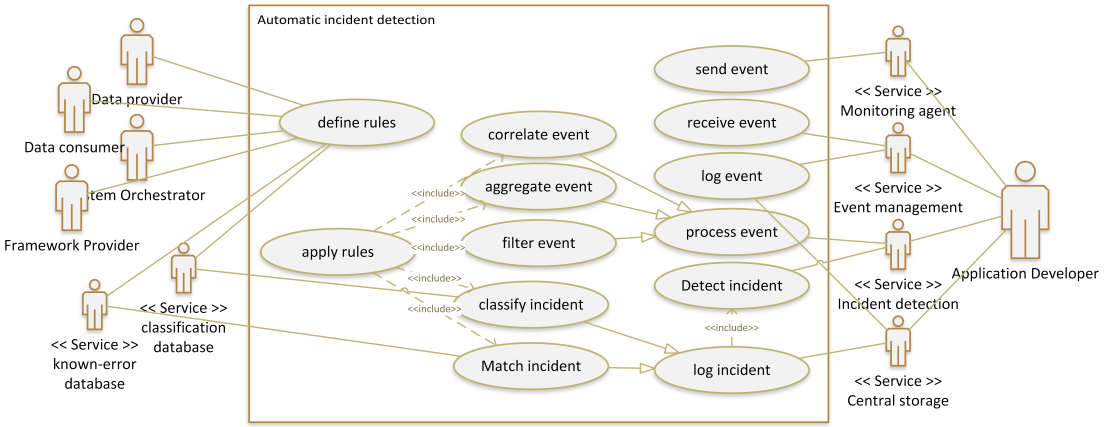

Figure 3.7: Use Case Detection Automatic.

Requirements:

1. The monitoring agents send events to the event management
2. The event management logs the event

3. The different stakeholders develop a rule set for incident detection that fulfils the different requirements
4. The incident detection aggregates, correlates, filters and logs events regarding defined rules
5. The incident detection service identifies incidents by analysing events
6. The incident detection service logs and classifies incidents
7. The incident detection service matches incidents against known errors
8. The application developer is responsible for developing the event management service and the monitoring agents

The automatic incident detection has to be extensible. There are various models to identify incidents from processing events. The stakeholders must have the possibility to add flexible new ways of incident detection to the system like complex event processing [69]. This applies to the automatic and the manual model in every stage of the processing of events regarding their properties and their possible escalation to incidents. Modules have to be added that incorporate different basic models of incident detection and identification.

An example scenario for this use case is a messaging service extracting log files from the IoT devices. It sends events via a monitoring agent directly to the event management. The event management system logs the event. The data scientist has defined that a specific combination of temperature readings from a specific IoT device has to be logged only if they occur several times in a row. The incident management filters the events corresponding to this rule. The incident detection system checks if the specified requirements are met and if it is a positive check an incident is classified and logged.

Manual Detection/Identification

Requirements:

1. A user of the incident management system can add incidents that got reported via different methods like web interface, telephone call or Email
2. A user can read logged events
3. A user can read logged incidents
4. A user can detect incidents, every detected incident has to be classified, logged and matched

There is a need for manual incident detection/identification. After the events have been processed and saved for automatic incident detection, the stakeholders can access the

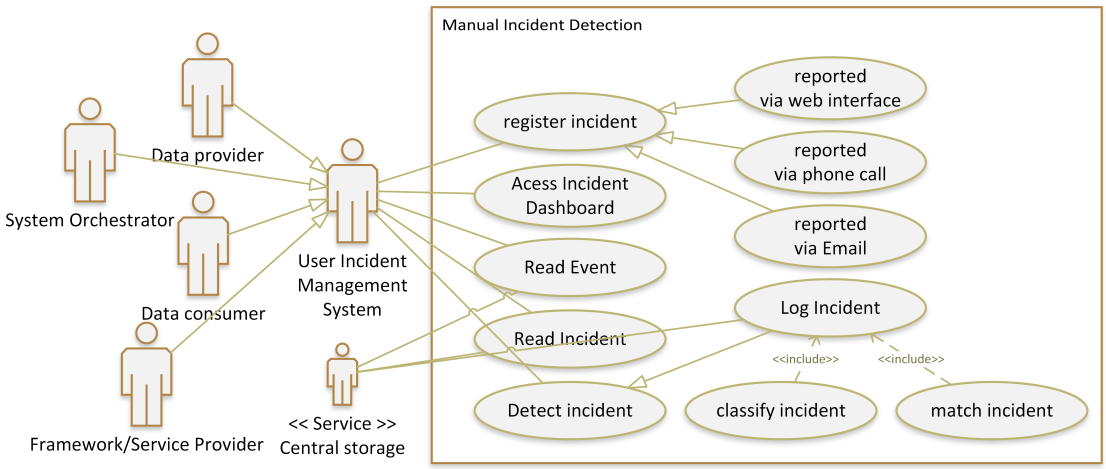

Figure 3.8: Use Case Detection Manual.

event and incident data to search for possible incident patterns that are not yet known. In succession the newly discovered incidents can be translated for the automatic incident detection process as illustrated in Figure 3.13.

Events and incidents have to be stored centrally. Since it is a system that is managed from different stakeholders every one has to have access in a reasonable way. Incidents and events are stored centrally. This enables the different stakeholders to see the events and incidents in the complete picture of the other systems. This guarantees that the whole system is monitored by one solution and not by an isolated application that can only act in a limited area. In complex systems the incident cause can be a combination of the failure of one or more systems and the involved systems are likely not all owned by one singular stakeholder.

An example scenario for this use case is a data scientist who accesses the incident management system and browses through the different logged incidents and events. The data scientist recognises specific events reporting Global Positioning System (GPS)-locations of IoT devices that do not make sense. The automatic incident management system has no rule regarding this specific case so the corresponding events did not trigger the automatic generation of an incident. The data scientist logs the corresponding event and classifies it as a data error.

Incident classification

- The different stakeholders need to define a body of knowledge regarding the specific classification needs of the Big Data application.
- Classification rules have to be defined that are able to automatically categorise incidents in classes.

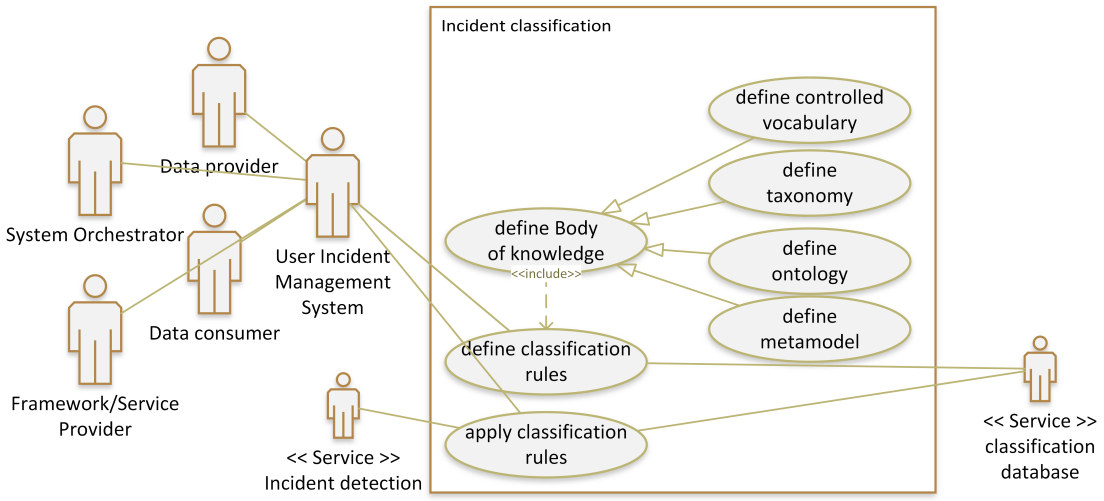

Figure 3.9: Use Case Detection Classification.

- The classification rules are applied by either the user of the incident management system in the manual use case and the incident detection service in the automatic use case.

The taxonomy is needed for the classification/categorisation of incidents. The deployed classification scheme needs to be precise regarding the structure and the possibility to add new rules to the taxonomy. The taxonomy needs to be simple with as few categories as possible. The taxonomy identifies the stakeholder that is responsible for the resolution of the incident. The taxonomy identifies where the incident stems from. Every domain has its own unique ontology. The IoT domains of multimedia, social media, operation of enterprises, trading of enterprises and scientific research have completely different ontologies and while some generic structure remains static most of the vocabulary describing, categorising and classifying incidents will change. The proposed taxonomy has to take into account that every stakeholder has to adapt the taxonomy corresponding to the ontology of the domain.

An example scenario for this use case are stakeholders involved in the Big Data application who have developed a taxonomy regarding the classification of incidents in their collectively provided Big Data service. An IoT device sends inconsistent temperature data. The incident is detected and the incident detection service classifies this incident as a data extraction incident before it is logged into storage.

Incident matching

- Classified incidents are automatically matched against a known-error database.
- The incident management user defines rules that decide if an incident matches a known error.

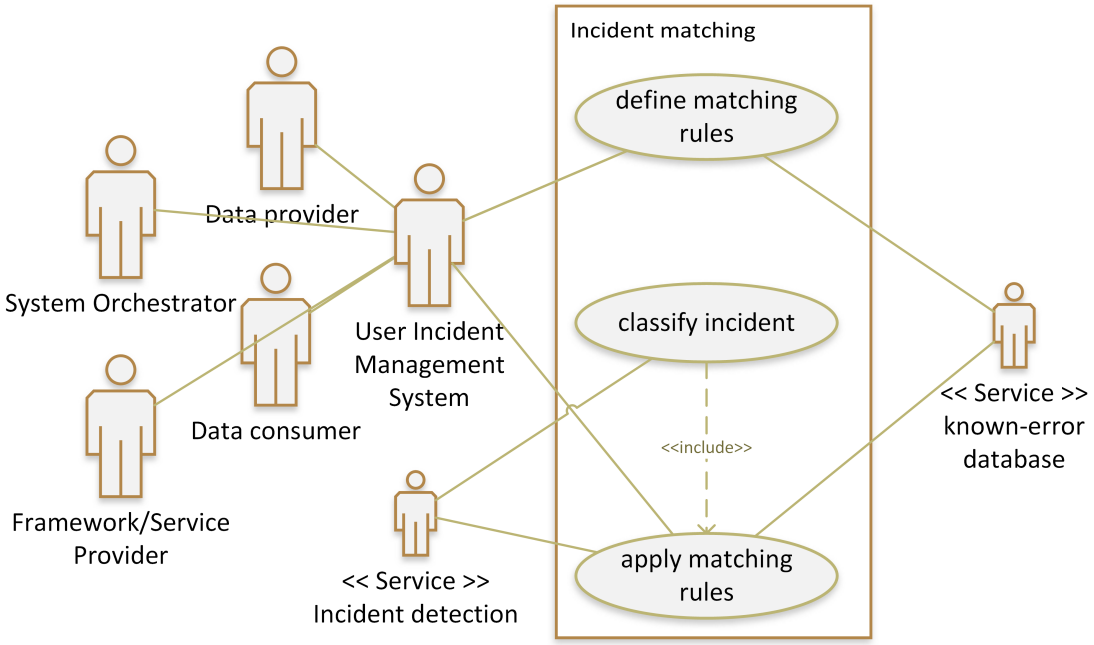

Figure 3.10: Use Case Detection Matching.

Incident classification data is matched against a known-error database to resolve redundant incidents efficiently. Any user of the incident management system is capable of solving a known incident that is sufficiently described in a corresponding knowledgebase entry. If an automatic resolve process is in place the identified incident can be solved automatically.

An example scenario for this use case is a user of the incident management system who gets a warning that an incident has happened in the cleaning process of a data asset. The incident management user opens the incident and sees that it has already been matched against the known error of a temperature reading out of bound from a very unreliable sensor in the network. The user already knows how to resolve this incident and starts to follow the prescribed steps from the knowledgebase entry regarding this specific incident.

3.2.3 Use Cases Analysis

Shared Incident Knowledge Base

- Every user of the incident knowledgebase has to write an entry that describes how the incident has been handled.
- Every user of the incident knowledgebase has the ability to read entries of resolved incidents

The detection of incidents needs knowledge about the monitored system. Since the knowledge is split between various stakeholders that may represent different companies

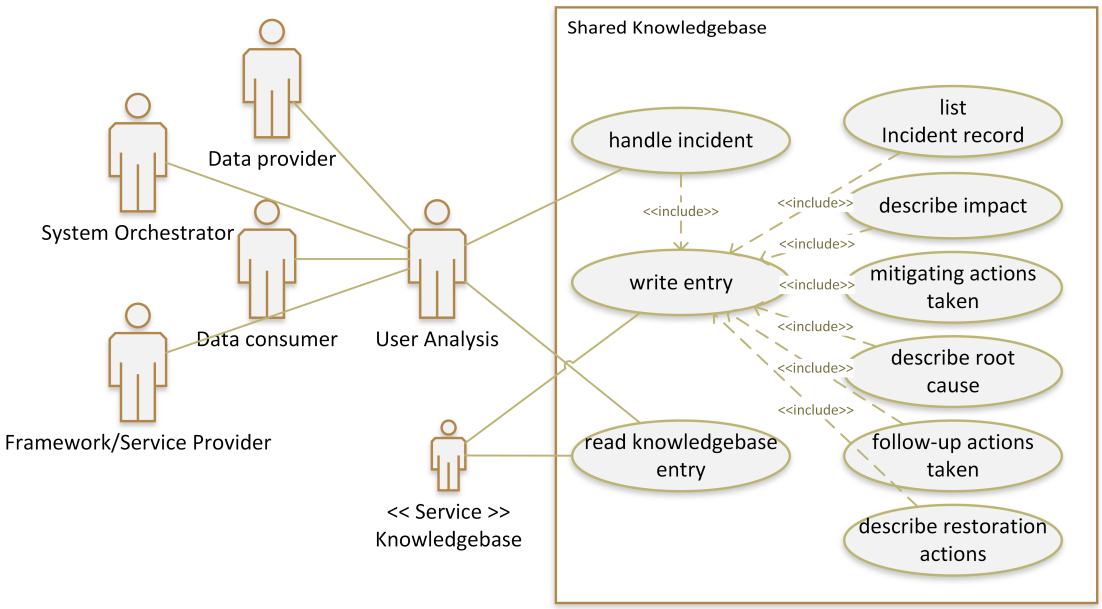

Figure 3.11: Use Case Shared Knowledgebase.

with different goals, it is imperative that the knowledge is well organised and well documented. There need to be a central mechanism to share knowledge learnt from past incidents[68].

An example scenario for this use case is an incident caused by a device sending GPS positions that are on a different continent. The data provider resolves the incident manually by exchanging the corresponding GPS module on the IoT device. After resolving the incident the data provider creates an entry in the knowledgebase and describes which events led to the identification of the malfunctioning device and how the root cause was identified. The data scientist who initially detected the wrong GPS entries reads the entry.

Reporting

- The different stakeholders use reporting to acquire an overview over incidents and events
- The reporting reads event and incident information from the central storage

An example scenario for this use case is a service provider of the distributed files systems who wants to know how often the system in the last 24 hours logged events regarding a specific computer in the cluster. The service provider generates a report and controls the output.

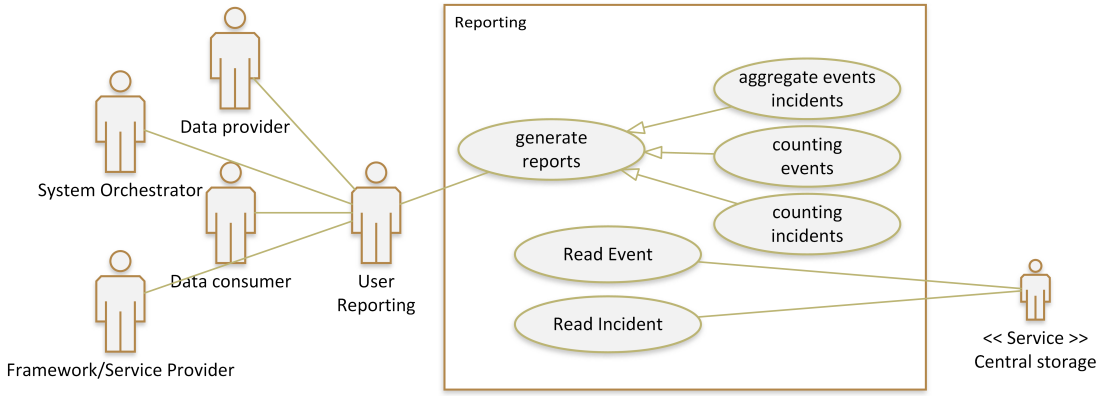

Figure 3.12: Use Case Reporting.

Post Incident

- The user analyses incidents after they have been resolved
- What happened
- What has to be done differently in the future
- How effective is the process
- How to prevent this incident in the future
- The outcome of the analysis is incorporated in the rule sets of the different use cases

Since the incident management process is always changing, it is not possibly to configure all necessary rules once and never change them in the future. Therefore the process generates feedback in the post-incident analyses (post-mortem or corrective and preventive action culture). The deliverables of this process are changes in the rule sets of the different services involved in the incident detection, classification and management process.

An example scenario for this use case is a data scientist who encounters an error in one of the results of the analysis process. The following investigation reveals an incident in the data processing software during the cleaning of the data assets. The data assets show impossible readings from sensors that until now are not identified in an event. The data scientist starts the process to expand the corresponding rule sets so that in the future this newly identified incident is automatically detected and reported during data cleaning.

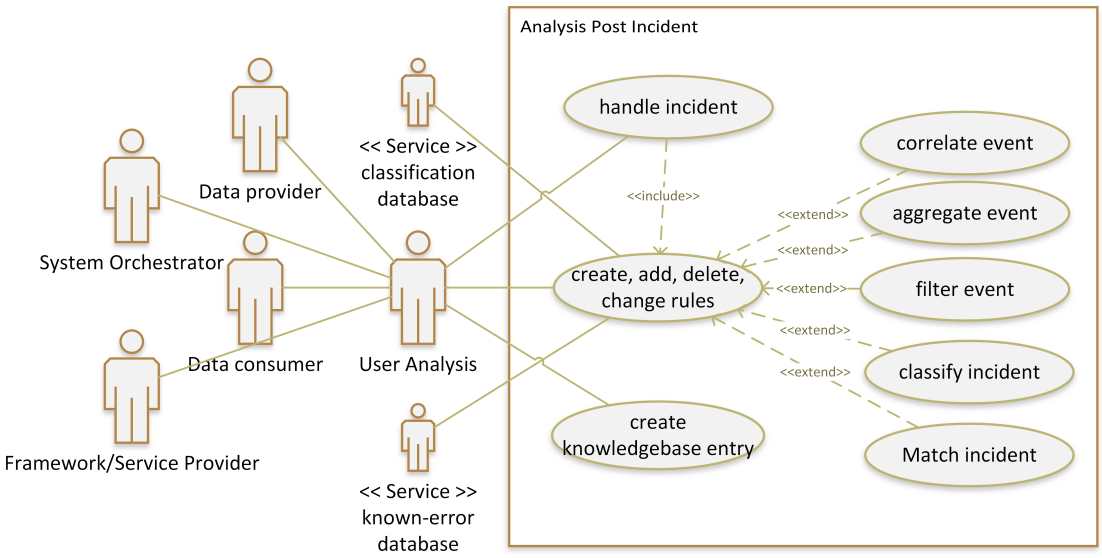

Figure 3.13: Use Case Post Incident.

CHAPTER 4

Incident Classification

4.1 Introduction

The classification process focusses on the automatic classification during incident creation. It is tempting to classify incidents at the end of the process but it is not recommended. The classification process has to be simple and reproducible to ensure that all incidents of the same type will be classified in the same way[33]. Multi-level incident categorisation describes an incident that is mapped to several categories in order to not loose any interesting information[64].

Incidents are classified according to a classification schema. The classification process determines the incident class with as much information as it can possible accumulate. This includes the person that reported the incident, data from monitoring systems, referring databases and other sources like relevant log files. The classification of incidents can at least take place at three different moments in the incident management process[64] during creation, during resolution or after resolution.

The incident classification scheme is recommended to be based on a taxonomy. A taxonomy leads to a systematic and regular incident management process. It produces meaningful statistics and delivers a common language regarding incidents for all stakeholders involved in the monitored system.

Taxonomies are established according to the team using it and specific to the system used by the team. The goal of this thesis is to develop an open taxonomy that can be easily expanded for the specific needs of different Big Data applications. A valid basic taxonomy consists of incident classes and types. The classes are practical and universal so that they do not get outdated easily. The incident types are part of the description of the incident.

The development of a taxonomy for a specific scenario works with the involved team to acquire meaningful but scenario specific incident classes and types. This is not possible

in the case of a generic architecture since the taxonomy has to be flexible enough to serve as many as possible types of Big Data applications. The taxonomy has to be expandable so that new incident categories can be added easily into the existing schema.

This leads to the following guideline to establish a stable, generic, long-term schema for classification. The taxonomy has to be simple with as few categories as needed. The categories are generic. Taxonomies that are already widely in use have to be considered during the creation of the taxonomy. Statistics about incident frequency can serve as an initial indicator[33]. The last consideration is how to treat an incident that does not fit into any category. There has to be a defined approach that is consequently used after the release of the taxonomy.

The driving element in Big Data are data assets. Data assets consist of grouped records and records consist of grouped data elements. Data elements are defined by their type and corresponding metadata. Characteristics of the data element are format, value, vocabulary, metadata, semantic, quality and veracity. A record is a group of data elements and is defined by format, complexity, volume, metadata and semantics. A dataset is a group of records with the characteristics quality and consistence. As an example if the data element is a temperature reading, the record is the temperature reading accompanied by a time stamp and the data asset is the whole transmission from a BTS device[79].

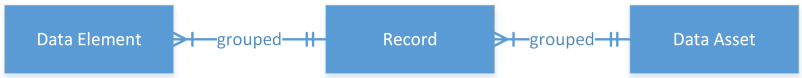

Figure 4.1: Data Asset

Data flows through the system comparable to a conveyor belt, there are several stations where the data is processed and changes its state. One process may change the data state and forward the data. Other processes may compress, transform or load the data. During every change of state of the data asset, there is the possibility to monitor events and detect and classify incidents accordingly. There is often the analogy that data is the new oil referring to Clive Humby an UK mathematician. *"Data is the new oil. It's valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals, etc. to create a valuable entity that drives profitable activity; so must data be broken down, analysed for it to have value."*[53] Therefore the focus of the incident process has to lie on the element it evolves around, data.

The smallest unit of interest regarding incident detection is the data pipeline. Every process in Big Data consists of data pipelines that consist of three processes:

1. Read data
2. Transform data
3. Output data

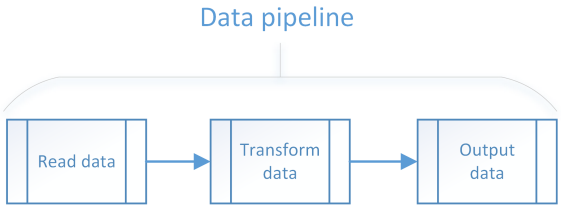

Figure 4.2: Data Pipeline

These core elements are part of state transitions of data assets and in all of them incidents can happen. This is illustrated in Figure 4.2. In Big Data pipelines are chained together to multiphase pipelines. The number of pipelines chained together is the depth of the pipeline illustrated in Figure 4.3.

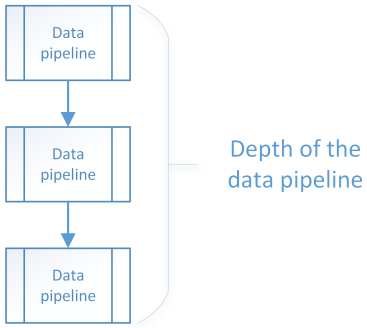

Figure 4.3: Data Pipeline Depth

The data pipeline concept can be mapped to the abstraction from Pääkkönen et. al. [83] with data flow, functionality and data source as the most basic elements. The interaction is illustrated in Figure 4.4.

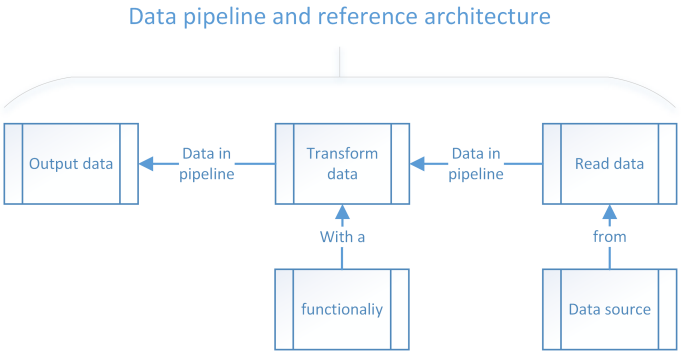

Figure 4.4: Steps in Data Pipeline

4.2 Survey of Incidents

4.2.1 Functionality Job Specification and Scheduling Incidents

The correct execution of jobs and tasks in the different phases is essential for the meaningfulness of the analysis. Since Big Data applications are based on several interdependent data pipelines the correct sequence of executing different pipelines is adamant. Failure of jobs and failure of tasks from failed jobs strain a significant amount of resources from clusters and are therefore an important source of resource efficiency, analytics time and cost[91]. This is relevant for data pipelines and multiphase data pipelines since the one represents the job (multiphase) and the other represents the tasks corresponding to a job (single data pipeline from a k-depth multiphase pipeline).

IC	What	Where	When	How
QoA	Job sequence is broken	Job specification - Job scheduling	Analysis phase	The jobs did complete in a specific order. If one job in the chain fails a notification has to be created. Example: One job in the sequence fails but the successive jobs execute ignoring the failed job and deliver false analytic results.
QoA	Job does not finish	Job specification - Job scheduling	Analysis phase	A defined job in the analysis process does not finish after a fixed time frame. Example: The analysis job is the successor of the cleaning job and the cleaning job does not finish therefore the analysis job can never start.
QoA	Non-idempotent job interrupted	Job specification - Job scheduling	Analysis phase	A non-idempotent job like data cleaning was interrupted. The repetition of the job leads to different results. Example: The cleaning of sensor data regarding a specific measurement stops halfway through, if the job is started again the already cleaned data is ruined.
QoA	Job scheduling	Data pipeline	Any phase	Unsuccessful job execution[92].

Table 4.1: Job Specification.

4.2.2 Typical Incidents in Contemporary CS

Typical incidents in contemporary CSs can propagate to incidents in the Cloud-based Big Data analytics application. Failures in the datacenter hosting a CS can negatively impact SLA with the CSC. The work in [12] divides the datacenter into IT infrastructure, power system and cooling system. The IT infrastructure is divided into processing, storage, networking and software. Any incidents in these elements have the possibility to disrupt the Big Data analysis application running on top of CSs that is deployed in the datacenter. According to Endo et. al. [30] the main reasons for outages are infrastructure or software failures, planning mistakes, human error, or external attacks.

IC	What	Where	When	How
QoS	Subsystem failure [94]	Datacenter	Any phase	IT infrastructure, power system and cooling system errors that negatively impact the availability of the CS that is dependent of this subsystems.
QoS	Software failures	Datacenter	Any phase	Failure in applications, operational systems, or hypervisors.
QoS	Planning mistakes or excessive datacenter usage	Datacenter	Any phase	Server and network-component performance bottlenecks, UPS overloading, and the production of too much heat.
QoS	Human intervention related	Datacenter	Any phase	External attacks and errors by an organisation's internal staff, such as a typo in code or a system misconfiguration.
QoS	Security related	Datacenter	Any phase	Data breaches, weak identity, credential and access management, insecure APIs, system and application vulnerabilities, account hijacking, malicious insiders, advanced persistent threats, data loss, insufficient due diligence, abuse of CSs, nefarious use of CSs, denial of service and shared technology vulnerabilities[4].

Table 4.2: CS Incidents.

4.2.3 Typical Incidents in Contemporary IoT

Typical incidents in contemporary IoT can propagate to incidents in the Cloud-based Big Data analytics application[56].

Incident Category	What	Where	When	How
QoS	Lack of resources	Data source	Data acquisition	The quality and cleanliness of data is negatively influenced when data collection policies consider trade-off for scarce resources.
QoS	Intermittent loss of connection	Network	Transmission of data	Sensors are only capable of transmitting small messages with a high ratio of packet loss due to scarce resources.
QoA	Sensor lack precision	Data source	Data acquisition	Every sensor has a precision class that defines how precise the measurements are. The sensor delivers measurements outside his precision class.
QoA	Sensor looses calibration	Data source	Data acquisition	The readings of the sensor are off by a specific margin.
QoA	Sensor has low accuracy	Data source	Data acquisition	The sensor readings are not accurate.
QoA	Inconsistencies in data sensing	Data source	Data acquisition	Faulty sensor delivers false readings.
QoS	Mechanical sensor failures	Data source	Data acquisition	Due to conditions like extreme heat or cold the sensor fails.
QoS	Sensor instability because of environment event	Data source	Data acquisition	Due to environmental events e.g. avalanche etc. the sensor fails.
QoA	Fail-dirty sensor	Data source	Data acquisition	The sensor fails but keeps reporting erroneous readings.
QoS	Stream processing	Data source	Data acquisition	The stream processing operator negatively affects the IoT devices by putting pressure on the scarce resources.

Table 4.3: IoT Incidents.

4.2.4 Functionality Data Storage Incidents

A simple distinction between the different data storage models encountered in Big Data is given in Figure 4.13.

IC	What	Where	When	How
QoS	Software component data store	Data Storage	Any phase	Performance related values such as latency, ineffective indexes, scale up, data level or scale out parallelism error, ineffective partitioning key
QoS	Software component data store	Data Storage	Any phase	Resource exhaustion connected to memory, network or storage space
QoA	Software component data store	Data Storage	Any phase	Functional errors concerning data persistence, data replication, data consistency, data durability, eventual consistency, timeline consistency
QoS	Software component data store	Data Storage	Any phase	Reliability and availability issues
QoS	Software component data store	Data Storage	Any phase	Relational storage properties such as atomicity, consistency, isolation, durability
QoA	Software component data store	Data Storage	Any phase	Data integrity issues

Table 4.4: Big Data Storage Incidents.

4.2.5 Data Extraction, Transformation, Loading and Preprocessing Incidents

Data acquisition, with the focus on gathering, filtering and cleaning of IoT data, is challenging. The data consists of a variety of objects and sensors that employ different methods for data representation and semantics[16]. The first component of this process are protocols that allow the collection of structured, semi-structured or unstructured information for IoT data sources. The second component contains frameworks that process the collected data and the third component consists of technologies that store the retrieved data persistently[22].

IC	What	Where	When	How
QoA	Data quality	Messaging communications platform	Data collection	The interfaces that collect data from the IoT devices lack necessary system resources like processing power. Not enough interfaces hosting protocols. Example: The BTS devices send so much data that the streaming application cannot process the data fast enough and the data pipeline gets clogged.
QoA	Analytics cost	Messaging communications platform	Data collection	The interfaces that collect data from the IoT reserve too much system resources. Example: There is an SLO between the framework provider and another stakeholder. Since the cloud system is highly elastic the framework provider needs to know when resources are not needed anymore.
QoA	Data quality	Processing platform	Data collection	Lost time and space correlation. IoT devices are placed at specific locations and IoT data has a time stamp, if one of these dimensions fail an incident is raised. Example: Incoming data from a BTS device has a time stamp that lies a month in the past.
QoA	Data quality	Processing platform	Data collection	An IoT device produces highly redundant data that wastes system resources especially storage capacities. Example: The data consumer is only interested in values that have changed since the last time they have been recorded.
QoS	Application	Messaging communications platform	Data collection	Something other than a valid sensor sends data to the data acquisition application. This false data can have a serious impact on data quality. Example: Somebody mimics the behaviour of a BTS device to poison the data storage with false data.

Table 4.5: Big Data Acquisition Incidents.

4.2.6 Quality of Analytics Incidents

QoA balances cost, performance and data quality. Typical incidents are illustrated in Table 4.6.

IC	What	Where	When	How
QoA	Response time too high.	Functionality	Any phase	The response time of a function (data ingestion, extraction, storage, processing, ...) is higher than previously defined as acceptable.
QoA	Scalability malfunction	Functionality, network, process	Any phase	A system, network or process is not able to handle the growing amount of work.
QoA	Elasticity malfunction	Functionality	Any phase	The system is not able to adapt to workload changes matching the current demand[50].
QoA	Data quality	Functionality	Any phase	This case is split into several incidents as illustrated in Table 4.7
QoA	Analytics cost too high	Functionality	Any phase	The cost of the analysis is higher than previously defined.
QoA	Analytics time too high	Functionality	Any phase	The time to analyse the data is higher than previously defined.
QoA	Resource efficiency	Functionality	Any phase	Essential for cost minimisation.

Table 4.6: Quality of Analytics.

4.2.7 Data Quality Incidents

Data quality is a critical requirement for the data consumer according to Karkouch et. al. [56]. The following incidents regarding data quality are notable in the context of IoT.

IC	What	Where	When	How
QoA	Accuracy	Functionality	Any phase	The maximum absolute systematic error α defines that specific values belong to the interval $[\nu - \alpha, \alpha - \nu]$. Example: One sensor from BTS reads temperature, the absolute accuracy error $\alpha = 4°$ values outside this range are errors.
QoA	Confidence	Functionality	Any phase	Real value with a confidence probability. Example: Sensor from BTS sends values and a confidence probability of $p = 99\%$ is defined. Example: On basis of the values of the sent data a confidence probability is defined. If the confidence probability is exceeded and incident is triggered.
QoA	Completeness	Functionality	Any phase	Example: Completeness criteria are defined for BTS sensors, if they fail to report data essential to completeness an incident occurs.
QoA	Data volume	Functionality	Any phase	Example: With respect to completeness a specified amount of data from the BTS sensors has to arrive within a defined time frame. If the received amount of data is lower than expected an incident is raised.
QoA	Timeliness	Functionality	Any phase	The difference between the current and the recorded time stamp. Example: For BTS sensor readings there is a defined maximum of the difference between current and recorded timestamps, if it is exceeded an incident is triggered.
QoA	Domain specific data quality	Functionality	Any phase	Every domain has specific data quality needs. The list here covers generic data quality incidents. If the generic architecture is mapped to a concrete example specific data quality incidents have to be added.

Table 4.7: Data Quality.

The necessity for data cleaning is linked with errors during data collection. IoT data collections are inherently inaccurate because of the limitations of the hardware[1]. Data cleaning identifies incorrect, inaccurate, incomplete, missing or unreasonable data and modifies or deletes it to improve data quality[18]. Data cleaning methods differ greatly regarding their field of application. Applications implement mechanism for data cleaning within their logic. This leads to a limited application-specific data cleaning post-process with increased development and deployment cost[55][56]. IoT has two points of contact with data cleaning. First, when it is seen as a live system and the cleaning happens in stream. Second, when collected sensor data is stored and analysed at a later point in time. This leads to different methodologies:

1. The more general method, define and determine error types, search and identify error instances, correct the uncovered errors[70]

2. The method for already stored data, explore the data set, detect possible problems, attempt to correct detected errors[102]

During data cleaning data formats, completeness, rationality and restriction are inspected[18]. A system for cleaning IoT sensor data consists of four major components: user interface, stream processing engine, anomaly detector and data storage. Popular models for data cleaning are regression models, probabilistic models and outlier-detection models[2].

4.2.8 Human Action Related Incidents

Incidents with root causes related to human errors illustrated in Table 4.8.

IC	What	Where	When	How
QoS	Software infrastructure	Where	Any phase	Misconfiguration. Example: Various jobs are queued and the next job is dependent on the outcome of its predecessor due to misconfiguration of the queue the analysis results are worthless.
QoS	Software	Where	Any phase	Programming error in an application. Example: An application has a bug that leads to a malfunction of a data processing functionality.

Table 4.8: Human Action Related.

4.3 Implementation of Incident Classification

Incidents are distinguished regarding the data processing functionality in which they occur. The abstraction from [83] gives the reference functionalities and functional areas that are used to distinguish incidents. The functionalities and functional areas that are used to classify incidents might vary regarding the scenario the classification is used in. The approach is to deliver the most granular hierarchy that is conceptual enough to give every possible domain the chance of a successful mapping. The functionality taxonomy is illustrated in Figure 4.7

In a distributed system where the responsibility is shared between different stakeholders quality is of the utmost importance. The quality domain regarding data processing applications is split between QoA and QoS. QoA includes data quality, analytics time and analytics cost. QoS includes performance and cost. The quality criteria are linked to the business view of the Big Data application. The instrumentation, monitoring and enforcement happen via SLO, SLA and SLI. Reduction of quality is an important issue in a distributed system where it forms an important basis of violations between different stakeholders. The owner of services and systems may differ from the application developer, the data scientist and the data user but all of them have to adhere to SLAs to provide an overall satisfactory result as illustrated in Figure 4.5.

Figure 4.5: Taxonomy Quality.

Another important classification aspect is the incident cause which can be categorised into technological, human action and natural phenomenon. The incidents can be caused by some natural phenomenon that negatively impacts big things like data centres or little things like IoT devices. Technological causes include incidents that are beyond the application developer's control i.e. bugs in platforms or infrastructure hardware failures. Incidents like programming flaws or misconfiguration are classified as human action as illustrated in Figure 4.6.

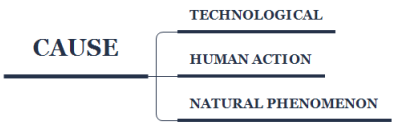

Figure 4.6: Taxonomy Cause.

One requirement of the incident management system is that every incident can be assigned to a stakeholder who is responsible for its resolution. Since the system is not owned

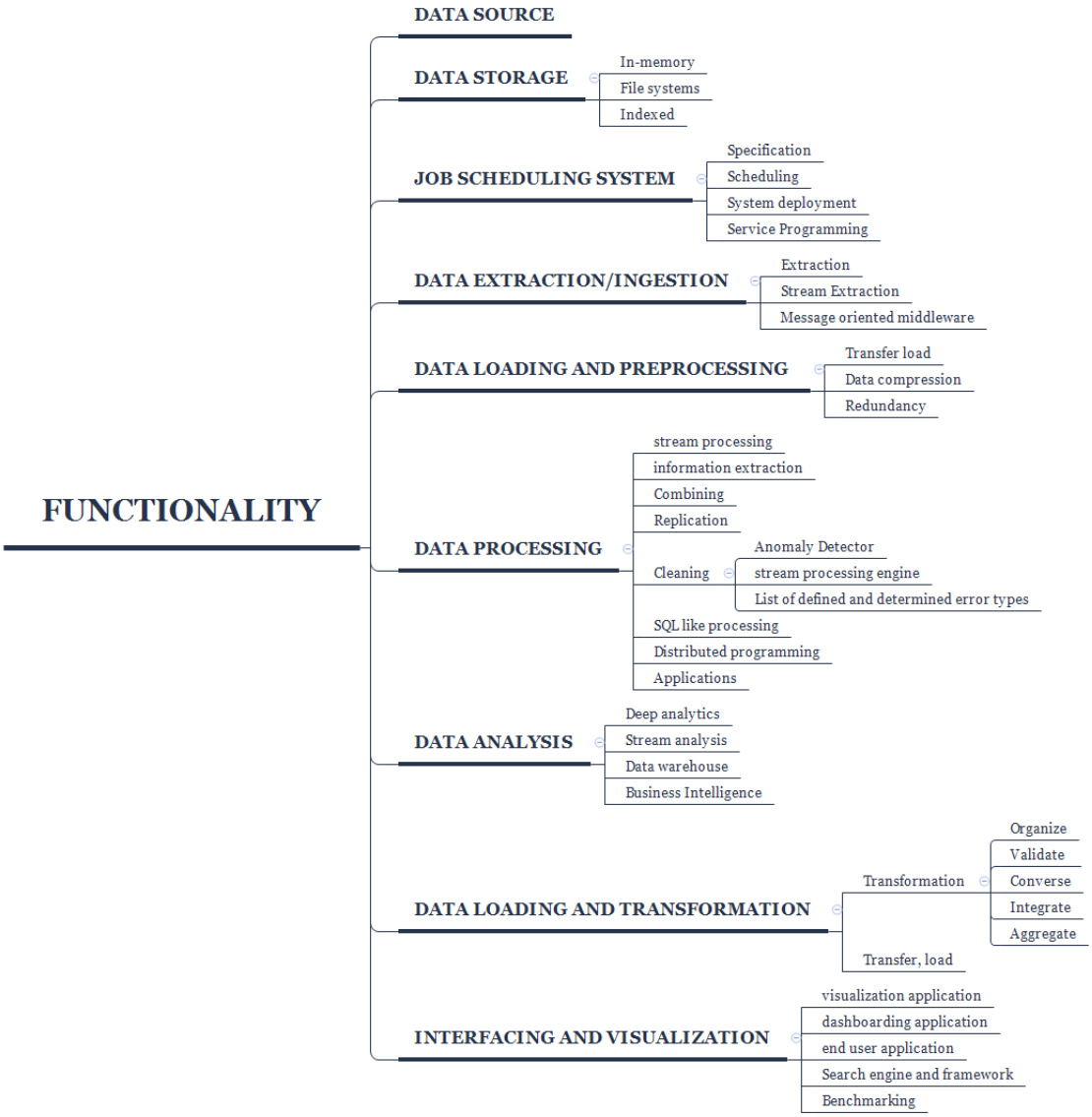

Figure 4.7: Taxonomy Functionality.

by one stakeholder alone, every incident has to have an owner. By overcoming the boundaries between the different stakeholders the overall functionality of the incident management system is improved. If the system is malfunctioning every stakeholder needs to know who is responsible for the incident and therefore responsible for the resolution of the incident. This adds the stakeholder taxonomy referenced in Figure 4.8 to the classification. Every stakeholder has various actors that deliver more granularity if needed. The list of stakeholders, their roles and their actors are based on the work of the National Institute of Standards and Technology[79].

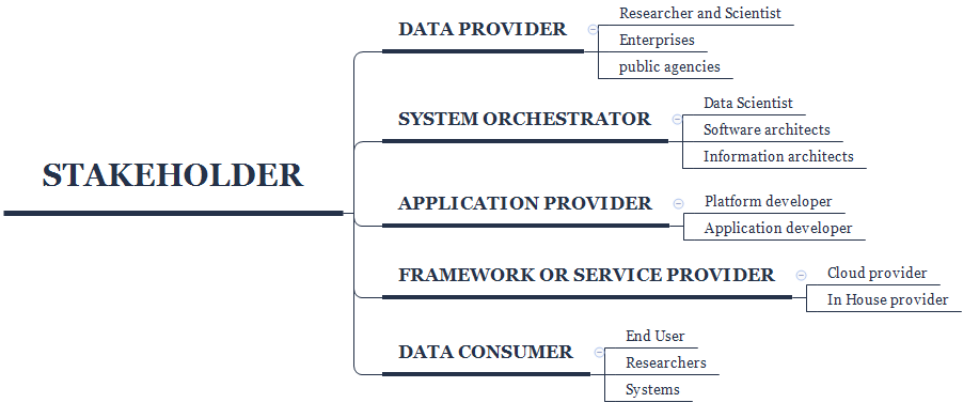

Figure 4.8: Taxonomy Stakeholder.

Data pipelines can be assigned to phases. Phases combine several data pipelines under one logical entity. The taxonomy has the focus on data assets and therefore it uses phases that describe the life-cycle of the data assets. The four phases used in the taxonomy are illustrated in Figure 4.9 with potential sub-categories.

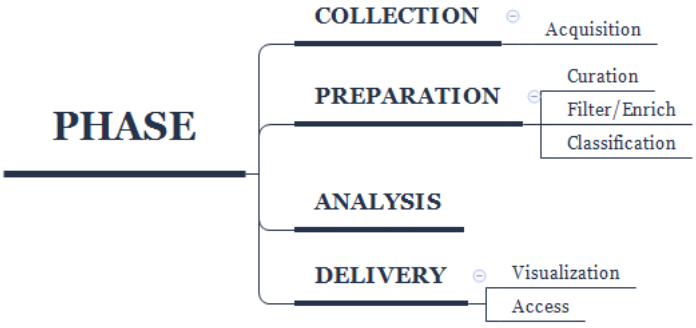

Figure 4.9: Taxonomy Phase.

ITIL introduces the concept of the configuration item as the unit that generates events and is therefore the base element of any incident. Configuration items typically include IT services, hardware, software, buildings, people and formal documentation such as process documentation and service level agreements [64]. Lou et al. define the term service-incident beacons which describes a small subset of system metrics that are symptoms pointing to the cause of the incident. Service-incident beacons are a combination of metrics with unusual values that produce a symptom[68]. This definition can be seen as a combination of configuration item and event detection. A service-incident beacon is the set of configuration items and the corresponding metrics that define if the configuration items work within specification. Since configuration items vary regarding

their respective domain the used terms in the taxonomy have to be generic enough to suffice the requirements of a generic architecture.

The locality of an incident in a distributed system is complicated because data processing functionalities are realised with software and software runs on system software and infrastructure. Systems are often realised with containers or virtualised components which adds to the problems regarding their classification. Figure 4.10 illustrates the different elements of locality.

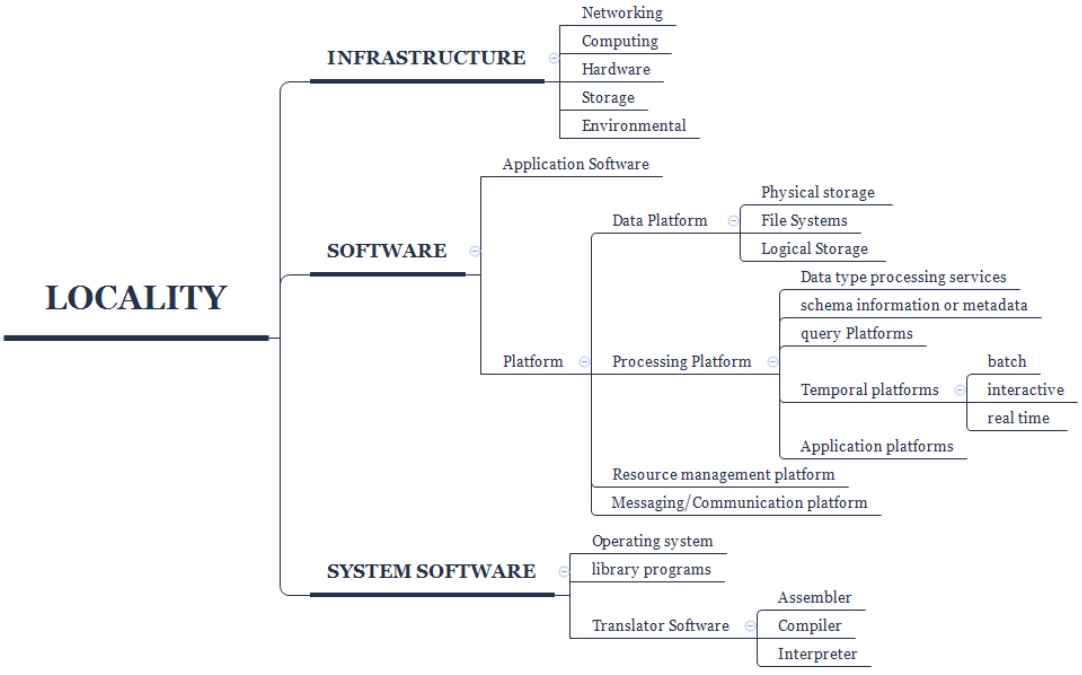

Figure 4.10: Taxonomy Locality.

The effect of an incident can be grouped into three categories regarding ITIL: the unplanned interruption of a service, the reduction of quality of a service and the failure of a system or service that has not yet impacted the application illustrated in Figure 4.11[64].

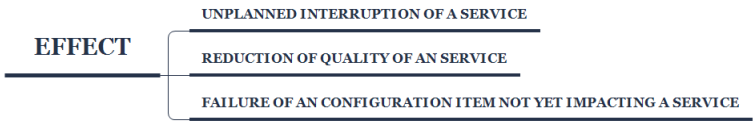

Figure 4.11: Taxonomy Effect.

The data states can be reduced to two states, data in processing and data at rest. Data at rest is the state where data is stored and not processed. In this state incidents like storage failures can happen. This state is also realised via the data storage functionality. The state in processing is every state where the data is processed by a functionality and therefore exposed to incidents stemming from applications, systems and so forth as illustrated in Figure 4.12.

Figure 4.12: Taxonomy Data State.

All the classification criteria and their relationships are illustrated in Figure 4.14. The classification is implemented in a Neo4J graph database that illustrates how the different elements are connected.

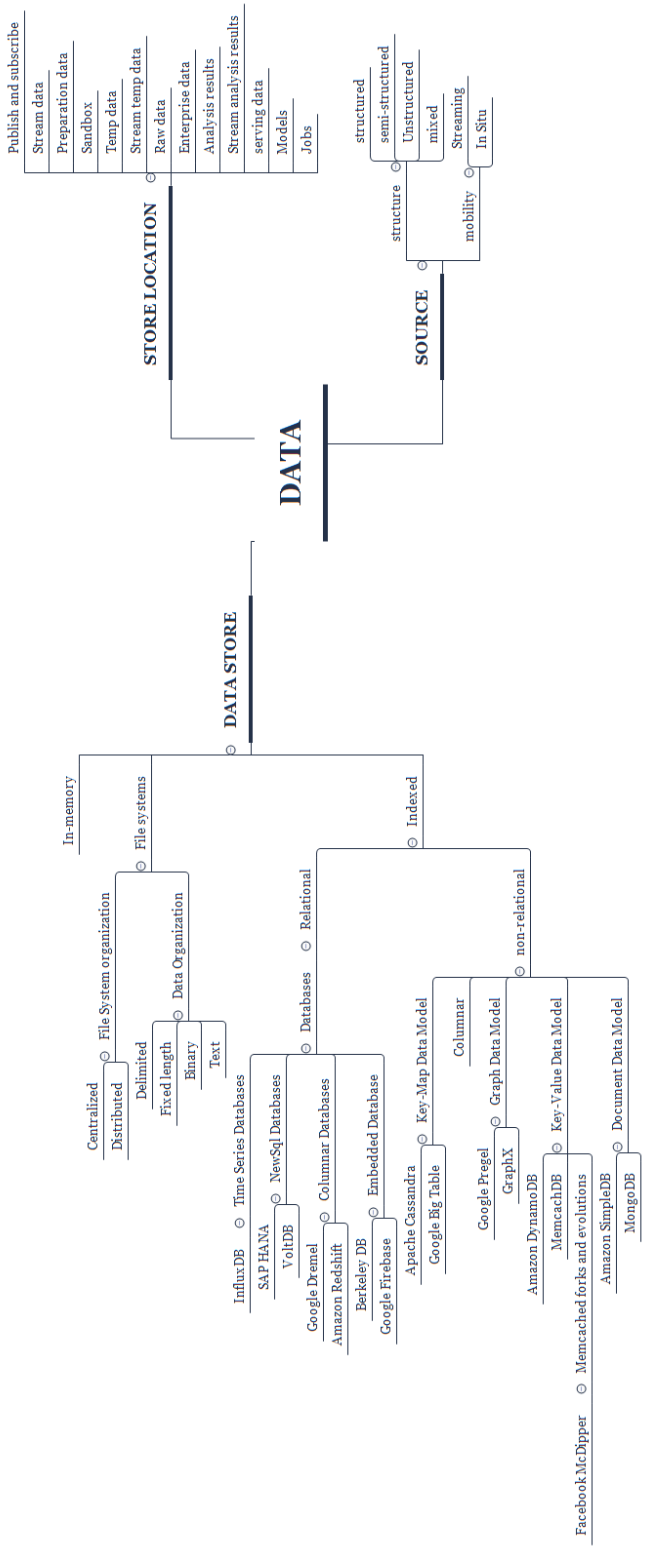

Figure 4.13: Taxonomy Data.

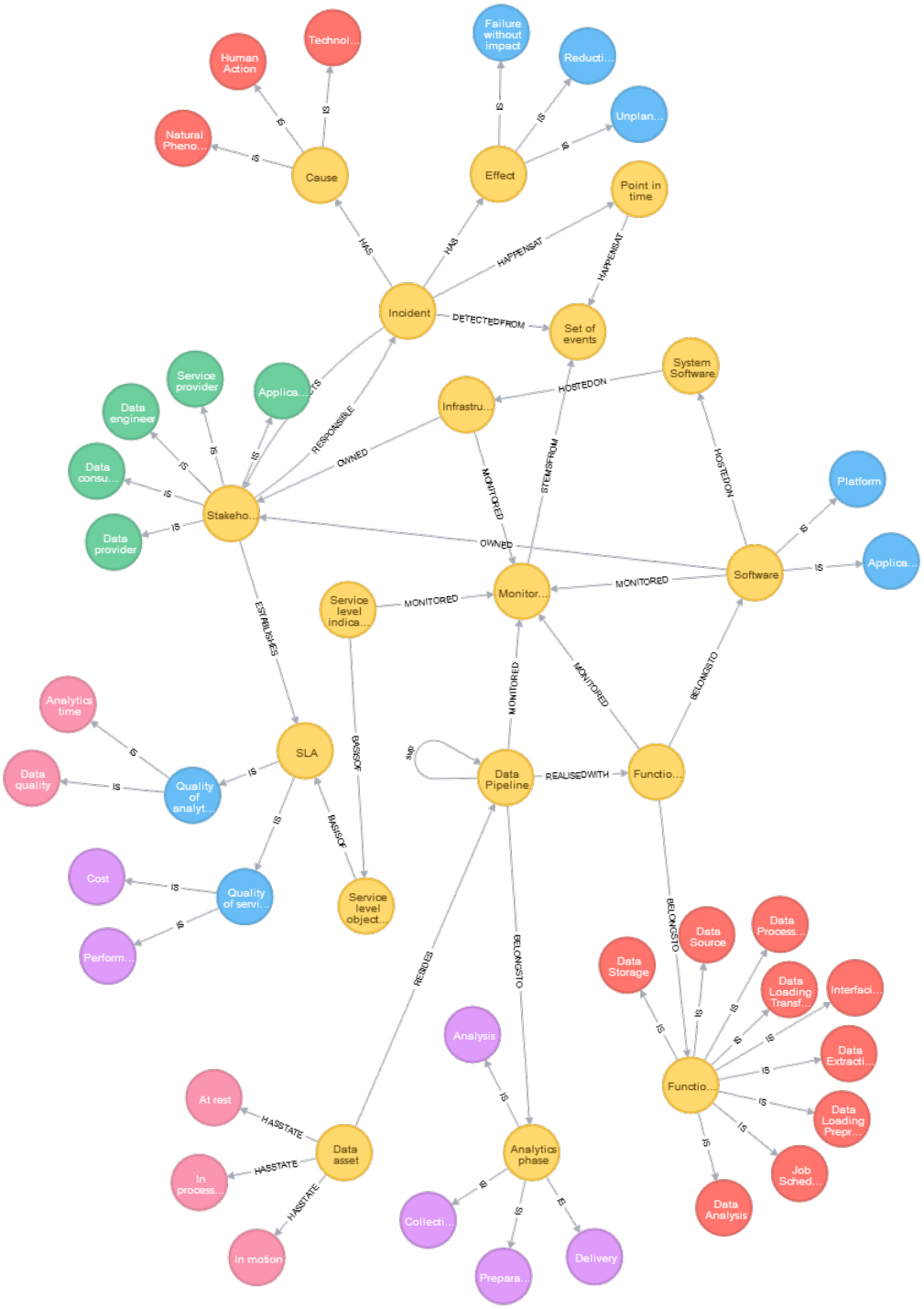

Figure 4.14: Classification Graph

4.4 Usage and Extensibility

4.4.1 Usage

The classification fulfils different purposes: First, it illustrates the key points of incident management in Cloud-based Big Data analytics applications. The classification of incidents regarding a complicated system with various aspects needs to be reduced to the very essentials. An architecture has to be generic so that the application of the architecture fits different purposes. The graph gives an excellent view of the different aspects that have to be taken in consideration when designing an incident management system for Cloud-based Big Data analytics applications.

Second, since the classification is already realised in a graph database it can be extended easily and used for any scenario for classifying incidents in Cloud-based Big Data analytics applications. The graph is in a form where the nodes are generic classes that have to be extended with specifics from the application that needs to be monitored. Therefore an implementation of the graph can be reduced to the nodes, edges and properties that are needed for a specific scenario. For example a specific data processing functionality adds nodes, edges and properties to refine the generic structure of the classification.

Third, the graph provides a reduction of a generic Cloud-based Big Data analytics application. It illustrates the inner workings of the application.

The first step in using the classification is the analysis of the Cloud-based Big Data analytics application. The process has to be understood and the different data pipelines have to be ordered regarding their data processing functionality. This is done by assigning the different used services to the corresponding data processing functionalities as illustrated in Figure 3.1. The different steps of the data pipeline are classified regarding their data processing functionalities.

Every data pipeline is realised with a functionality, every functionality runs on software that again runs on system software which is hosted on infrastructure. If a data asset is currently not part of a data pipeline, it is at rest and uses the data storage functionality. The different data pipelines are classified regarding the phase they are assigned to. This means that an application like a messaging service that is assigned to the data collection phase has to be associated with this phase to successfully classify it.

The software implementing the data processing functionality and the supporting software and infrastructure are owned by stakeholders. This guarantees that for every part involved in the Big Data application a responsible stakeholder can be identified.

4.4.2 Extensibility

If the generic classification graph is not sufficient for a concrete problem, the graph can be expanded to the individual needs. Since the model here represents a generic architecture the classes represent the smallest denominator regarding the Cloud-based Big Data application. This means that every implementation of the generic architecture

has to add the relevant characteristics to these basic classes. This includes but is not limited to:

(i) Specific functionalities, stakeholders and configuration items

(ii) Quality goals

(iii) Fine grained incidents

The classes in the graph are main classes that can be expanded regarding the represented scenario. Every additional class has to be linked to a generic class. The architecture can be used to catch security incidents, processing incidents, violating SLOs, software incidents or hardware incidents on the operating level. The generic nature of the classification guarantees that a responsible person is found via the dimension of the stakeholder. The classification describes when a state change of a data asset led to an incident, where it happened and who is in charge for resolving the incident. These are the basic classification parameters that need to be captured to guarantee an incident management system that can be used by all involved parties similarly.

The taxonomies from Chapter 4.3 already extends the basic nodes of the generic model but these extensions are theoretical in nature and no concrete examples. Figure 6.6 shows possible nodes with common storage software from the Big Data application area. This illustrates how the graph can be extended to assign concrete software to corresponding classes of the generic architecture.

A table of value attribute lists accompany the nodes of the classification graph. Metrics for different parts of the Cloud-hosted Big Data application vary regarding their intended quality goals(QoA or QoS), monitored applications, systems and services. Developing a standard set of key value attribute pairs that fit any implementation of the generic architecture is futile. The following rule is applied on every scenario individually, only measure when recording, only record when analysing and only analyse when it is followed by action[96]. Instrumentation, monitoring and classification are a closed process that needs to be designed and implemented thoughtfully.

The value attribute list concentrates on the four golden signals latency, traffic, errors and saturation that deliver a decent coverage of a service and can be seen as starting points for any implementation of the generic architecture[90].

The following abbreviations are used in the value attribute lists. The abbreviation INT is a variable of the 32-bit integer data type. The abbreviation INT64 is a variable of the 64-bit integer data type. The abbreviation INT[] is an array of variables of the integer data type. The abbreviation STRING is a variable of the string data type. The abbreviation STRING[] is an array of variables of the string data type. The integer data type is used to store numbers and the string data type is used to store a sequence of characters.

Functionality Data Storage The values listed in Table 4.9 are an excerpt from the Google Cloud Platform metrics for BiqQuery, an enterprise data warehouse. Real world examples are better at illustrating the possibilities of value attribute combinations since they are not theoretical. Examples that are already implemented also do not have to be connected with a possible implementation and/or example. A gauge represents a value that can go up and down. A distribution also known as histogram, samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.

Attribute Name	Attribute Value
uploadedBytesBilled	Uploaded bytes billed per minute. INT64, Byte/min
executionTimes	Distribution of queries execution times. Distribution

Table 4.9: Functionality Data Storage

Functionality Data Extraction Metrics for messaging applications are illustrated in Table 4.10.

Attribute Name	Attribute Value
messagingRoundtrip	Start subscriber, start publisher, publish message, receive message. Seconds
queueMessagesTotal	Total number ready and unacknowledged messages in cluster. INT64

Table 4.10: Functionality Data Extraction

Infrastructure Metrics relevant for SLAs and provided by the framework and service provider are illustrated in Table 4.11.

Attribute Name	Attribute Value
processStarttimeSeconds	Start time of the process since unix epoch in seconds. seconds
processCPUSecondsTotal	Total user and system CPU time spent in seconds. seconds

Table 4.11: Infrastructure

System Software A functionality is a software and is executed atop of system software that is hosted on infrastructure. The system software delivers metrics from the infrastructure layer listed in Table 4.12.

Attribute Name	Attribute Value
CPUCoreThrottle	Number of times the CPU core has been throttled. INT64
CPUFreq	Current CPU thread frequency in Hertz.
uName	Exposes system information as provided by the uname system call. String

Table 4.12: System Software

Functionality Data Processing, Extraction, Transform, Load and Analysis Metrics of data processing functionalities with data in the state of data-in-pipeline are listed in Table 4.13.

Attribute Name	Attribute Value
workersExecutor	Size of the workers executors. Gauge[INT]
memUsedMb	Memory used by the worker in MB. Gauge[INT]
workers	Number of workers. Gauge[INT]
aliveWorkers	Number of alive workers. Gauge[INT]
threadpoolActiveTasks	Gauge for executor thread pool's actively executing task counts. Gauge[INT]
threadpoolCompleteTasks	Gauge for executor thread pool's approximate total number of tasks that have been completed. Gauge[INT]

Table 4.13: Processing Functionality

Job Scheduler Generic attributes regarding job schedulers are listed in Table 4.14.

Attribute Name	Attribute Value
appsSubmitted	The number of applications submitted. INT
appsCompleted	The number of applications completed . INT
appsPending	The number of applications pending . INT
appsRunning	The number of applications running. INT
appsFailed	The number of applications failed. INT

Table 4.14: Job Scheduler

Stakeholder Stakeholders are responsible for software, systems and services. This way every part of the Cloud-based Big Data analytics application is attributed to a stakeholder that in succession is reliable for the unobstructed operation of the parts the stakeholder is responsible for. The corresponding metrics are illustrated in Table 4.15

Attribute Name	Attribute Value
stakeholderSoftware	Array of software elements this stakeholder is responsible for. STRING[]
stakeholderInfrastructure	Infrastructure elements this stakeholder is responsible for. STRING[]
stakeholderService	Service elements this stakeholder is responsible for. STRING[]

Table 4.15: Stakeholder

Quality The QoS and QoA have to be defined over metrics that are measured and monitored in the Cloud-based Big Data analytics application. The defined parameters for quality are key elements for SLAs between stakeholders. The quality parameters refer to measurements from software and infrastructure and set acceptable limits for them. Several parameters are listed in Table 4.16.

Attribute Name	Attribute Value
availabilityStreamProcessing	Measured in percent per year. INT[]
accuracySensorData	Number of readings that are above the maximum absolute systematic error α in one month. INT
analysisTime	Time summed up for all applications involved in the data processing in a specific multiphase data pipeline in seconds. INT

Table 4.16: Quality

Incident The incident description is not dependent on specific functions used in the Big Data application. It describes the incident and references the contributing factors. While the incident document is relevant until the effects of the incident are mitigated the post-incident has the purpose to identify the root cause and deliver insights into the incident management process. Metrics for the incident description are listed in Table 4.17.

stakeholderResponsible	A unique stakeholder responsible for the resolution of the incident. STRING
stakeholderAffected	An array of stakeholders that is affected by the incident. STRING[]
pointInTime	A unix time stamp. INT
events	A set of events that are responsible for triggering the incident. INT[]
incidentTimeline	An array of time, user-name and text field that describes the resolution of the incident stepwise . Array[INT][STRING][STRING]
status	A field showing the status of the incident. STRING
cause	The root cause of the incident. STRING
effect	The effect of the incident. STRING

Table 4.17: Incident

CHAPTER 5

Incident Monitoring

5.1 Important Aspects of Incident Monitoring

The system described in the motivating scenario is distributed over several stakeholders and there are many potential causes for an incident. The data from various sources has to be correlated to obtain incidents[68].

Monitoring of incidents in Cloud-based Big Data analytics applications is a challenging task. Big Data systems pose a highly complex problem space with many potential causes that trigger incidents and therefore different types of monitoring data needs to be collected to properly classify and identify an incident. This leads to the challenge of large-volumes of irrelevant data that gets collected during the monitoring of Big Data applications. Each type of collected data only reflects certain aspects of the monitored system and the data has to be aggregated and filtered to detect an incident. Another big problem is the incomplete and disaggregated knowledge of incidents in Big Data applications. The monitored system is composed of many components that are provided by different stakeholders each with a set of experts regarding their own domain, the monitoring application has to be centralised and all stakeholders need to have access to the monitoring system to coordinate the incident resolution efforts[68].

Monitoring has to be as simple as possible and it is not to be combined with other aspects of inspecting complex systems such as system profiling, single process debugging, load testing or traffic inspection. The monitoring system has to be kept simple and clear if the need arises loosely coupling it to other inspecting systems is the better strategy. According to Richard et. al.[90]:

1. The rules identifying incidents have to be simple predictable and reliable.

2. Data collection, aggregation and alerting routines that are never used are removed.

3. Metrics that are collected but not used are candidates for removal.

The correct identification, aggregation, filtration and correlation of events form the basis of incident detection. The patterns that describe the set of events that identify an incident have to be defined. Information from different sources has to be correlated and relations between patterns has to be identified[73]. When designing rules, in general simpler and faster monitoring systems are preferable to systems that try to learn thresholds or automatically detect causality. We do not use complex dependency hierarchies because they have limited success rates. Therefore monitoring systems and their employed rule sets have to be kept simple and comprehensible for every stakeholder. In a Cloud-based Big Data analytics application, aggregation is the cornerstone of rule evaluation. An Aggregated set of time-series from the tasks in a job entails taking the sum to treat the job as a whole[90].

Measurement resolution has to be chosen appropriately, very frequent measurements yield good data sets but they are expensive to collect store and analyse. Measurements take their toll on the monitoring systems resources and on the resources of the monitored system itself. If set incorrectly measurements can lead to incidents regarding the performance[90].

We follow the guidelines of Murphy et al. and focus on the four golden signals of monitoring:

1. Latency, the time it takes to service a request
2. Traffic, high-level system specific metric that measures the demand placed on the monitored system
3. Errors, requests that fail explicitly, implicitly or by policy
4. Saturation, the level between wasteful and system degrading utilisation

Systems that are not entirely self hosted and therefore under the administration from different stakeholders rely on SLAs. SLAs define non-functional requirements of the services specified in QoA and QoS. Reduction of the quality of a service is one of the three major incident classes and it is necessary to monitor the adherence to the SLAs. SLAs include obligations, service pricing and penalties in the case of agreement violations[29]. The monitoring of SLA results from the definition of SLO and SLI. The availability of necessary measurement data for SLIs is not guaranteed SLI because the system is not entirely self hosted. The CSP needs to establish the means that make the monitoring of SLIs possible otherwise the other stakeholders cannot control the compliance to the defined SLA and thereby rendering them meaningless[29].

A monitoring solution has to offer interfaces on all important elements of the Cloud-based Big Data analytics application to enable the stakeholders to adapt the monitoring system to the specific needs of their unique application. In the following Chapter 5.2 the software components of our generic architecture are described.

5.2 Architecture

Relationship with the classification. The classification surveyed incidents and ordered them to classes. The classes describe stakeholders, locations, cause, effect, functional and non-functional parts, data states and phases. The connection between monitoring and classification is that monitored resources can only be classified if the connection between classifiers - stemming from monitoring - and incident classes - originating from the classification - is realised with a comprehensive rule-set. The metadata needed to classify an incident has to be gathered during monitoring. The classification also provides a plan for the parts of any Big Data application that has to be monitored. Data pipelines and multiphase data pipelines have to be monitored since they are the core of every Big Data application. The different elements contributing to the Big Data application have to mapped to the reference architecture from [83] to identify their functional area. The functional areas run on software, infrastructure or platforms that have to be identified and monitored. The different data refinement steps are divided in phases. The incident classification can only be done when the necessary information is available. The information is always collected and provided by the monitoring respectively the event management. The classification cannot exist without the monitoring and the monitoring needs the classification to identify important points to monitor. The classification illustrates places where incidents might occur and defines the metadata that is needed from the monitoring to apply classification rules.

Points of data collection. The life-cycle of a data asset in every Big Data application can be split into different pipelines and grouped into multi phase data pipelines. Every single one of the data pipelines influences the analysis results. A Big Data application can be broken down into data pipelines, their connections and their relations. Since the data pipelines are the places where data is refined and value is added it is imperative to monitor not only the correct execution of the data pipeline illustrated in Figure 5.1 but also the interdependencies inside a multi phase data pipeline.

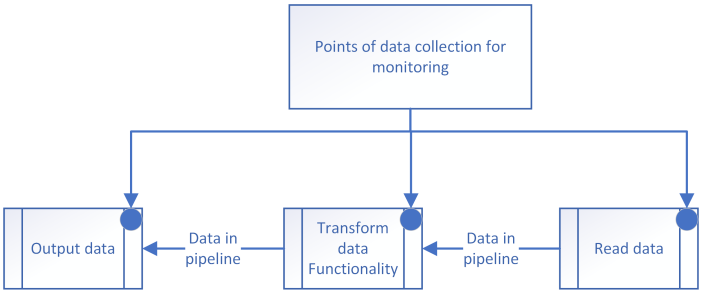

Figure 5.1: Monitoring Data Pipeline.

There is also a distinction between multi phase data pipelines that can be monitored partly as illustrated in Figure 5.3 and multi phase data pipelines where every part can be monitored independently as illustrated in Figure 5.1.

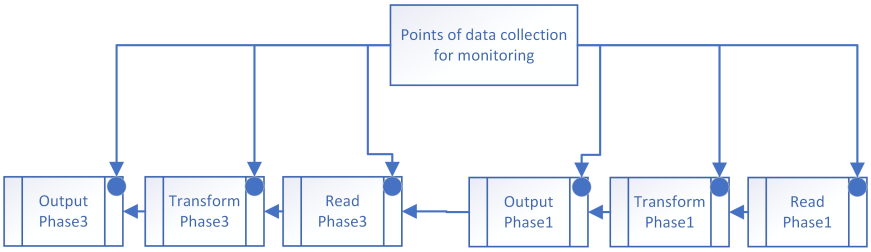

Figure 5.2: Monitoring Multi Phase Data Pipeline.

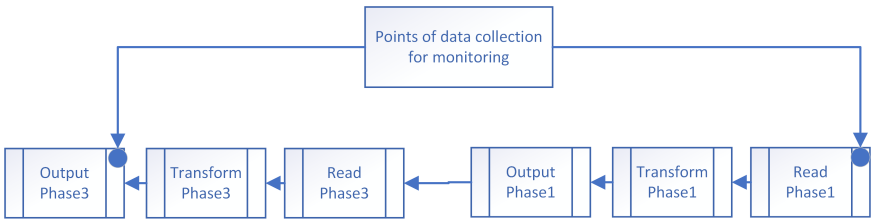

Figure 5.3: Monitoring Multi Phase Data Pipeline partly.

Data pipelines use data processing functionalities to create value out of data. This functionalities have to be identified and subsequently monitored. The set of functionalities are grouped into functional areas. A functionality enables the transformation of data and can happen several times in a single multiphase data pipeline. The functionality is the centre of the data pipeline. It is the next generalisation of the building blocks of the Cloud-based Big Data analytics application as illustrated in Figure 5.4. Data processing functionalities can now be combined to build different multi phase data pipelines to illustrate the various value adding data pipelines designed by the data scientist.

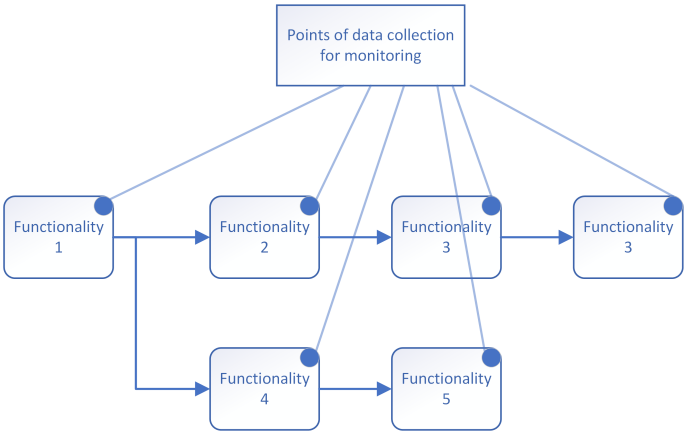

Figure 5.4: Monitoring Multi Phase Data Pipeline Functionalities.

Functionalities are realised with software and software runs on infrastructure and system software. Functionalities are classified in functional areas and multiple functionalities can

be realised on singular software components. Beside the monitoring of the value adding process the architecture that enables this processes has to be monitored. The software and the infrastructure that hosts and realises the Big Data application can - in case of a failure - lead to the reduction of service quality, unplanned interruption and/or failure without impact. Value is added with a data processing functionality carried out in a data pipeline, the functionality is realised with software and runs on infrastructure as illustrated in Figure 5.5. Any part in this chain that fails leads to incidents in the parts on top of it. For example if a virtual machine fails and not enough computation power is available to run a processing software within the defined parameters, this leads to a slowed function refining data and to an error in a multiphase pipeline that is monitored as a reduction in service quality.

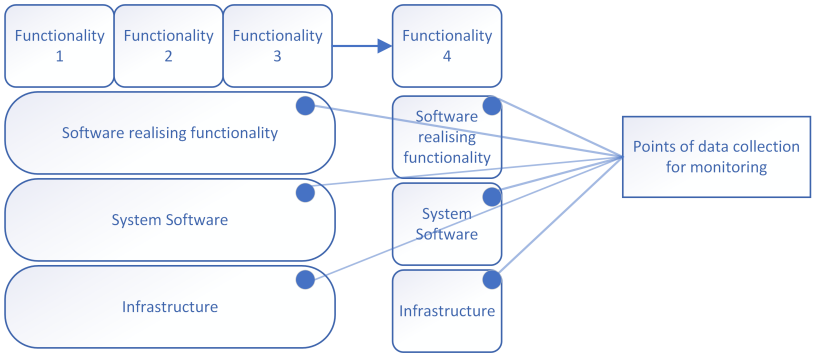

Figure 5.5: Monitoring Software and Infrastructure.

Data monitoring. Every data pipeline in a Cloud-based Big Data analytics application is classified regarding their functionality. Every functionality has specific requirements that need to be fulfilled and monitored. The data analytics monitoring requires that every step that refines the data and adds value to it is monitored for potential events, alerts and incidents. The different functionalities are illustrated in Figure 4.14. For every part involved in the refinement of the raw data, the data engineer has to define metrics that have to be monitored. Monitoring agents have to be implemented or configured to deliver the corresponding metrics for the identification of incidents. The data monitoring focuses completely on the functionality of the value adding parts of the Big Data application. The data monitoring has as goal the QoA. It defines the requirements of the applied functionalities and develops and monitors metrics for the overall analysis process. The QoA monitoring is directly connected to the definition of SLAs to achieve the goals of the different stakeholders.

System architecture monitoring. The focus of data monitoring lies on the the value adding process of data refinement. The system architecture monitoring focusses on infrastructure and software. This includes monitoring requirements regarding software, hardware, network, scaling and elasticity. This part of monitoring concentrates on the parts of the system that enable the data processing functionalities of the Cloud-based Big Data analytics application. QoA defines metrics for the data domain, QoS defines

metrics for the system architecture.

Service level agreements. A possible incident is the degradation of a service. In a Cloud-based Big Data analytics application, where different stakeholders are responsible for the QoS of software and infrastructure parts, the degradation of a service can also incur a violation of SLAs. The Quality objectives are defined over SLO that are measured by SLI. The SLI have to be monitored for any violations regarding the negotiated SLAs that reflect the QoA and QoS goals of the Cloud-based Big Data analytics application. The necessary requirements to measure SLIs have to be fulfilled by the application provider. When a Cloud-based Big Data analytics application is developed the monitoring needs of the various stakeholders have to be considered during the development phase. Getting meaningful monitoring data from an application that is not build to deliver it can prove impossible.

Shared responsibility. The distinction between the different monitoring elements and their assignment to data analytics and system architecture is owed to the fact that the system is hosted in a shared environment. The Cloud-based Big Data analytics application is realised in various degrees with CSs. A distinction has to be made to attribute the incidents exactly to the stakeholder owning the corresponding part of the system illustrated in Figure 5.6.

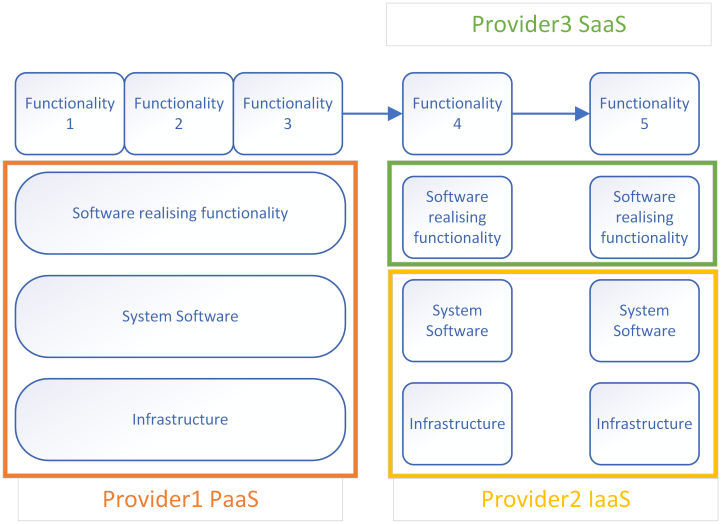

Figure 5.6: Monitoring Cloud Services.

5.2.1 Software Components

Our proposed architecture has several core components that are necessary for the monitoring of incidents of a Cloud-based Big Data analytics application.

- The monitored system or service

- The monitoring agents embedded in the monitoring layer
- The central logging
- The incident and detection and classification module with the centralised rule set
- The visualisation

The monitored system or service. The monitored system or service generates events that form the basis of incidents. It has an instrumentation module that enables the stakeholders to configure exactly which metrics from the system/service are collected. It has a module that offers an interface to fetch the instrumented metrics and a configuration interface that allows to configure the component.

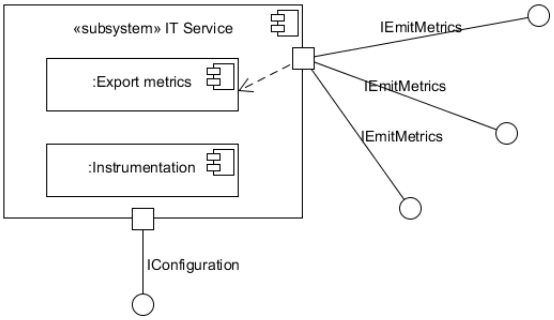

Figure 5.7: Software Component System or Service.

The emitted metrics have different forms, e.g. performance counters, events from the underlying operating system or logs created by an application. The components have to be configured or programmed to generate the metrics that are needed by the monitoring. It is important to specify the frequency of the generation of emitted metrics within meaningful limits. Otherwise it is possible that instrumentation negatively affects performance and becomes its own source of incidents.

The monitoring layer. The provided metrics are fetched, processed and forwarded to the central logging component. This layer hosts various monitoring agents that are tailored to their purpose. The different capabilities of the monitoring agent are affected by the type and form of metrics they monitor. One possibility of a monitoring agent is an agent polling the system metric CPU utilisation with a predefined frequency, aggregating the data, filtering the data and deciding on basis of a configured rule set when to generate events. This kind of monitoring agent is shown in Figure 5.8.

Another form of agent can be embedded into the application it monitors. This is the case for Log4j, a Java-based logging utility from Apache that has built-in log levels and messages. These functionalities are embedded in the monitoring agent in case of the CPU utilisation. The logged data is saved to the file system, filtered, aggregated, evaluated.

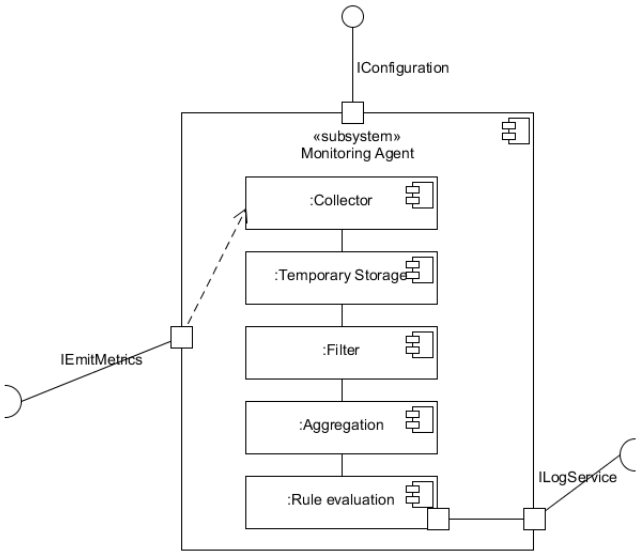

Figure 5.8: Software Component Monitoring Agent.

The data is then fetched by a monitoring agent or directly forwarded by the application to the logging system, both cases are illustrated in Figure 5.9.

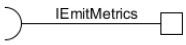

Figure 5.9: Software Component Embedded Monitoring Agent.

The two examples above illustrate the flexibility of the monitoring layer. The different systems and services need monitoring that is tailored to their requirements. This means that the modules employed in monitoring agents can differ from a very complex agent that aggregates, filters and evaluates metrics before creating events to a very simple agent that forwards primed metrics like log data directly into the logging system.

Central event and incident logging. This component is the endpoint of the monitoring layer, any agent delivers his events to the central logging. The central logging offers interfaces for the monitoring agents to write event information into the central logging. The information stored in the central logging is accessible for the stakeholders to analyse and infer incidents. The central logging needs interfaces that enable graphic solutions to build dashboards from the saved data. The component provides options for stakeholders to manually add incidents if they are procured outside the automatised systems. Figure 5.10 illustrates the model view of the component.

Detection and classification. The events from all monitored systems and services are centrally stored in the logging component. In the detection and classification component

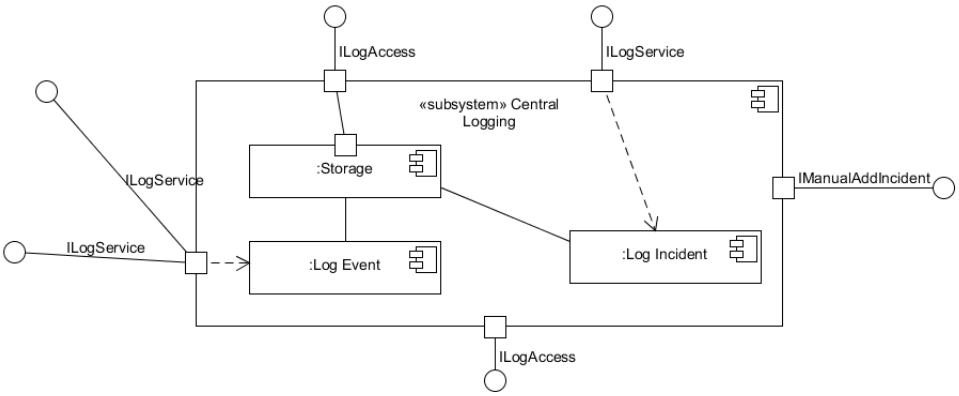

Figure 5.10: Software Component Central Logging.

a centralised rule set is responsible for the filtering, aggregation and correlation of events to detect and classify incidents. This component is separated from the central logging to enable modularity. The different stakeholders have different requirements regarding the instrumentation, monitoring and transfer of events and the detection and classification of incidents. The approaches employed have to be flexible so that the incident management can cover the whole Cloud-based Big Data analytics application. The rule set has to be as simple, predictable and reliable as possible and has to be coordinated with instrumentation and monitoring. The rule set has to be centrally accessible to enable the user to adapt the rules to changing requirements. If an incident detection is rarely exercised it has to be removed. The centralised rule evaluation keeps the configuration small but through its expressiveness powerful. Figure 5.11 illustrates the component.

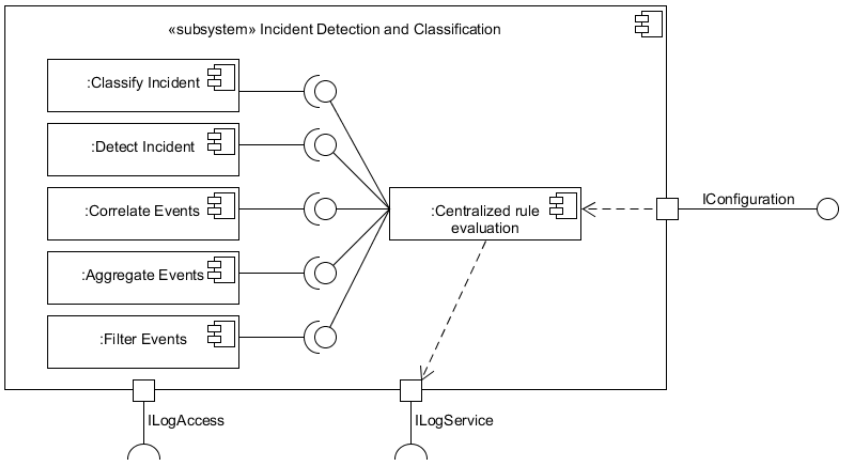

Figure 5.11: Software Component Detection Classification.

Visualisation. The monitored metrics and the detected incidents can be visualised with dashboards. This is one possibility to visualise the data - logged by the monitoring layer - and the incidents detected and classified by the corresponding component. The visualisation component needs access to the logging component to aggregate, filter and visualise the data regarding defined selectors. The component also needs an interface for the users to define what is visualised and how it is visualised. The component is illustrated in Figure 5.12.

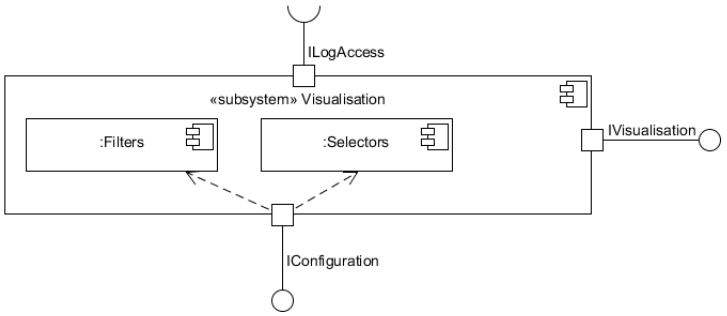

Figure 5.12: Software Component Visualisation.

In a Cloud-based Big Data analytics application it is imperative that the different components can be implemented in various ways to match the requirements of the monitored system or service. Any component illustrated here connects to the other via interfaces and it is always possible to swap or add components that better fulfil the needs of the user of the system. Every stakeholder can implement their own components like monitoring agent or instrumentation and add them to the system. The logging and rule evaluation has to be kept central and simple. The system enables users to use different visualisation components against the same set of data. Different incident identification techniques like complex event processing can be added to the detection component to satisfy the needs of the various stakeholders. Figure 5.13 illustrates the different components and their connections.

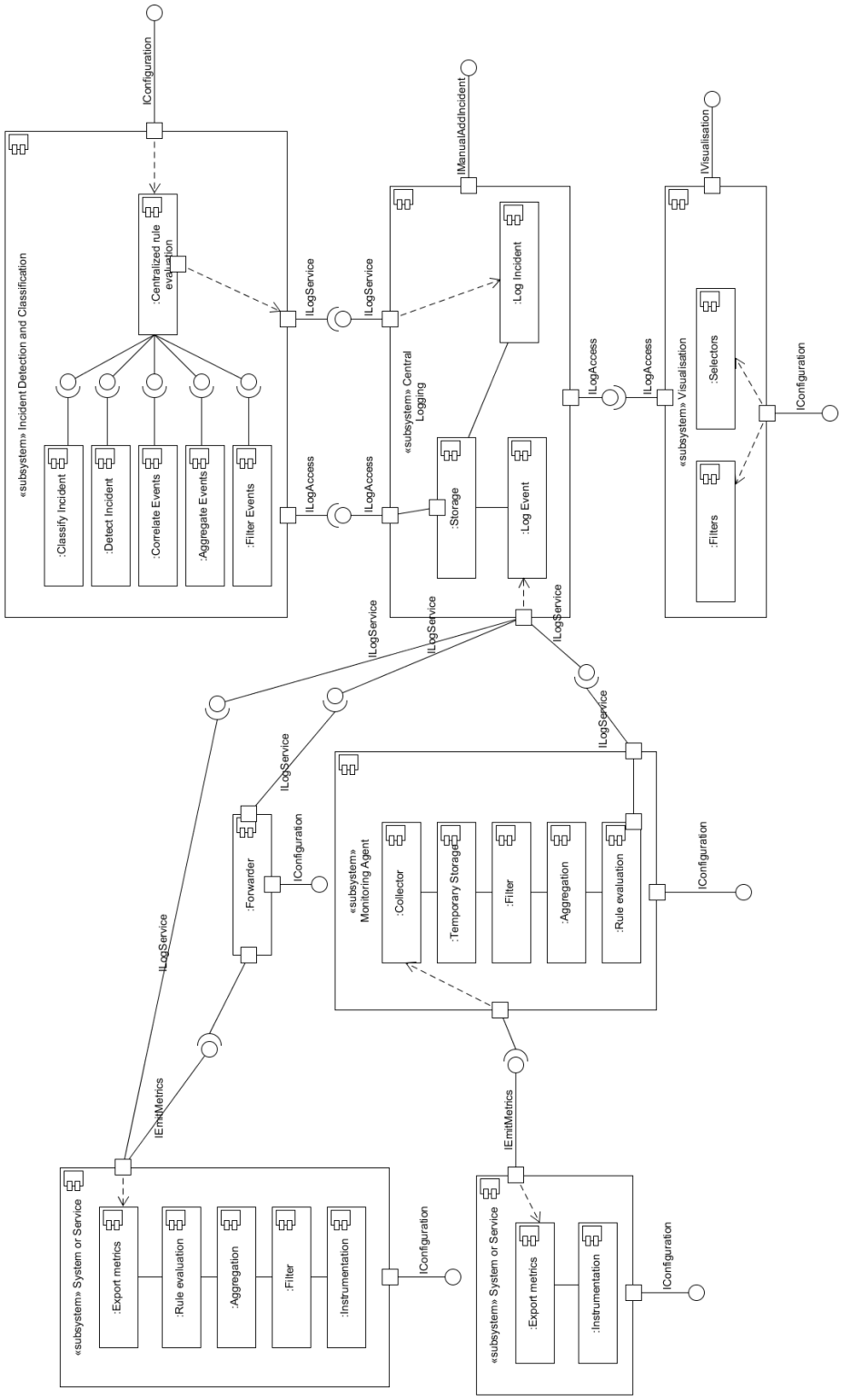

Figure 5.13: Software Components.

CHAPTER 6

Prototype and Experiment

In this chapter the generic architecture for classification and monitoring of incidents in Cloud-based Big Data Analytics is implemented guided by a test scenario to illustrate the feasibility of the architecture. A short example with open source software lowers the entry barrier for other implementations of the generic architecture in different domains. The prototype shows with an example how the architecture is applied to a scenario.

6.1 Prototype

The evaluation of our generic architecture is done by one possible implementation. The components of our architecture can be realised with different applications and technologies but to prove the viability of our architecture one possible implementation is done.

The implementation uses the so called Elastic Stack that bundles different applications in one platform. The stack comprises of the following components:

1. Elasticsearch[26], a distributed RESTful search and analytic engine that centrally stores the data
2. Logstash[67], server-side data processing pipeline
3. Kibana[59], visualisation of Elasticsearch data
4. Beats[15], single-purpose data shipper

The choice of used software is just one out of many possibilities but it fulfils all the criteria for the implementation of our generic architecture. Beats and Logstash realise the monitoring agent component that can apply integrated filtering. Filebeats is a specialised version of Beats and can be used to filter log files and transfer interesting

events directly - or over a Logstash instance - into Elasticsearch. The log files are a typical implementation of a system that collects metrics and emits them into log files on their host's file system. Since most of the Big Data data processing functionalities are written in Java[54], Log4j[66] a configurable Java-based logging utility from Apache is used to emit event data from applications into log files. Log4j therefore fulfils all the instrumentation and emitting requirements of the system or service component of the generic architecture.

Another possibility of a monitoring agent is Telegraf[99], an agent for collecting and reporting metrics and data. Prometheus[84] is a tool that generates alerts based on time-series data like Borgmon[90] from Google. We chose Elasticsearch as central logging store since its usage is widespread and therefore accepts a lot of monitoring agents out of the box or via plugins. Elasticsearch offers storage for custom events programmed by the application provider. This need may arise if for the control of various data pipelines the out-of-the-box means from the data processing functionalities don't suffice.

The decision to use commonly available and widespread technology was done consciously to help others adapt the architecture. By using commonly available technology the implementation of the generic architecture is encouraged and real life examples can be adapted to the specific needs of the managed application.

Where necessary Logstash can function as a data pipeline to process and refine monitoring data before it is saved into Elasticsearch. Logstash can also be used as an intermediary messaging system to control the flow of data into Elasticsearch. The refinement of data becomes necessary if metadata has to be added to enable rule evaluation regarding the detection and classification of incidents.

The detection and classification component is a collection of simple rules that applies the search and analytic capabilities of Elasticsearch to filter, correlate and detect incidents. The classification uses the Neo4J database described in Chapter 4.3. The incident detection and classification component can be realised with a set of Python scripts, this way the implementation is kept simple, flexible and expandable. It is important that the different stakeholders will apply various methodologies to detect incidents therefore the central logging of events and alerts has to offer various options of connection. This enables a wide spectrum of algorithms to satisfy the different needs of the various stakeholders.

The visualisation component is realised with Kibana since it is designed to work on top of Elasticsearch. The visualisation component has to access the event and incident central logging component to access and visualise the stored and analysed data.

Every incident that has to be detected needs to be implemented through the chain of components. For example, the monitored application is a simple one phase data processing step. Specific incidents can be identified by first implementing Filebeats on all the servers running a map reduce job. The Filebeats are configured to transfer all log messages with a level of "WARN" or higher to the Elasticsearch. The Filebeats enrich the documents with metadata in from of the host name and the name of the monitored application. A Python script searches the entries for a specific warning that happens

when the map reduce job - implemented in Java - has not enough memory and fails. In this case the Python script discovers the log messages and evaluates with the rule set that an incident occurred and classifies it as a data analysis incident in the analysis phase in the map reduce software. The script generates an incident. A dashboard in Kibana is configured to visualise any open incidents.

This example illustrates to measure only when recording, record only when analysing and analyse only when it leads to action. Every incident has to be seen as a product of instrumentation, monitoring, analysis and visualisation. It has to be defined beforehand what has to be monitored, what metrics a system or service emits and how to evaluate with rules if or if not a set of events comprises an incident.

Figure 6.1 shows the implementation and the corresponding mapping of the generic architecture.

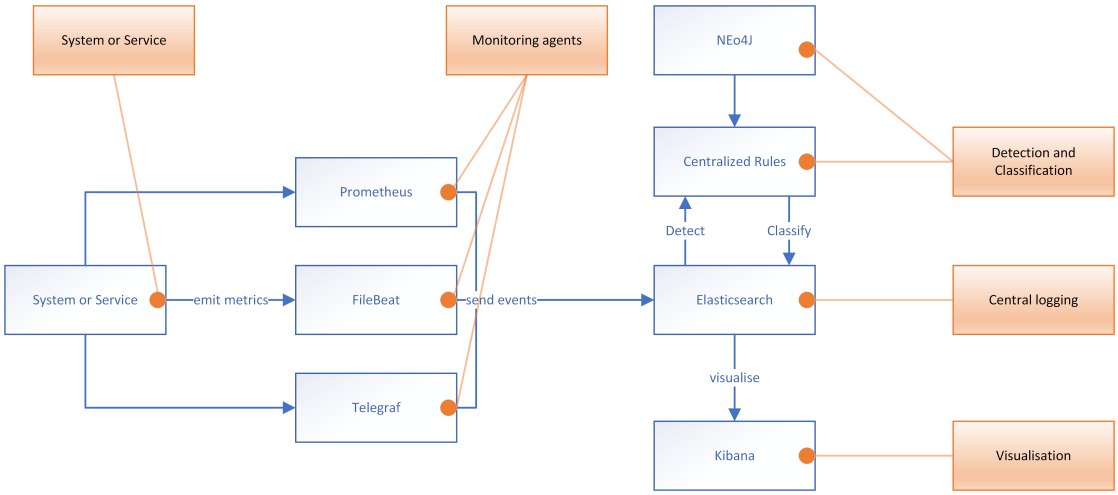

Figure 6.1: Implementation Monitoring.

6.2 Test System

The test system that is watched for incidents is a simplified version of the motivating scenario of BTSs spanning a whole country and providing wireless communication. A simplified version of the system described by I & A computing lab from Vietnam[61] is illustrated in Figure 6.2

The first part of the scenario is the simulation of data generated by IoT BTSs. This part reflects the data acquisition part of the abstraction. The log files are taken from actual BTS. The data from the log files is read with a small application written in Python that uses the Pika library[81] to transfer the data to a RabbitMQ[85] message broker. The message broker hosts a non-durable fanout exchange with a durable queue to store the byte messages from the Python application. The protocol of the message

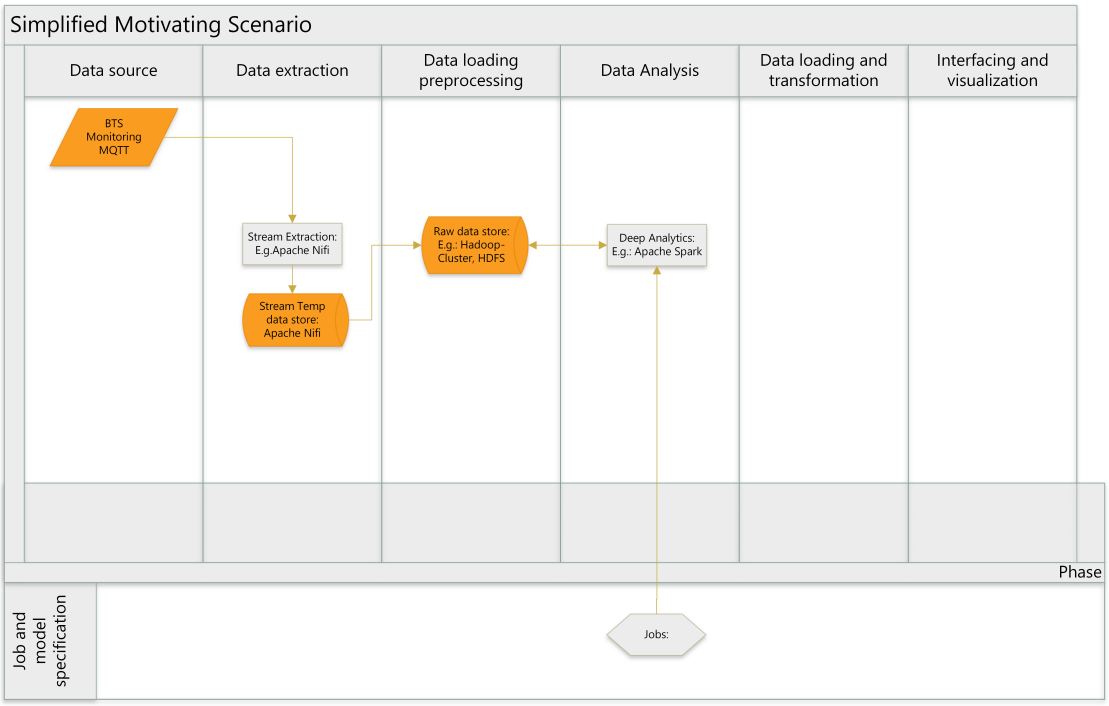

Figure 6.2: Simplified Motivating Scenario.

broker is AMQP. The byte messages are sent from different Raspberry Pi single-board computers generation 2 and 3 model B. The amount of messages sent simultaneously can be manipulated by (i) starting several processes and/or (ii) starting processes on several computers.

Apache Apex is split into Apex-core[8] and Apex-malhar[9] where the latter offers various interfaces for message brokers. Apex is a Apache Hadoop YARN native platform and consists of ports, operators and streams. Apex is written in Java and uses Maven for deployment. The initial data pipeline was built in the following way: a Python script reads from a *.csv file - beforehand collected - real-life data and feeds it into RabbitMQ. Apex is used to subscribe to the exchange, read the messages from RabbitMQ and stream them to the Hadoop file system (HDFS)[49]. Apex reflects one instance of the processing part of the scenario. In this scenario the inputport is RabbitMQ and the outputport is a lineOperator that saves the byte message from RabbitMQ into the Hadoop file system. The RabbitMQ inputport of Apex had several problems some of them remained unsolved and the most pressing was that it needed to use the default user of RabbitMQ which is deactivated for external access. The inputport is not capable of any authentication scheme therefore the RabbitMQ server was configured to accept the default user with no password.

The combination of these two applications made the access of monitoring data complicated since YARN creates the directories from logging with insufficient access rights for potential

monitoring agents. The libraries Apex-malhar offered at that time where not working together with RabbitMQ, if the recommended setup from RabbitMQ was respected. In a first version this scenario was set up on a Rasperry Pi[80] which had to be abandoned since it led to inexplicable errors regarding the Hadoop ecosystem. The first idea was to make it more portable and distributed while keeping it affordable but Big Data applications need a lot of resources and are just not made for small computers like the Pi.

The current version of the data source part of the test system that simulates the BTS devices is realised by IoT Cloud Samples project[93]. This system uses real BTS monitoring data and simulates with Docker-containers[24] the data provider of a big telco infrastructure from Vietnam. This part of the test system consists of sensor containers[42] that read from a *.csv file the collected real BTS monitoring data from the Vietnamese telco provider and ingestion containers[41] that are realised with the open source MQTT Broker Mosquitto[72]. This part of the test scenario starts the sensor containers and the corresponding ingestion containers (for the sensors to log their messages) with Docker Compose[25] and Pipenv[82]. This part completely covers the data provider site of the simplified motivating scenario. The Listing 6.8 illustrates the rough idea behind the operating mode of the sensors.

```
1  import pika
2  import csv
3  import time
4  from collections import OrderedDict
5
6  connection = pika.BlockingConnection(pika.ConnectionParameters(host='
       192.168.33.63'))
7  channel = connection.channel()
8
9  with open('/home/manfred/mqqtTest/IoT/iot-test-data.csv') as csvfile:
10     reader = csv.reader(csvfile)
11     sortedlist = sorted(reader, key=lambda t: t[2])
12     for i,row in enumerate(sortedlist):
13         channel.basic_publish(exchange='apex',
14                               routing_key='rktest',
15                               body=str(', '.join(row)))
16         print(str(', '.join(row)))
17         time.sleep(1)
18         if(i >= 30):
19             break
20
21 connection.close()
```

Listing 6.1: An previous version of the data input. This illustrates how data is fed into the messaging infrastructure. The same approach is used in the final version by the sensor containers.

The next part of the scenario enters various domains of different stakeholders. The system orchestrator - here realised as the data scientist actor - is involved in the whole scenario, any incidents stemming from the data source to data processing are of interest to this

stakeholder as soon as they influence the outcome of the analysis. The data extraction is realised with an Apache Nifi[38] server. The main purpose of this service is to extract the data from the messaging infrastructure and ingest it into the storage infrastructure. This step can include different data preprocessing steps. The configuration of Apache Nifi works via Browser and is illustrated in Figure 6.3. There are two processors, one to consume the MQTT messages and a second to write the messages into the HDFS. There is one message queue that connects the two processors. It is possible to add several Big Data mechanics like filtering and/or data preprocessing tasks between the two Nifi processors. Apacha Nifi is a streaming application and therefore offers different methods to manipulate the data sets during their transition to the HDFS.

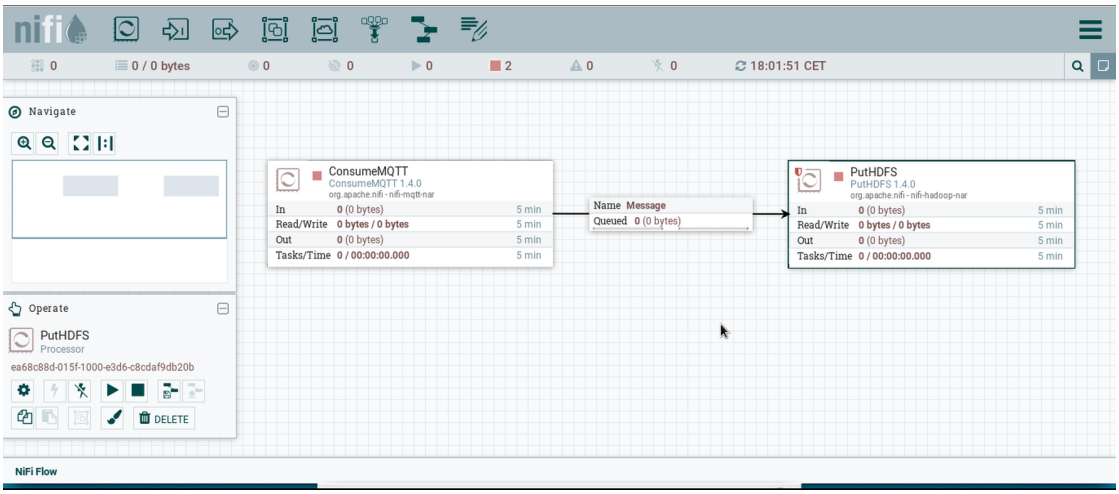

Figure 6.3: Apache Nifi Configuration.

The data is stored in the HDFS. The Hadoop installation used in the test scenario is a single node cluster. Despite being a highly distributable application the installation runs only on one node. Other parts of an standard Hadoop installation is Hadoop YARN, a framework for job scheduling and cluster resource management and Hadoop MapReduce, a YARN-based system for parallel processing of large data sets. These two parts are standard Big Data analysis applications that are used to process the collected data sets.

6.3 Application of the Generic Architecture

The first step was the identification of all the classification information and adding them to the classification graph. Regarding the generic architecture the management of an incident always includes the following steps :

1. Instrumentation
2. Collection

3. Central storage
4. Incident detection and classification

The functionality of the different elements involved in the test scenario are defined using the generic architecture. The phases can be assigned in the same way as the functionality. In the next step the multiphase data pipeline has to be identified and represented in the classification illustrated in Figure 6.4

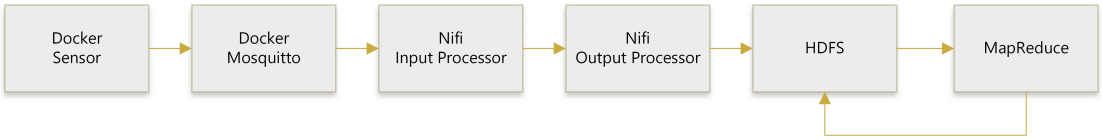

Figure 6.4: Test Scenario Data Pipeline.

6.3.1 Instrumentation

We show how the instrumentation of the application generating monitoring data works. This illustrates the first part of the incident life-cycle.

The instrumentation is partly provided by the applications itself as illustrated in Figure 5.9. In the case of Apache Nifi generates messages regarding the state of the application and stores them in log files. Logstash searches the log files for a defined pattern, enriches it with information necessary for the classification and sends it to Elasticsearch illustrated in Listing 6.2. This concludes the instrumentation.

```
#Definition input
input {
        file {
        path => "/opt/nifi/nifi-1.4.0/logs/*"
        id => "nifiLogstash"
        start_position => "beginning"
        codec => multiline {
            pattern => "^%{TIMESTAMP_ISO8601},"
            negate => true
            what => "previous"
        }
        }
}

#add possible filters
filter {
    grok {
        match => { "message" => "%{DATESTAMP:date},%{NUMBER:errorcode} %{
    LOGLEVEL:loglevel} %{GREEDYDATA:syslog_message}"}
    }
}

filter {
```

```
23    if [loglevel] == "INFO" {
24      drop { }
25    }
26  }
27
28  #Definition of output just writes to the command line
29  output {
30      elasticsearch {
31          id => "nifi1_Plugin"
32          index => "nifilogfiles"
33          hosts => ["localhost:9200"]
34      }
35      stdout {codec => rubydebug }
36  }
```

Listing 6.2: The configuration of the second part of the instrumentation of an application in the Cloud-based Big Data analytics data pipeline. The index is set so that the application can be identified and classified in the central logging.

6.3.2 Classification and Central Storage

We show how a central storage for incidents and events is realised with open source components. We illustrate how the basic classification has to be extended to cover the collected monitoring data.

The next step is the central storage, in the prototype this is realised with an instance of Elasticsearch. The usage of dashboards supplied by Kibana simplifies the administration. The image in Figure 6.5 illustrates that for the example of the indexed messages from the Apache Nifi element of the data pipeline of the test scenario.

Figure 6.5: Kibana Dashboard for Apache Nifi.

Before classification and detection can happen various nodes are added into the classification. Since the classification is a graph database nodes, edges and properties describing the data pipeline have to be added.

```
CREATE (ApacheNifi:DataPipeline{name:'ApacheNifi'})
CREATE (MQTT:DataPipeline{name:'MQTT'})
CREATE (HDFS:DataPipeline{name:'HDFS'})
CREATE (Linux:SystemSoftware{name:'Linux'})
CREATE (Archibald:Infrastructure{name:'Archibald'})
CREATE (FrameworkProvider:Stakeholder{name:'Framework Provider'})
CREATE (Provider1:FrameworkProvider{name:'Framework Provider One'})
CREATE (Provider2:FrameworkProvider{name:'Framework Provider Two'})
```

Listing 6.3: The usability of the classifiaction is only guaranteed when the corresponsing analysis work is done beforehand and the various data pipeline elements are added to the classification database.

```
  (ApacheNifi)-[:IS]->(DataPipeline),
  (MQTT)-[:IS]->(DataPipeline),
  (HDFS)-[:IS]->(DataPipeline),
  (MQTT)-[:PREDECESSOR]->(ApacheNifi),
  (ApacheNifi)-[:PREDECESSOR]->(HDFS),
  (ApacheNifi)-[:BELONGSTO]->(DataLoadingPreprocessing),
  (ApacheNifi)-[:IS]->(ApplicationSoftware),
  (ApacheNifi)-[:BELONGSTO]->(Preparation),
  (ApacheNifi)-[:DataAssetState]->(InProcessing),
  (ApacheNifi)-[:HostedOn]->(Linux),
  (Linux)-[:IS]->(SystemSoftware),
  (Stakeholder)-[:IS]->(FrameworkProvider),
  (Provider1)-[:OWNS]->(ApacheNifi),
  (Provider2)-[:OWNS]->(HDFS),
  (DataProvider)-[:OWNS]->(MQTT),
  (Provider1)-[:IS]->(FrameworkProvider),
  (Provider2)-[:IS]->(FrameworkProvider),
```

Listing 6.4: The corresponding relationsships.

The added edges and properties to nodes enable the classification of a detected incident. The last step is a as-simple-as-possible rule to identify an incident. As stated in the research the openness of the Elasticsearch installation offers various interfaces to apply incident analytics. Different stakeholders need different analytics and any imaginable analytic procedure can be applied to the centrally stored data. The classification always remains central and it can be extended to fulfil the needs of the stakeholders.

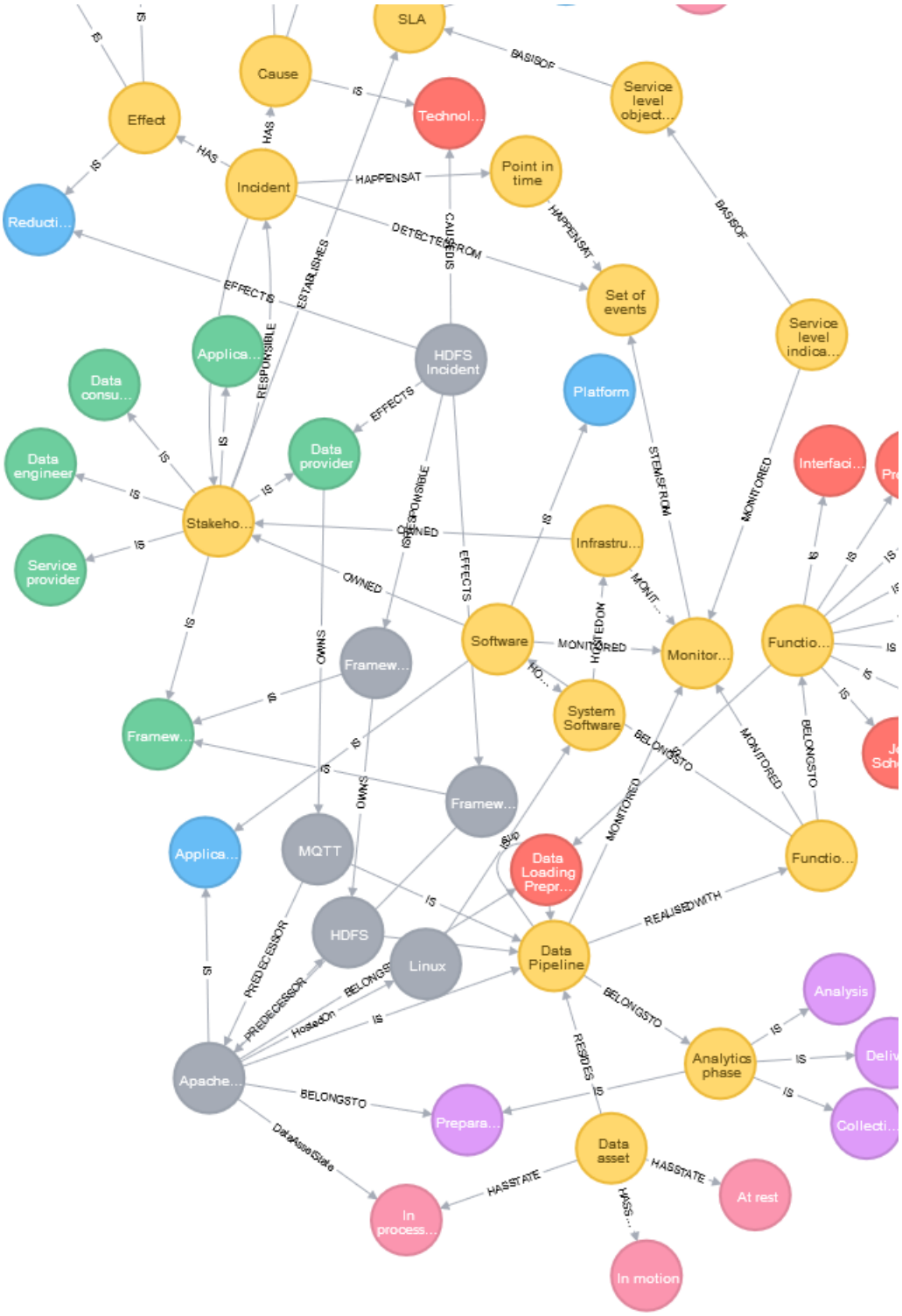

Figure 6.6: Classification Example.

6.3.3 Simple Identification and Classification Rule

We show how a simple rule that identifies an incident is implemented, by analysing the centrally stored data and classifying it using the classification graph.

A simple rule written in Python identifies, classifies and writes the incident back into Elasticsearch adding id and index so that it can be easily searched in the Kibana dashboard. This example specifically searches for errors regarding the connectivity to the HDFS and the MQTT message container. If any of them is unreachable, the rule deducts that the element itself and/or the predecessors in the corresponding data pipeline have a problem. The corresponding owners and pipeline elements are added to the incident.

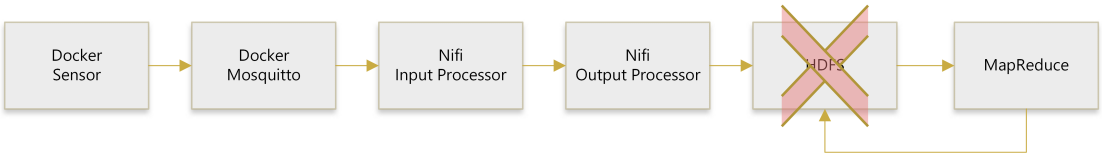

Figure 6.7: Test Scenario Data Pipeline Incident.

The simulated incident is the complete failure of one part of a data pipeline. The centralised logging is searched for potential base events and alerts that suffice the requirements of an incident. The events are scanned to find out if they are already part of an incident, in that case they are ignored and no new incident is created. The Python script uses the Elasticsearch Python library[27].

```
DATABASE = GraphDatabase("http://localhost:7474", username="neo4j",
    password="Ztu5C!:")
data = {}

#Search for incidents
RESPONSE = CLIENT.search(
    index='nifilogfiles', body={
        "size" : 100, "query": {
            'bool' : {
                'must': [
                    {"match": {'loglevel':'ERROR'}},
                    {"match": {'message':'HDFS'}}
                ],
                'must_not': [
                    {"match": {'partofincident':'YES'}}
```

Listing 6.5: A simple search for events in the created index.

In the next step the Neo4j database is queried for the classification data. The script traverses the database and returns the corresponding values that are needed for the creation of the incident. The connection to the graph database is realised with the neo4jrestclient[75].

```
1  def classify_incident():
2      """Classifies the data regarding the neo4j database"""
3      data['owner'] = (DATABASE.query(('MATCH (nifi {name: "ApacheNifi"})<-[:OWNS]- (provider)RETURN
       provider.name'),data_contents=True)[0][0])
4      data['phase'] = (DATABASE.query(('MATCH (nifi {name: "ApacheNifi"})-[:BELONGSTO]-(phase)-[:IS]-(:
       Element{name:"Analytics phase"})RETURN phase.name'),data_contents=True)[0][0])
5      data['functionalarea'] = (DATABASE.query(('MATCH (nifi {name: "ApacheNifi"})-[:BELONGSTO]-(function)
       -[:IS]-(:Element{name:"Functional area"})RETURN function.name'),data_contents=True)[0][0])
6      data['assetstate'] = (DATABASE.query(('MATCH (nifi {name: "ApacheNifi"})-[:ASSETSTATE]-(asset)-[:
       HASSTATE]-(:Element{name:"Data asset"})RETURN asset.name'),data_contents=True)[0][0])
7      data['predeccessor'] = (DATABASE.query(('MATCH (pipes:DataPipeline)-[relatedTo]->(:DataPipeline {name:
        "ApacheNifi"})RETURN pipes.name'),data_contents=True)[0][0])
8      data['sucessor'] = (DATABASE.query(('MATCH (pipes:DataPipeline)<-[relatedTo]-(:DataPipeline {name: "
       ApacheNifi"})RETURN pipes.name'),data_contents=True)[0][0])
9      data['effect'] = 'reduction of quality'
10     data['cause'] = 'unknown'
11     data['date'] = (datetime.today().strftime("%Y-%m-%d %H:%M:%S"))
```

Listing 6.6: The classification of the incident regarding the previously defined classification parameters.

Errors that already caused an incident are not used for another incident. Events or alerts get an additional field to mark them as already being part of an incident.

```
def change_events():
    """Adds the events to the corresponding incidents"""
    for hit in RESPONSE['hits']['hits']:
        if "HDFS" in hit['_source']['message']:
            print('HDFS error:' + hit['_source']['errorcode'] + hit['_id'])
            if 'events' in data:
                data['events'].append(hit['_id'])
            else:
                data['events'] = [hit['_id']]
            CLIENT.update(index='nifilogfiles',\
                    doc_type='doc',\
                    id=hit['_id'],\
                    body={
                        "script" : "ctx._source.partofincident = 'YES'"
                        }
                         )
```

Listing 6.7: The events or alerts are marked.

The last step is the creation of the incident. The data collected consists of the participating events and the results of the classification from the graph database. The collected data is written back into the Elasticsearch central logging.

```
def create_incident():
    """Creates an incident"""
    result = CLIENT.index(index="incident", doc_type='doc', body=data)
    print(result['created'])
```

Listing 6.8: The incident is written into the database.

Now the incident is added to the corresponding index in the Elasticsearch database and becomes searchable. The incident contains the identification tags from all the events it is based on.

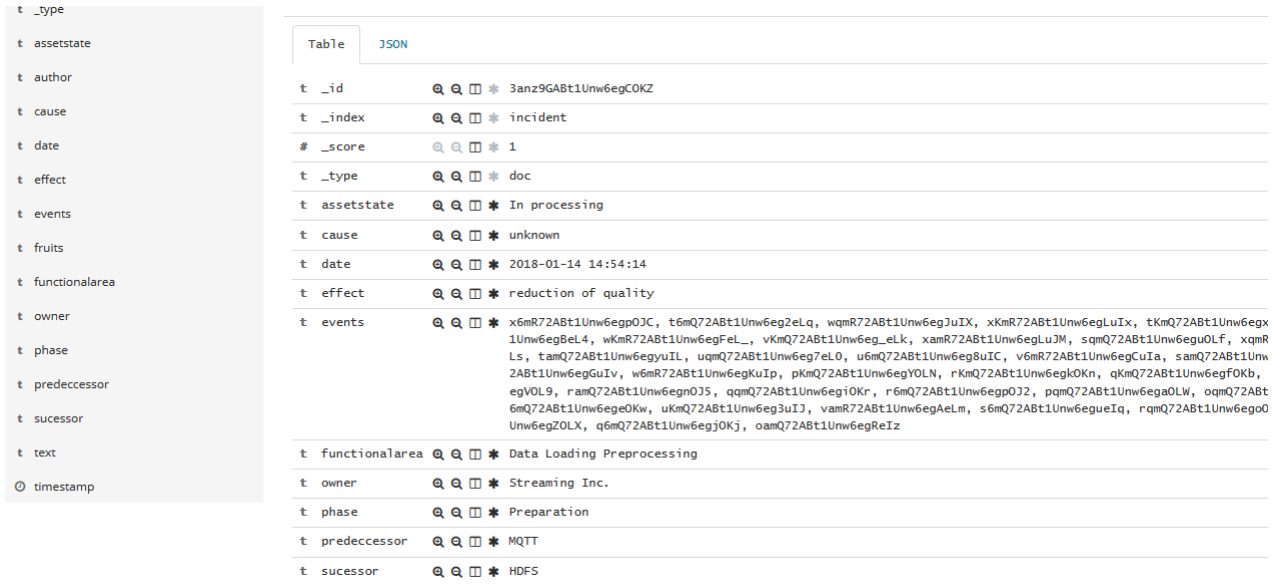

Figure 6.8: Detected and Classified Incident.

This proof-of-concept illustrates a simple rule that realises the concept of generating rules as simple as possible. Since the data and the corresponding created incident are all stored in a central logging unit any user of the system has access to them. The use of Python to create a rule is only one possibility. The central logging unit in this example is accessible with a wide variety of tools and enables any stakeholder to design rules in their preferred programming language.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

Our intensive related and background research delivers the basis for all the other contributions, our stakeholder analysis, our specification of generic use cases for incident management, our incident survey, our classification, our metrics, our specification of monitoring requirements, our development of software component diagrams and our proof-of-concept of a generic architecture of an incident monitoring and classification system for incidents in Cloud-based Big Data systems. We found a motivating scenario that helped orienting in a field with high complexity. We identified use cases and stakeholders considering the new challenges stemming from technology paradigms and combined them with proven standards. Our incident survey delivered input to identify key points in this technological area of expertise. Another contribution is our classification of incidents based on our survey and our research regarding cloud computing, IoT and Big Data. The combination of the survey with the classification delivered exemplary metrics regarding quality goals like QoA and QoS. We developed monitoring requirements and combined them with all the other intermediate contributions to deliver the software components of our generic architecture. The thesis finishes with a proof-of-concept regarding the implementation of our generic architecture with open source technologies. This example illustrates the feasibility of our prototype and reiterates how the generic architecture has to be applied.

The software used in the proof-of-concept is released as open source under `https://github.com/rdsea/bigdataincidentanalytics`.

7.2 Future Work

There are several use cases that have been identified in the research but were not examined in the prototype. The next step is to introduce the system to a real case scenario and

implement more and more sophisticated rules. The generic architecture presented in this thesis develops the understanding and illustrates the possibilities in the field of incident management in Cloud-hosted Big Data analytics. The prototype presented here is a proof-of-concept to verify the feasibility of the architecture.

Bibliography

[1] C. C. Aggarwal. *An Introduction to Data Mining*. Springer International Publishing, Cham, 2015.

[2] C. C. Aggarwal, N. Ashish, and A. Sheth. *The Internet of Things: A Survey from the Data-Centric Perspective*, pages 383–428. Springer US, Boston, MA, 2013.

[3] U. Ahsan and A. Bais. A review on big data analysis and internet of things. In *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 325–330, Oct 2016.

[4] C. S. Alliance. The treacherous 12 - cloud computing top threats in 2016. Technical report, Cloud Security Alliance, 2017.

[5] R. Altschaffel, S. Kiltz, and J. Dittmann. From the computer incident taxonomy to a computer forensic examination taxonomy. In *2009 Fifth International Conference on IT Security Incident Management and IT Forensics*, pages 54–68, Sept 2009.

[6] S. Angelov, P. Grefen, and D. Greefhorst. A framework for analysis and design of software reference architectures. *Information and Software Technology*, 54(4):417 – 431, 2012.

[7] S. Angelov, J. J. M. Trienekens, and P. Grefen. *Towards a Method for the Evaluation of Reference Architectures: Experiences from a Case*, pages 225–240. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[8] apex core. `https://github.com/apache/apex-core/blob/v3.6.0/CHANGELOG.md` Version:3.6.0 Accessed: 21.03.2017.

[9] apex malhar. `https://github.com/apache/apex-malhar/tree/v3.6.0` Version 3.6.0 Accessed: 21.03.2017.

[10] K. Aravind. *Framework for Building a Big Data Platform for Publishing Industry*, pages 377–388. Springer International Publishing, Cham, 2015.

[11] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79(Supplement C):3 – 15, 2015. Special Issue on Scalable Systems for Big Data Management and Analytics.

[12] L. A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition.* Morgan & Claypool Publishers LLC, 2013.

[13] L. A. Bass, R. C. Kazman, and P. C. Clements. *Software architecture in practice.* Addison Wesley, second edition edition, 2003.

[14] L. A. Bass, R. C. Kazman, and P. C. Clements. *Software architecture in practice.* SEI series in software engineering Software architecture in practice. Addison Wesley, [Place of publication not identified], 2013.

[15] Beats. `https://www.elastic.co/de/products/beats` Accessed: 06.01.2018.

[16] E. Borgia. The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1 – 31, 2014.

[17] J. M. Cavanillas, E. Curry, and W. Wahlster. *New Horizons for a Data-Driven Economy.* Springer International Publishing, 2016.

[18] M. Chen, S. Mao, and Y. Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[19] M. Chen, S. Mao, Y. Zhang, and V. C. Leung. *Big Data.* Springer International Publishing, 2014.

[20] P. Cichonski, T. Millar, T. Grance, and K. Scarfone. Computer security incident handling guide, aug 2012.

[21] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, June 2012.

[22] E. Curry, A. F. (NUIG), A. T. (UIBK), A. F. (UIBK), A. N. (INFAI), I. E. (INFAI), K. L. (INFAI), M. S. (AGT), H. R. (AGT), M. L. (AGT), J. D. (AGT), A. Z. (INFAI), P. K. (AGT), J. D. (STIR), N. L. (UIBK), M. N. (INFAI), M. M. (INFAI), M. M. (INFAI), P. F. (INFAI), S. C. (INFAI), S. H. (INFAI), T. B. (DFKI), T. van Kasteren (AGT), and U. U. H. (NUIG). Big data public private forum (big), may 2014.

[23] Y. Demchenko, C. de Laat, and P. Membrey. Defining architecture components of the big data ecosystem. In *2014 International Conference on Collaboration Technologies and Systems (CTS)*, pages 104–112, May 2014.

[24] Docker. `https://www.docker.com/` Accessed: 07.01.2018.

[25] Docker. Compose. `https://docs.docker.com/compose/` Accessed: 07.01.2018.

[26] Elasticsearch. Version 6.1.1 https://www.elastic.co/de/products/elasticsearch Accessed: 06.01.2018.

[27] elasticsearch py. https://pypi.python.org/pypi/elasticsearch/6.1.1 Version 6.1.1 Accessed: 14.01.2018.

[28] S. A. Elnagdy, M. Qiu, and K. Gai. Cyber incident classifications using ontology-based knowledge representation for cybersecurity insurance in financial industry. In *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 301–306, June 2016.

[29] V. C. Emeakaroha, M. A. S. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. F. De Rose. Towards autonomic detection of sla violations in cloud infrastructures. *Future Gener. Comput. Syst.*, 28(7):1017–1029, July 2012.

[30] P. T. Endo, G. L. Santos, D. Rosendo, D. M. Gomes, A. Moreira, J. Kelner, D. Sadok, G. E. Gonçalves, and M. Mahloo. Minimizing and managing cloud failures. *Computer*, 50(11):86–90, November 2017.

[31] ENISA. Good practices on reporting security incidents, 2009.

[32] ENISA. Exploring cloud incidents, jun 2016.

[33] E. U. A. for Network and I. Security. Good practice guide for incident management, dec 2010.

[34] T. I. O. for Standardization and T. I. E. Commission. Information technology - security techniques - information security incident management. Standard ISO/IEC 27035:2011, The International Organization for Standardization, sep 2011.

[35] T. I. O. for Standardization and T. I. E. Commission. Information technology - cloud computing - overview and vocabulary. Standard ISO/IEC 17788:2014(E), The International Organization for Standardization, Geneva, CH, oct 2014.

[36] T. I. O. for Standardization and T. I. E. Commission. Information technology - cloud computing - reference architecture. Standard ISO/IEC 17789:2014(E), The International Organization for Standardization, oct 2014.

[37] T. I. O. for Standardization and T. I. E. Commission. Information technology cloud computing overview and vocabulary. Standard, The International Organization for Standardization, 10 2015.

[38] T. A. S. Foundation. Nifi. https://nifi.apache.org/docs.html Version: 1.4.0 Accessed: 07.01.2018.

[39] M. Galster and P. Avgeriou. Empirically-grounded reference architectures: A proposal. In *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA*

and Architecting Critical Systems – ISARCS, QoSA-ISARCS '11, pages 153–158, New York, NY, USA, 2011. ACM.

[40] J. Gantz and D. Reinsel. Extracting value from chaos. *IDC IView*, jun 2011.

[41] L. Gao. rdsea ingestion. `https://hub.docker.com/r/rdsea/ingestion/` Accessed: 07.01.2018.

[42] L. Gao. rdsea sensor. `https://hub.docker.com/r/rdsea/sensor/` Accessed: 07.01.2018.

[43] I. Gartner. Gartner's hype cycle special report for 2010. online, oct 2010. `https://blogs.gartner.com/hypecyclebook/2010/09/07/2010-emerging-technologies-hype-cycle-is-here/` Acessed: 6.01.2018.

[44] I. Gartner. Gartner's 2013 hype cycle special report. online, aug 2013. `https://www.gartner.com/newsroom/id/2575515` Acessed: 6.01.2018.

[45] I. Gartner. Gartner's 2014 hype cycle special report. online, aug 2014. `https://www.gartner.com/newsroom/id/2819918` Acessed: 6.01.2018.

[46] I. Gartner. Gartner's 2015 hype cycle special report. online, aug 2015. `https://www.gartner.com/newsroom/id/3114217` Acessed: 6.01.2018.

[47] I. Gartner. Gartner's 2017 hype cycle special report. online, aug 2017. `https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/` Acessed: 6.01.2018.

[48] D. Greefhorst, H. Koning, and H. v. Vliet. The many faces of architectural descriptions. *Information Systems Frontiers*, 8(2):103–113, Feb 2006.

[49] A. Hadoop. `https://hadoop.apache.org/` Accessed: 07.01.2018.

[50] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, 2013. USENIX.

[51] J. D. Howard and T. A. Longstaff. A common language for computer security incidents, 1998.

[52] H. HU, Y. WEN, T.-S. CHUA1, and X. LI3. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, jul 2014.

[53] C. Humby, 2006.

[54] Java. `https://java.com/de/` Accessed: 06.01.2018.

[55] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative support for sensor data cleaning. In K. P. Fishkin, B. Schiele, P. Nixon, and A. Quigley, editors, *Pervasive Computing: 4th International Conference, PERVASIVE 2006, Dublin, Ireland, May 7-10, 2006. Proceedings*, pages 83–100. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[56] A. Karkouch, H. Mousannif, H. A. Moatassime, and T. Noel. Data quality in internet of things: A state-of-the-art survey. *Journal of Network and Computer Applications*, 73:57 – 81, 2016.

[57] K. Kent, S. Chevalier, T. Grance, and H. Dang. *Guide to Integrating Forensic Techniques into Incident Response*. The National Institute of Standards and Technology, aug 2006.

[58] S. Khan, A. Gani, A. W. A. Wahab, M. A. Bagiwa, M. Shiraz, S. U. Khan, R. Buyya, and A. Y. Zomaya. Cloud log forensics: Foundations, state of the art, and future directions. *ACM Computing Surveys (CSUR)*, 49(1), may 2016.

[59] Kibana. `https://www.elastic.co/de/products/kibana` Accessed: 06.01.2018.

[60] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, Nov 1995.

[61] I. . A. C. Lab. `www.inacomputing.com` Accessde: 06.01.2018.

[62] D. Laney. 3-d data management: Controlling data volume, velocity and variety. META Group Research Note, feb 2001.

[63] A. Limited. Itil service design, 2011.

[64] A. Limited. Itil service operation, 2011.

[65] A. Limited. Itil service strategy, 2011.

[66] Log4j. `https://logging.apache.org/log4j/2.x/` Accessed: 06.01.2018.

[67] Logstash. `https://www.elastic.co/de/products/logstash` Accessed: 06.01.2018.

[68] J.-G. Lou, Q. Lin, J. Ding, Q. Fu, D. Zhang, and T. Xie. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ASE 2013, November 2013.

[69] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[70] J. I. Maletic and A. Marcus. Data cleansing: Beyond integrity analysis. In *Fifth Conference on Information Quality (IQ 2000)*, pages 200–209, 2000.

[71] L. Marinos. *ENISA Threat Taxonomy A tool for structuring threat information.* ENISA, jan 2016.

[72] Mosquitto. `https://mosquitto.org/` Accessed: 07.01.2018.

[73] V. I. Munteanu, A. Edmonds, T. M. Bohnert, and T.-F. Fortis. Cloud incident management. Challenges, Research Directions and Architectural Approach. *IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014.

[74] P. Murthy, A. Bharadwaj, P. A. Subrahmanyam, A. Roy, and S. Rajan. Big data taxonomy, sep 2014.

[75] neo4jrestclient. `https://pypi.python.org/pypi/neo4jrestclient/2.1.1` Version 6.1.1 Accessed: 14.01.2018.

[76] N. I. of Standards and Technology. The nist definition of cloud computing, 01 2011.

[77] N. I. of Standards and Technology. Nist big data interoperability framework: Volume 1, definitions, sep 2015.

[78] N. I. of Standards and Technology. Nist big data interoperability framework: Volume 6, reference architecture, sep 2015.

[79] N. I. of Standards and Technology. Nist big data interoperability framework:volume 2, big data taxonomies, sep 2015.

[80] R. Pi. `https://www.raspberrypi.org/`.

[81] Pika. `https://pika.readthedocs.io/en/0.10.0/` Version 0.10 Accessed: 22.01.2018.

[82] pipenv. `https://pipenv.readthedocs.io/en/latest/` Accessed: 07.01.2018.

[83] P. Pääkkönen and D. Pakkala. Reference architecture and classification of technologies, products and services for big data systems. *Big Data Research*, 2(4):166 – 186, 2015.

[84] Prometheus. `https://prometheus.io/` Accessed: 06.01.2018.

[85] RabbitMQ. `https://github.com/rabbitmq/rabbitmq-server/releases/tag/rabbitmq_v3_6_10` Version: 3.6.10 Accessed: 21.03.2017.

[86] J. Radatz. *The IEEE Standard Dictionary of Electrical and Electronics Terms.* IEEE Standards Office, New York, NY, USA, 6th edition, 1997.

[87] N. H. A. Rahman and K.-K. R. Choo. A survey of information security incident handling in the cloud. *Computers & Security*, 49:45–69, mar 2015.

[88] B. K. Raju and G. Geethakumari. A novel approach for incident response in cloud using forensics. *COMPUTE '14 Proceedings of the 7th ACM India Computing Conference*, 2014.

[89] P. Ray. A survey on internet of things architectures. *Journal of King Saud University Computer and Information Sciences*, oct 2016.

[90] M. N. Richard, B. Betsy, and P. Jennifer. *Site Reliability Engineering*. O'Reilly Media, 2016.

[91] A. Rosà, L. Y. Chen, and W. Binder. Catching failures of failures at big-data clusters: A two-level neural network approach. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pages 231–236, June 2015.

[92] A. Rosa, L. Y. Chen, and W. Binder. Failure analysis and prediction for big-data systems. *IEEE Transactions on Services Computing*, PP(99):1–1, 2017.

[93] I. C. Samples. `https://github.com/rdsea/IoTCloudSamples` Accessed: 07.01.2018.

[94] G. L. Santos, P. T. Endo, G. Gonçalves, D. Rosendo, D. Gomes, J. Kelner, D. Sadok, and M. Mahloo. Analyzing the it subsystem failure impact on availability of cloud services. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 717–723, July 2017.

[95] S. Sharma. Expanded cloud plumes hiding big data ecosystem. *Future Generation Computer Systems*, 59:63 – 92, 2016.

[96] J. Spiess, Y. T'Joens, R. Dragnea, P. Spencer, and L. Philippart. Using big data to improve customer experience and business performance. *Bell Labs Technical Journal*, 18(4):3–17, March 2014.

[97] R. N. Taylor, N. Medvidoviâc, N. Medvidovic, and E. M. Dashofy. *Software architecture : foundations, theory, and practice*. John Wiley, [Place of publication not identified], 2010.

[98] H. TECHNOLOGIES. NodeB V200R013 Technical Description, sep 2011.

[99] Telegraf. `https://github.com/influxdata/telegraf` Accessed: 06.01.2018.

[100] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[101] VMware. Proactive incident and problem management. intelligent analytics and automated control reduce downtime, increase responsiveness, and drive increased investment in innovation, 2012.

[102] M. Woodard, M. Wisely, and S. S. Sarvestani. Chapter three - a survey of data cleansing techniques for cyber-physical critical infrastructure systems. In A. R. Hurson and M. Goudarzi, editors, *Advances in Computers*, volume 102 of *Advances in Computers*, pages 63 – 110. Elsevier, 2016.

[103] I. Yaqoob, I. A. T. Hashem, A. Gani, S. Mokhtar, E. Ahmed, N. B. Anuar, and A. V. Vasilakos. Big data: From beginning to future. *International Journal of Information Management*, 36(6, Part B):1231 – 1247, 2016.