Data Warehousing for Cloud Computing Metrics

Projektdokumentation

Tim Meusel Nikolai Luis Marcel Reuter

25. Mai 2017

PARTNERFIRMA
Steve Quin
David Schmitt

BETREUER
Dirk Stegemann

Danksagung

Wir bedanken uns bei der Firma Puppet, insbesondere bei David Schmitt und Steve Quin. Euer Fachwissen und eure Unterstützung haben uns sehr geholfen, das Projekt ordentlich zu strukturieren und auf einem hohen technischen Niveau abzuschliessen. Außerdem möchten wir uns bei Hunter Haugen für die große Hilfestellung im Bereich Unit Testing bedanken.

Ein besonderer Dank gebührt Ulli Kehrle. Ohne seine unermüdlichen Erklärungen zum Thema LaTeX und seine Hilfestellungen, auch in den späten Abendstunden, würde diese Dokumentation nicht in dieser Form existieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

Tabellenverzeichnis

1 Einführung

Das Heinrich-Hertz-Europakolleg Bonn verlangt im fünften und sechsten Semester der Weiterbildung zum staatlich geprüften Informatiker eine fachbezogene Projektarbeit. Dieses Projekt wird in Gruppen von zwei bis vier Personen durchgeführt und soll fachliche Inhalte sowie Techniken aus dem Projektmanagement kombinieren. Es handelt sich um eine praktische Arbeit. Jeder Abschnitt enthält am Ende das Kürzel des Autors, hierbei bedeutet:

- [MR] erstellt von Marcel Reuter
- [NL] erstellt von Nikolai Luis
- [TM] erstellt von Tim Meusel

Die Zusammensetzung dieses Teams entstand während der regulären Unterrichtszeit im dritten Semester. Herr Meusel und Herr Luis sprachen während eines Abendessens über die Inhalte ihres täglichen Arbeitstages. Herr Meusel arbeitete zu diesem Zeitpunkt im Bereich Cloud-Hosting und Herr Luis war mit Datenanalysen von großen Datenmengen und dessen Methodiken beschäftigt. Sehr schnell entstand aus diesen beiden Gesprächsinhalten die Softwareidee, welche in diesem Dokument vorgestellt wird. Da Herrn Meusel und Herrn Luis der Aufwand dieser Lösung bewusst war und beide keine Expertise im Bereich Frontend-Entwicklung, bzw. Datenvisualisierung besaßen, stellten Sie die Idee ihren Mitstudenten vor. Herr Reuter zeigte dabei hohes Interesse und war vor allem vom englischsprachigen Teil des Projektes überzeugt. Die von den Dozenten definierten Vorgaben waren somit erfüllt und das Projekt konnte für das fünfte und sechste Semester angemeldet werden.

Die größte Hürde in einem Projekt von diesem Umfang ist die fehlende Zeit für die Projektarbeit. Alle drei Schüler führen die Weiterbildung nebenberuflich durch und müssen die Leistung neben einer regulären Arbeitszeit, Unterrichtseinheiten und Privatleben erbringen. Eine gut strukturierte Zeitplanung und Projektorganisation ist dabei unverzichtbar. Gleichzeitig muss jeder der Schüler auf sein eigenes Wohlbefinden achten. Ein zu schnelles Arbeitstempo würde einen Einzelnen unter der Last zusammenbrechen lassen und somit, wie auch bei einem zu langsamen Arbeitstempo, das Projektziel gefährden können.

In einer globalisierten Arbeitswelt stehen die Möglichkeiten von IT-Systemen und dessen verbundenen Werkzeugen, wie die Optimierungen von Arbeitsabläufen mit Hilfe von Analysen und Erkenntnissen, sehr stark im Vordergrund. Gleichzeitig beschäftigen sich Computersystem-Anbieter mit der optimalen Auslastung ihrer Serverfarmen (Hardware) und den damit verbundenen Kostenoptimierungen. Durch einen neu entstandenen Trend des Cloud-Hostings ist die optimale Auslastung der Hardwaremittel wichtiger denn je. Jedoch besitzt der Markt zum aktuellen Zeitpunkt noch keine Management-Software, welche eine voll- oder halbautomatische Optimierung von ganzen Rechenzentren oder Servergruppen übernehmen kann. Entsprechend versuchen Systemadministratoren weiterhin auf ihren zugewiesenen Systemgruppen eine optimale Leistung zu erzielen. Dabei fehlt diesen Personen eine Gesamtsicht auf Ihre oder andere Servergruppen, entsprechend fällt der Nutzen der getätigten Optimierungen im Vergleich sehr gering aus.

Das Problem der fehlenden Übersicht soll in dieser Projektarbeit gelöst und dokumentiert werden. Das genaue Ziel ist die Entwicklung von einem Softwaresystem aus drei Hauptkomponenten für den Einsatz in großen Rechenzentren für die Ausarbeitung einer Ressourcen-Übersicht. Diese soll primär von IT-Fachkräften einsehbar sein, sodass die Gesamtperformance des Rechenzentrums optimiert werden kann. Je nach Datenschutzeinwilligung der Kunden der einzelnen Computersystem-Anbieter können diese Daten auch sekundär für ein optimiertes Kundengespräch oder Marketing-Kampagnen eingesetzt werden. Jedoch handelt es sich dabei nur um ein Nebenprodukt und wird in dieser Projektarbeit und Dokumentation nicht betrachtet.

Die Methodiken der Projektorganisation, sowie der vereinzelte Zugang zu Literatur, erarbeiten sich die Projektmitglieder über ein Selbststudium, Inhalte aus dem Unterricht oder Erfahrungsberichte vom Auftraggeber. Dies beinhaltet auch die Beurteilung von den genutzten Quellen, welche für die Projektarbeit genutzt und zitiert werden. Es wird vorab bereits durch diese Wissensdatenbanken ein grober Projektvorgang herausgearbeitet, welcher jedoch auch in den nachfolgenden Seiten beschrieben ist.

[TM, NL, MR]

2 Projektvorstellung

Cloud Provider bieten verschiedenste virtuelle Instanzen auf physischen Hosts an. Dabei nutzt jeder virtuelle Server die vorhandenen Ressourcen des physischen Systems unterschiedlich. Hier kommt es, aufgrund der Mischkalkulation für die Ressourcen, zu einer Überbuchung (Overcommitment) des Hosts. Weil das Monitoring nicht ausreichend ist oder kein sinnvolles Placement implementiert ist (Placement beschreibt den Algorithmus, der einen Node ermittelt auf dem eine neue virtuelle Instanz angelegt wird), kommt es regelmäßig zu Leistungseinbußen. Für Kunden gibt es keine Transparenz über die ihm zugeteilten und durch ihn genutzten Ressourcen, weshalb auch keine ressourcenbasierte Abrechnung erfolgen kann. Teamleiter sind häufig mit der Effizienzsteigerung der Plattform beschäftigt und müssen die Auslastung steigern. Dies ist ohne detaillierte Berichte über die Auslastung nicht möglich.

In diesem Projekt soll eine funktionierende Open Source Software entwickelt werden, die sich in drei Teile gliedert:

- Die verschiedenen Ressourcetypen (CPU Zeit / Datendurchsatz / RAM Auslastung / Speicher Auslastung / Netzwerkdurchsatz) der einzelnen virtuellen Server müssen in einem sinnvollen Intervall periodisch ermittelt werden.
- Die Daten müssen aggregiert und gespeichert werden. Hierbei ist auf eine Skalierung auf mindestens 10.000 virtuelle Instanzen unter Berücksichtigung der Verfügbarkeit und Performance der Datenbank zu achten (Sharding oder Replikation, verteilt oder zentral, dokumentenbasiert oder relational).
- Diese Daten können dann dem Endanwender präsentiert werden (API und Web-UI). Hierzu wird eine Userstory-Erhebung unter den drei Anwendertypen Kunde, Administrator und Manager bei Partnerunternehmen durchgeführt, um gewünschte Algorithmen zur Visualisierung zu ermitteln (zum Beispiel Reports von freien oder überbuchten Nodes, grafische Auswertung für Kunden).

Dieses Projekt eignet sich besonders gut als Projektarbeit, da es in drei gleich große Teile gegliedert ist. Jeder dieser Teile ist eigenständig und wird einem Projektmitglied zugeordnet. Dies erleichtert die spätere Bewertung durch die Dozenten. [TM, NL, MR]

2.1 Projektteam

Das Projektteam besteht aus den drei Mitgliedern Marcel Reuter, Nikolai Luis und Tim Meusel. [TM, NL, MR]

2.1.1 Marcel Reuter

Herr Reuter beendete 2013 seine Ausbildung zum IT-Systemelektroniker und arbeitet seitdem bei der Firma EBF-EDV Beratung Föllmer GmbH. Er ist verantwortlich für die visuelle Schnittstelle des Projekts (Punkt 3). [MR]

2.1.2 Nikolai Luis

Herr Luis begann die Weiterbildung zum Techniker während seiner Ausbildung zum Fachinformatiker Anwendungsentwicklung, welche er im Januar 2016 beendete. Er arbeitet als BIG Data Analyst bei der Deutschen Telekom. Er ist verantwortlich für die Speicherung der Daten und die automatisierte Schnittstelle (Punkt 2). [NL]

2.1.3 Tim Meusel

Herr Meusel schloss seine Ausbildung zum Fachinformatiker Systemintegration 2012 ab. Aktuell arbeitet er als Systems Engineer bei der Host Europe Group. Er verantwortet die Ermittlung sowie Übertragung der Daten. [TM]

2.2 Auftraggeber

Der Ansprechpartner für dieses Projekt ist das Unternehmen Puppet Inc. (im folgenden Puppet) in der Rolle als Auftraggeber, welches von den Mitarbeitern Herrn David Schmitt und Herrn Steve Quin vertreten wird. Puppet ist Marktführer im Bereich der Konfigurationsmanagement-Software. Software dieser Art hilft Administratoren, sehr einfach Testumgebungen aufzubauen und auch Produktivumgebungen zu verwalten. Das Kernprodukt der Firma Puppet, welches ebenfalls puppet heißt, steht unter einer Open Source Lizenz und darf frei genutzt werden. Der Einsatz der Software erlaubt es dem Projektteam, verschiedenste Prototypen in kurzer Zeit zu bauen. Puppet entwickelt außerdem Software zum Testen. Dies vereinfacht das Qualitätsmanagement im Projekt. Puppet hat in der Vergangenheit bewiesen, mit

agiler Entwicklung und diversen Testverfahren umgehen zu können. Beides wird intensiv in deren Teams genutzt. Herr Quin und Herr Schmitt bieten tiefgreifendes Wissen zu den verschiedenen Programmen und Techniken, um das Projektteam zu unterstützen und zu beraten. [TM, NL, MR]

2.3 Aktuelle Situation

Im Abschnitt 2 wurden bereits die Intentionen des Projekts erläutert. In der Vergangenheit wurde schon mal versucht eine passende Softwarelösung zu entwickeln. OpenStack ist ein Softwareprojekt, welches große Mengen an Rechen- und Netzwerkkapazitäten, sowie persistenten Speicher in einem Rechenzentrum zu einer Public Cloud oder Private Cloud Umgebung zusammenfasst und orchestriert (vgl. [Fou16c]). Eines der Teilprojekte ist Telemetry. Das Ziel hiervon ist es, zuverlässig Daten von physischen und virtuellen Ressourcen zu ermitteln und verlässlich zu speichern. Die Daten sollen zur Analyse genutzt werden und Aktionen auslösen, wenn bestimmte Kriterien erreicht sind (vgl. [Fou16b]). Telemetry bringt leider mehrere Nachteile mit sich:

- Die Standarddatenbank für Telemetry war lange Zeit MongoDB. MongoDB ist eine dokumentenorientierte Datenbank. Darin werden Daten nicht in Tabellen strukturiert, sondern in eigenständigen Dokumenten. Jedes Dokument hat eine eigene Datenstruktur (zum Beispiel im JSON Format)(vgl. [16a]). Dies ermöglicht es, die Datenbank sehr einfach vertikal zu skalieren. Hierbei werden alle Dokumente auf mehrere Server verteilt. Schreib- und Lese-Anfragen können ebenfalls auf alle Server verteilt werden. Die Skalierung ist nahezu linear (vgl. [Mon16a],vgl. [Mon16b]). Die Grundidee ist sehr gut und eignet sich für besonders große Datenmengen oder Installationen die hochverfügbar sein müssen. Die Implementierung in MongoDB hat allerdings diverse Nachteile. Kyle Kingsbury hat intensiv MongoDB mit Jepsen getestet. Die Datenbank hat sich mehrfach als sehr unzuverlässig herausgestellt (vgl. [Kin16]). Die beiden folgenden Punkte disqualifizieren MongoDB für einen Einsatz als persistenten Datenspeicher, da die Persistenz nicht sichergestellt ist. Es wurde von Kyle Kingsbury bewiesen, dass:
  - MongoDB bei einem Schreibvorgang bestätigt, dass Daten persistent gespeichert wurden, dies allerdings nicht immer der Fall ist. Ein unbemerkter Datenverlust ist die Folge.
  - MongoDB in bestimmten Situationen die falschen Daten bei einer Leseanfrage zurückliefert.

- Das Telemetry Projekt hat diese Probleme ebenfalls erkannt und nach alternativen Datenbanken gesucht. Als keine vorhandene Lösung ihren Anforderungen entsprach, entschieden sie sich für die Entwicklung einer eigenen Datenbank: Gnocchi. Diese Datenbank ist noch in der Entwicklung und noch nicht für den produktiven Einsatz bereit.
- Es wird immer wieder von Skalierungsproblemen berichtet. Das CERN betreibt eine OpenStack-Installation und konnte die Probleme mit Telemetry nur mit immenser Hardware lösen. Die Kosten für diese Infrastruktur, als auch die Komplexität, sind viel zu hoch. Sie übersteigen das Budget der meisten Cloud-Umgebungen, wodurch diese unwirtschaftlich werden (vgl. [Bel+15]).
- Das Telemetry Projekt ist sehr stark in OpenStack eingebunden. Der Einsatz in anderen Cloud-Umgebungen ist nicht vorgesehen, da die einzelnen Dienste aus dem Telemetry Projekt weitere OpenStack Komponenten benötigten. Ein eigenständiger Betrieb ohne OpenStack ist für die Zukunft nicht geplant.

Aufgrund der langen Liste an Problemen ist Telemetry aktuell innerhalb einer kleinen OpenStack-Installation teilweise nutzbar, jedoch nicht wirtschaftlich in großen Umgebungen oder außerhalb von Openstack. Es ist nicht davon auszugehen, dass diese Probleme kurz- bis mittelfristig gelöst würden, da diese im Design und Architektur des Projekts liegen. Es gibt aktuell keine Alternativen zu Telemetry mit einem gleichwertigen Funktionsumfang. Somit entschied sich das Projektteam, eine Alternative zu entwickeln, die unkompliziert zu installieren ist und unabhängig von OpenStack arbeitet. [TM]

2.4 Anforderungen

Die Detailanforderungen werden getrennt für die drei Bereiche des Projekts im folgenden Abschnitt beschrieben. Diese ergeben sich aus dem Lasten- und Pflichtenheft sowie aus den Userstories der Partner. Für alle Lösungen gilt, dass sie eine aktive Community haben und unter einer Open-Source-Lizenz stehen. [TM]

2.4.1 Datenerfassungssysteme

Unterschiedliche Ausbaustufen für virtuelle Maschinen werden über mehrere verschiedene Ressourcen, die sich in ihrer Leistungsklasse unterscheiden, differenziert. Diese Typen müssen für jede virtuelle Maschine ausgelesen werden. Als Referenz

werden die Typen der größten deutschen Anbieter für virtuelle Maschinen genommen. Diese sind aktuell:

- Zugewiesener Arbeitsspeicher
- Anzahl/Leistung der Prozessorkerne
- Durchsatz des Datenträgers
- Anbindung an das Internet

Diese Werte müssen sowohl in den virtuellen Maschinen als auch auf dem Host ermittelt werden können. Im Bereich Managed Hosting hat der Hoster Zugriff in die virtuellen Instanzen und kann dort direkt sehr detailliert Daten ermitteln. Im klassischen vServer-Bereich (dem Bereitstellen des vServers als Produkt, nicht als Service) hat der Betreiber keinen Zugriff in die VM und muss Daten vom Host aus ermitteln. Auf dem Host muss zusätzlich die Gesamtauslastung der Ressourcen sowie der Zustand der Hardware ermittelt werden. Jede Virtualisierungssoftware bietet Schnittstellen, um Daten über die laufenden virtuellen Instanzen zu ermitteln. Dies gilt aber nur für Ressourcen, die die Virtualisierungssoftware direkt verwaltet. Außerdem ist auf dem Host nicht ersichtlich, wofür Ressourcen in den VMs genutzt werden. So kann zum Beispiel ermittelt werden, wie viel CPU-Zeit einer Instanz zur Verfügung gestellt wird, aber nicht wofür die Leistung genutzt wird (zum Beispiel Soft-IRQs, Interrupts, iowait). Um detaillierte Daten für eine bessere Analyse zu bekommen, muss deshalb auch in den virtuellen Maschinen eine Datenerfassung erfolgen, sofern Zugriff vorhanden ist.

Gesammelte Daten müssen lokal zwischengespeichert werden können, um einem Verlust bei Netzwerkstörungen zu vermeiden. Der Versand muss nach dem Push- und nicht nach dem Polling-Verfahren arbeiten. Somit kann eventbasiert gearbeitet werden. Dies verhindert Overhead im Netzwerk. Es werden somit nur dann Daten geschickt, wenn auch tatsächlich welche vorhanden sind. Beim Polling-Verfahren muss eine zentrale Instanz in konstanten Intervallen abfragen, ob neue Daten vorhanden sind. Eine Visualisierung in Echtzeit ist nicht gefordert, weshalb Daten lokal zwischengespeichert werden können, um sie gebündelt zu verschicken. [TM]

2.4.2 Datenhaltungssystem

Nachfolgende Anforderungen an das Datenhaltungssystem wurden vor der Projektphase im Team und auch in der ersten Rücksprache mit dem Auftraggeber definiert. Sie dienen als erster Leitpfaden der Softwarelösung und definierten sich wie folgt:

- Zunächst muss das Datenhaltungssystem für das Abspeichern von 10.000 Datensätzen in der Sekunde ausgelegt sein.
- Gleichzeitig muss das Datenhaltungssystem über Analysewerkzeuge verfügen, sodass auf den gespeicherten Daten eine fachliche Logik aufgebaut werden kann.
- Die daraus resultierenden Analyseergebnisse sollten anschließend über im Projekt definierte Schnittstellen an verschiedene Systeme weitergeliefert werden können. Entsprechend soll das Datenhaltungssystem die dafür notwendigen Treiber und Schnittstellen mitliefern.

Es soll in der Ziellösung eine stabile Schnittstelle im Gesamtsystem bilden und im Falle von einem Ausfall der beiden anderen Softwarekomponenten die Informationen weiterhin sicher und vollständig zur Verfügung stellen können. Eine konkretisierte Anforderungsübersicht an das Datenhaltungssystem kann dem Abschnitt 5.2 entnommen werden. [NL]

2.4.3 Datenvisualisierung

Die Datenvisualisierung soll dem Benutzer eine Übersicht über die verbrauchten oder freien Ressourcen, wie zum Beispiel CPU-Auslastung oder RAM-Auslastung einzelner virtueller oder physischer Maschinen bereitstellen. Bei der Datenvisualisierung werden mit unter sensible Daten verarbeitet, weshalb hier auf die Sicherheit in anbetracht auf die Authentifizierung, Authentisierung und Autorisierung geachtet werden muss. Innerhalb der Projektmeetings wurde die Sicherheit der Daten diskutiert, dabei ging es hier speziell darum, wie die Daten am sichersten geschützt werden. Dabei wurde festgelegt, das eine Benutzerverwaltung mit Rechtevergabe, sowie die Anbindung an die Datenbank über eine API Schnittstelle, (siehe Abschnitt 5.3.3) realisiert werden muss. Dabei muss die Datenvisualisierung Abfragen über eine API stellen können. Dies ermöglicht es, den Zugriff nur für die Datenvisualisierung auf die sensiblen Daten gewährleisten zu können. Die Webseite soll Administratoren bei einer Neukonfiguration von virtuellen Maschinen unterstützen und visualisiert die Auslastung der einzelnen physischen Systemen in Form von Graphen (siehe Abschnitt 5.4.1). Der Administrator kann so gezielt die einzelnen physischen Maschinen besser auslasten und verteilen.

Eine genaue detaillierte Auflistung aller Kriterien an die Datenvisualisierung, die für eine spätere Nutzwertanalyse verwendet wurden, kann dem Abschnitt 5.4.2 entnommen werden. [MR]

3 Projektmanagement

Die Liste an verfügbaren Managementmethoden ist lang. Methoden wie PRINCE2 oder Lean Management kommen aus dem Bereich des Projektmanagements und sind seit vielen Jahren auch im Bereich der Softwareentwicklung vertreten. Hinzu kommen Vorgehensmodelle aus der Softwareentwicklung selbst, wie das Wasserfallmodell oder das Spiralmodell. In den letzten Jahren gab es einen Wandel hin zu agiler Entwicklung [Bec+01]. Es hat sich gezeigt, dass sich in der schnelllebigen Informationswelt Anforderungen an Software regelmäßig ändern, auch während der Entwicklungs- und Planungsphase und nicht erst im späteren Betrieb. Außerdem gibt es in den meisten Projekten unvorhergesehene Zwischenfälle. Dazu gehören unter anderem:

- Sicherheitslücken in verwendeten Bibliotheken werden entdeckt. Diese müssen oftmals aufwendig aktualisiert werden.
- Die Zeiteinschätzung für die Implementierung von Funktionen oder die Behebung von Fehlern benötigt wesentlich mehr oder weniger Zeit als geschätzt.
- Die Dokumentation einer Bibliothek, anhand dessen eine Zeitschätzung gemacht wurde, ist fehlerhaft. Die Bibliothek verhält sich anders als in der Dokumentation beschrieben und muss genauer begutachtet werden.
- Die eingesetzte Software erfüllt ihren Zweck, benötigt aber viel zu viel Leistung. Hier muss nun die Software analysiert werden oder man migriert auf eine Alternative.
- Bei der Verwendung mehrerer Hard- und Softwarekomponenten kommt es zu unerwarteten Inkompatibilitäten.

Die Techniken und Vorgehensweisen der agilen Softwareentwicklung, versuchen dem entgegen zu wirken. Sie zeichnen sich durch drei Merkmale aus:

- Die Arbeit erfolgt iterativ und transparent
- Der bürokratische Mehraufwand ist sehr gering
- Die Methoden passen sich flexibel an das Projekt und an Änderungen an

[TM]

3.1 Agile Softwareentwicklungsmethoden

Die Firma Puppet entwickelt seit mehreren Jahren erfolgreich Software. Sie steht dem Projektteam mit hilfreichen Tipps und Schulungen zur Seite. „Agile Softwareentwicklung“ gilt als Oberbegriff für alle Techniken und Vorgehensweisen in dem Bereich. Aus diesem kann man sich die Methoden heraussuchen, die am besten zum Projekt passen. In den letzten Jahren haben sich daraus die beiden Softwareentwicklungsmodelle Scrum und Kanban abgeleitet. Diese nutzen Teile der agilen Softwareentwicklung. [TM]

3.1.1 Scrum

Das Hauptaugenmerk von Scrum liegt auf den Sprints. Dies sind die Abschnitte in denen gearbeitet wird. In der Regel beträgt ein Abschnitt zwei Wochen. Am Anfang von jedem Arbeitstag des Abschnittes muss eine Besprechung erfolgen. Die Besprechungen, „Daily Standup“ genannt, ist genau 15 Minuten lang und soll im stehen abgehalten werden. Die Gesprächszeit muss gleichmäßig auf alle Teilnehmer verteilt werden. Hierbei spielt es keine Rolle, wie groß das Team ist. Bei jeder Teamstärke beträgt die Gesamtzeit nur 15 Minuten. Dies ist ein großes Problem für Teams die sehr groß sind oder asynchron arbeiten. Aufgrund der Vollzeitbeschäftigung aller Mitglieder in diesem Projekt, in Kombination mit dem Abendschulunterricht, erfolgen viele Arbeiten am Projekt asynchron. Es ist zeitlich nicht realisierbar, dass an jedem Tag jedes Teammitglied am Projekt arbeitet. Die Durchführung von täglichen Besprechungen ist somit nicht möglich. Scrum erlaubt es nicht, einzelne Elemente des Systems nicht zu nutzen oder zu modifizieren. Somit kann in diesem Projekt nicht nach Scrum gearbeitet werden (vgl. [Haa16]). [TM]

3.2 Agile Vorgehensweise im Projekt

Die Projektzeit wird in zweiwöchige Abschnitte, Sprints genannt, eingeteilt. Am Anfang jedes Sprints erfolgt ein Rückblick auf den vergangenen Sprint (Retrospective). Es wird kurz besprochen was besonders positiv oder negativ lief und ob die Projektmitglieder zufrieden sind. Bei Unstimmigkeiten im Team oder bei negativen Ereignissen muss der Projektleiter sich dieser annehmen. Seine Aufgabe ist es ein positives Arbeitsklima zu schaffen und nach Möglichkeit wiederkehrende, negative Punkte zu beseitigen.

Wenn ein Mitglied einen besonderen Fortschritt im Projekt erzielt hat, wird dieser

kurz präsentiert (Review). Hierbei wird keine vollständige Präsentation mit Folien erwartet, sondern eine kurze Erklärung der erledigten Arbeit mit einer Demonstration. Jedes Projektmitglied darf selbstständig entscheiden, ob es seine Arbeit hier präsentiert. Review und Retrospective werden oft als „R & R“ abgekürzt.Am Ende jedes Sprints erfolgt die Planung für den kommenden Sprint (Planning). Es wird überlegt, welche Arbeit erledigt werden muss, danach erfolgt die ungefähre Schätzung des Zeitaufwandes. Für jede Aufgabe wird ein Ticket in der Projektmanagementsoftware erstellt. Das Projektteam hat sich für „JIRA“ entschieden, da dies aktuell am verbreitetsten in der Wirtschaft ist. Neue Featurewünsche werden als Userstory angelegt. Dies ist ein Ticket in dem aus Sicht der anfragenden Person ihr Wunsch beschrieben ist. Für wichtige Ereignisse werden Meilensteine festgelegt. Jeder darf neue Tickets zu jeder Zeit anlegen. Diese landen dann im Backlog. Dies ist der Sammelbegriff für ausstehende Tickets, welche keinem Sprint zugeordnet sind. Es wird nicht an Tickets gearbeitet, welche nicht im aktuellen Sprint sind. [TM]

4 Schnittstellen im Projekt

Zwischen den Softwarekomponenten im Projekt werden Daten ausgetauscht. Dies erfolgt über definierte Schnittstellen (auch Interfaces genannt). Im Folgenden sind die einzelnen Schnittstellen zwischen den Komponenten und von den Komponenten nach außen erklärt. [TM]

4.1 Datenerfassungssysteme

Die Datenerfassungssysteme stellen zwei Schnittstellen zur Verfügung. Sie kommunizieren mit dem lokalen System über eine bidirektionale Schnittstelle. Die Datenerfassung wird für die meisten Ressourcetypen über Polling realisiert. Hierzu fragt das Datenerfassungssystem periodisch die Schnittstelle des Hosts nach neuen Daten. Abfrageschnittstelle des Hosts wird durch den Linux-Kernel bereitgestellt, von ihr kann ausschließlich gelesen werden. Die Erfassungssysteme arbeiten zusätzlich eventbasiert. Hierbei schickt der Kernel die neuen Informationen direkt zur Schnittstelle der Datenerfassungssysteme. Der Kernel ist der kritischste Teil eines laufenden Computers. Er hat volle Rechte auf alle Hardwareschnittstellen, weshalb die Interaktion mit ihm besonders geschützt und minimal sein muss. Programmierfehler in der Vergangenheit erlaubten es mehrfach, auf Schnittstellen auf den Kernel zu schreiben, obwohl das Interface nicht beschreibbar sein sollte. Der Kernel selbst bietet nur lokale Schnittstellen, aber über weitere Software werden diese indirekt im Netzwerk bereitgestellt.

Die zweite Schnittstelle der Datenerfassungssysteme befindet sich innerhalb des Projektes und interagiert mit der Datenbank. Diese Schnittstelle ist aus Sicherheitsgründen unidirektional, um den Kernel zu schützen. Die Erfassungssysteme können also nur Daten über das Netzwerk verschicken, jedoch akzeptieren sie keine Anfragen, welche über die Netzwerke eingehen.

Bei beiden Schnittstellen der Datenerfassungssysteme handelt es sich um Binärschnittstellen. Hierbei werden Daten nicht in einer für Menschen lesbaren Form übertragen, sondern in einem Binärprotokoll. Diese Protokolle sind besonders effizi-

ent in der Datenübertragung und Datenverarbeitung. Es ist nicht erforderlich, dass Menschen mit den beiden Schnittstellen kommunizieren. [TM]

4.2 Datenhaltungssystem / API

Das Datenhaltungssystem, sowie die mit inbegriffene API, ist eine zentrale Schnittstelle des Gesamtsystems. Im Gegensatz zu dem Datenerfassungssystem, welches nur über die Informationen eines einzelnen physischen Servers verfügt, besitzt das Datenhaltungssystem eine Gesamtsicht über alle angebundenen Informationsquellen. Darunter fallen zunächst alle am System angebundenen physischen Server, welche aktiv die Nutzungsdaten an das Datenhaltungssystem liefern. Jedoch kann dies auf Wunsch des jeweiligen Kunden auch um weitere Informationen erweitert werden, ein Beispiel wären die Kundendaten mit dem jeweiligen Bestand. In der Ziellösung wird diese Art von Erweiterung jedoch zunächst nicht beachtet und wurde auch nicht vom Auftraggeber angefordert, sodass eine Anpassung in einem Folgeauftrag erledigt werden müsste.

Die API ist ebenfalls eine essentielle Schnittstelle des Gesamtsystems. Durch die direkte Anbindung an das Datenhaltungssystem kann sie zeitkritische Daten innerhalb kürzester Zeit an Drittsysteme liefern. Dabei arbeitet die API oftmals auf einer Webapplikation, sodass diese von einem beliebigen Endgerät erreicht werden kann. Zudem liefert sie die Daten in Form von Text aus, welcher nach den Datenformaten des JSON oder XML aufgebaut ist. Die beiden Datenformate sind die weitverbreitetsten Formate auf dem Markt und können in vielen Datenvisualisierungs-Systemen ohne zusätzliche Entwicklungskosten angebunden werden. Sollte ein Drittsystem diese Formate nicht unterstützen, so ist der Implementierungsaufwand bei einem erfahrenen Entwickler ebenfalls gering, da diese oftmals mit den Datenformaten bereits in Kontakt gekommen ist und keine neuen Strukturen erlernen muss. [NL]

4.3 Datenvisualisierung

Eine Schnittstelle auf einer Webseite dient in erster Linie zur Anzeige von Daten in Form von Graphen. In diesem Projekt werden bewusst zwei Schnittstellen verwendet, damit die Daten innerhalb des Datenhaltungssystem geschützt sind, wie auch die Sicherheit der eigentlichen Datenvisualisierung gewährleistet werden kann. Im nachfolgendem werden beide Schnittstellen erläutert.

Die erste Schnittstelle der Webseite ist ein standardisierter Treiber, welchen man

verwenden kann, um auf ein Datenhaltungssystem zuzugreifen. Diese liefert der Webseite alle nötigen Funktionen und Werkzeuge zur Datenextraktion, Datenmanipulation und der Rechteverwaltung auf einem Datenhaltungssystem. In der Ziellösung von diesem Projekt ist ein rein lesender Datenhaltungszugriff für die Webseite vorgesehen. Dies wird durch den Datenhaltungssystem-Administrator gewährleistet.

Neben der Verwendung eines Treibers in Verbindung mit einem Berechtigungskonzept, kann auch ein API-System zum Einsatz kommen. Die Vorteile, sowie die Rolle dieser Schnittstelle innerhalb dieses Projektes, können dem Abschnitt 5.3 entnommen werden.

Die zweite Schnittstelle dient zur Anzeige und Ausgabe der durch das Datenhaltungssystem erhaltenen Daten. Ohne diese Schnittstelle würden Anwender, welche mit der Webseite arbeiten möchten, nur unstrukturierte Daten erhalten und können diese nicht zur weiteren Analyse verwenden. Der Anwender erhält somit einen direkten und schnellen Überblick in Form eines Graphens, siehe Abschnitt 5.4.1 oder Diagramms über die gesamten Datenressourcen der Infrastruktur. Diese können bei Bedarf vom Anwender oder dem Administrator exportiert und als Abbild festgehalten werden. Hiermit können anschließend Personen, welche keinen direkten Zugriff auf das System besitzen, Analysen durchführen oder Statistiken beurteilen. Die Übertragung der Daten zwischen den Benutzern und dem Webserver erfolgt mit HTTPS. Dabei wird die Kommunikation zwischen Browser und Webserver verschlüsselt, sodass diese durch Dritte nicht abgegriffen werden kann. Das abgreifen der Kommunikation wird im Fachkreis auch Man-in-the-middle-Angriff genannt.

[MR]

5 Analyse von Softwarekomponenten

5.1 Datenerfassungssysteme

Im Abschnitt 2.4.1 wurde bereits auf die einzelnen Ressourcetypen eingegangen, welche erfasst werden müssen. Um die verschiedenen Datenerfassungssysteme im Hinblick auf die Ressourcetypen zu evaluieren, muss zunächst geklärt werden was eine Metrik, ein Trend und eine Zeitreihe (englischer Fachausdruck: Timeseries) ist. [TM]

5.1.1 Begriffsklärung und Anforderungen

Eine Metrik ist die Kombination aus:

- Dem Wert eines bestimmten Ressourcetypen zu einer bestimmten Zeit (Momentaufnahme)
- Dem Zeitstempel der Aufnahme
- Dem Namen des Ressourcetypen

Eine Metrik wird periodisch als Ganzzahl ermittelt. Die Kombination mehrerer Werte einer Metrik ergibt eine Timeseries (auch „Datenreihe“ oder „Abfolge“ genannt). Um Speicherplatz zu sparen, können die Werte aggregiert werden. Hierbei werden für einen bestimmten Zeitraum einer Timeseries die Werte genommen und folgende Funktionen angewendet:

- `min()`: Ermittlung des niedrigsten Wertes
- `max()`: Ermittlung des höchsten Wertes
- `count()`: Addieren aller Werte
- `sum()`: Alle Werte werden aufaddiert. Diese Berechnung ist nicht bei jedem Ressourcentyp sinnvoll, allerdings wird das Ergebnis für weitere Berechnungen benötigt
- `avg()`: Berechnung des Durchschnitts mit den Ergebnissen aus `count()` und `sum()`

- `95pct()`: Berechnung des Durchschnitts, zuvor werden die Werte nach Größe sortiert und die höchsten 5% ignoriert

Außerdem gibt es die Möglichkeit, die erhobenen Daten in Relation zur Zeit zu setzen. Hierbei wird die Differenz zwischen zwei aufeinander folgenden Messpunkten betrachtet. Beispiel: Das Betriebssystem misst die geschriebene Datenmenge auf einem Datenträger seit dem Startvorgang in Bytes. Interessant für den Anwender ist aber der Datendurchsatz in MB/s oder GB/s. Dies kann wie folgt berechnet werden:

Ab Systemstart wird in regelmäßigen Zeitabständen Δt ausgelesen, welche Datenmenge seit Systemstart geschrieben und gelesen wurde. Bezeichne m_i den i-ten Messwert. Dann kann die durchschnittliche Durchsatzrate D_i zwischen zwei Messpunkten m_i und m_{i-1} bestimmt werden durch:

$$D_i = \frac{m_i - m_{i-1}}{\Delta t}.$$

Danach werden nur die Ergebnisse einer oder mehrerer Funktionen gespeichert und die eigentlichen Daten verworfen. Das Aneinanderreihen mehrerer aggregierter Werte bildet einen Trend (auch History Trend genannt). Trends werden oftmals visualisiert. Hierbei lässt sich ressourcensparend ein langer Verlauf der Metrik erkennen. Basierend auf vorhandenen Trends kann auch eine „Trend Prediction“ erstellt werden. Hierbei werden Daten der Vergangenheit analysiert und auf eine mögliche Zukunft vorausberechnet.

„Tiered Trends“ bezeichnet eine Datensammlung auf welche mehrfach die oben genannten Funktionen angewendet wurden. Die benötigte Granularität ändert sich oftmals mit dem Alter der Daten. Beispiel: In den ersten 24 Stunden benötigt man Daten mit einer Auflösung von 30 Sekunden. In den darauffolgenden 2 Wochen reichen Trends mit einer Auflösung von 5 Minuten und für die folgenden 3 Monaten reicht eine Auflösung von 30 Minuten.

Hier nimmt man sich Daten die älter als 24 Stunden sind und teilt diese in Einheiten von 5 Minuten. Hierauf werden die Funktionen angewendet, die originalen Daten gelöscht und die Ergebnisse gespeichert. Parallel wird regelmäßig nach bereits vorhandenen Trends geprüft, welche älter als 2 Wochen sind. Diese werden wieder in Einheiten von 30 Minuten aufgeteilt und die Funktionen erneut angewendet. Somit wurden „Tiered Trends“ gebildet.

Das Bilden von Trends kann schon während der Datenerfassung erfolgen, indem die gesammelten Werte lokal zwischengespeichert werden und ausschließlich die gebildeten Trends zur Datenhaltung geschickt werden. Dies ist besonders effizient, da jeder Server nur eine geringe Menge an Trends zu berechnen hat. Es ist jedoch er-

forderlich, dass der Client alle Daten zumindest so lange vorhält, bis alle benötigten Trends gebildet sind. Im obigen Beispiel zu „Tiered Trends“ sind dies 2 Wochen. Oftmals gibt es die Anforderung Trends ad hoc zu bilden oder die Intervalle regelmäßig zu verändern. Hierzu muss eine neue Konfiguration für die Datenerfassungskomponente auf jedem Server erstellt werden, da diese keine automatisierte Schnittstelle bieten.

Ebenfalls kann die Generierung der Trends auf dem Datenbanksystem erfolgen. Dies benötigt Rechenkapazitäten, bringt aber diverse Vorteile:

- Trends können nicht nur über eine Metrik eines Servers erstellt werden, sondern auch über mehrere Server hinweg
- Trends können erstellt werden, wenn diese angefordert werden (Eventbasiert)
- Trends können über verschiedenste Zeitrahmen erstellt werden. Außerdem lässt sich dieser einfacher ändern, da er nur zentral konfiguriert ist und nicht auf jedem Server

Aufgrund der Flexibilität der Trendgenerierung auf der Datenbank entschied sich das Projektteam dazu, die Generierung von den Servern auf die Datenbank zu verlagern. Die Fähigkeit Trends generieren zu können ist somit kein Evaluationskriterium für Datenerfassungssysteme.

Im Folgenden werden die hier gelisteten Softwareprodukte evaluiert. Die Liste basiert auf dem Lasten- und Pflichtenheft:

- Coreutils
- atop
- collectd
- zabbix-agent
- python-diamond
- sysstat
- Logstash
- Riemann

[TM]

5.1.2 Coreutils

Coreutils ist eine Sammlung von Standardwerkzeugen auf jedem Linux-Betriebssystem. Sie bieten einfache Möglichkeiten zur Editierung von Dateien. Coreutils ist mit der GPL3 lizenziert (vgl. [Fre16]) und zählt somit als Open-Source-Software. Verwaltet wird die Programmsammlung aktuell von der Free Software Foundation. Der Linux Kernel stellt Informationen über schreibgeschützte Dateien bereit. In Kombination mit der Programmsammlung procps kann auf die Informationen des Kernels zugegriffen werden, um Metriken zu ermitteln (vgl. [gro16b]). Über das Programm `free` oder die Datei `/proc/meminfo` erhält man dann die Menge des vorhandenen sowie zur Zeit benutzten Arbeitsspeichers. Die Anzahl der Prozessorkerne liefert das Programm nproc oder die Datei `/proc/cpuinfo`. Über die Datei `/proc/diskstats` erhält man Informationen über die Anzahl der Schreib- und Lesevorgänge auf allen Datenträgern. Außerdem listet die Datei die geschriebene als auch gelesene Datenmenge. Die Informationen beziehen sich auf die Zeit seit dem letzten Startvorgang des Servers bis zum Ausgeben der Datei. Ein kontinuierliches Ausgeben ermöglicht die Berechnung des Durchsatzes pro Sekunde. Diese Aufgabe wird von der Datenbank übernommen und ist näher im Abschnitt 5.1.1 erklärt. Das gleiche gilt für die Datei `/proc/net/dev`. Sie listet alle Netzwerkadapter des Systems und die übertragene Datenmenge (Senden und Empfangen) pro Adapter. Mit Coreutils und Linux Bordmitteln ist es möglich alle benötigten Informationen lokal auszulesen. Die Aufbereitung der Daten muss allerdings selbst erledigt werden.

Unter Listing 13.15 gibt es ein Beispiel, welches alle 30 Sekunden die gesendeten sowie empfangenen Daten des Netzwerkadapters `enp1s0` in Bytes ausliest. Die generierte Ausgabe ähnelt einer CSV Datenstruktur und ist unter Listing 13.16 zu sehen.

Für das Auslesen all dieser Daten werden kleine Skripte wie das im Figure 13.15 benötigt. Um die Daten über das Netzwerk zu verschicken, werden weitere Skripte benötigt. Diese Sammlung an Skripten kann auf einem Server ausgeführt werden, um seine lokalen Statistiken zu erfassen oder direkt in virtuellen Maschinen. libvirt, eine Bibliothek zum Verwalten virtueller Maschinen, stellt auf einem Hostsystem Informationen über die lokalen Instanzen bereit. libvirt kümmert sich hier um die Interaktion mit diversen Hypervisors und bietet eine einheitliche Schnittstelle an. Dies ermöglicht es Coreutils, vom Host aus Daten der VMs zu ermitteln. [TM]

5.1.3 atop

Das Linux-Kommandozeilenprogramm atop ist zum Visualisieren von Prozessen und des Systemzustands. Es ist unter der GPL lizenziert und fällt damit unter Open-

Source-Software. Der aktuelle Maintainer ist Gerlof Langeveld. Die erste stabile Veröffentlichung war im Dezember 2001. Seitdem wird das Programm kontinuierlich entwickelt. Es ist darauf ausgelegt, im Vollbildmodus eines Terminals zu arbeiten (siehe 13.1 und 13.2 für Screenshots). Neben der Echtzeitanalyse bietet atop aber auch die Möglichkeit, im Hintergrund periodisch eine Logdatei zu schreiben. Hier wird ein selbst entwickeltes Binärformat genutzt. Dies kann von atop aber auch zurück in ASCII umgewandelt werden. Dann erhält man alle Infos, die auch die interaktive Version anzeigt. Im Standardverhalten werden alle 10 Minuten Logs geschrieben, dieses Intervall ist allerdings modifizierbar. Das ASCII Format beruht nicht auf bekannten Standards und lässt sich deshalb nicht automatisiert weiterverarbeiten (siehe 13.1 für eine exemplarische Ausgabe). Die Daten müssen, ähnlich wie bei Coreutils, vor einer Speicherung in einer zentralen Datenbank erst noch bearbeitet werden. Es ist keine Funktionalität eingebaut, um gesammelte Daten über das Netzwerk auszutauschen, dies muss ebenfalls nachgerüstet werden. Es ist keine Funktionalität vorhanden, um Daten auf einem Host von virtuellen Maschinen zu ermitteln. Außerdem kann die Software nicht erweitert werden (vgl [Lan16]). [TM]

5.1.4 collectd

Das Projekt collectd wurde von Florian Forster gestartet. Die erste stabile Version veröffentlichte er im Juli 2005. Seitdem erfolgt eine stetige Entwicklung mit mehreren neuen Veröffentlichungen pro Jahr. collectd bietet eine Vielzahl von Möglichkeiten, um Daten auf einem System zu ermitteln und diese in diversen Formen über das Netzwerk auszutauschen oder lokal abzulegen. Die Software ist in C geschrieben.

Wie im Listing 13.11 zu sehen, haben 415 Leute 9059 Änderungen im Versionsverwaltungssystem durchgeführt. collectd steht unter der MIT Lizenz und zählt somit als Open-Source-Software. Es kann flexibel mit Plugins erweitert werden. Hierbei wird zwischen mehreren Kategorien unterschieden (vgl. [gro16a]):

- Read, Plugins, die Daten aus zusätzlichen Quellen lesen
- Write, Plugins, die erfasste Daten in einem bestimmten Format lokal speichern
- Binding, ein Metaplugin, welches den Interpreter bestimmter Skriptsprachen einbindet. Dies erlaubt es, eigene Plugins in beliebigen Skriptsprachen zu schreiben
- Logging, erlaubt es, collectd interne Fehler und Warnungen in verschiedenen Formen zu speichern und auszugeben

Das Sammeln aller benötigter Daten wird von collectd unterstützt. Es kann lokale

Daten auf einem Hostsystem oder in einer virtuellen Maschine ermitteln. Außerdem kann es mit Hilfe des `Virt` Plugins auf einem Host auch alle notwendigen Informationen über virtuelle Instanzen abrufen(vgl. [gro15]). [TM]

5.1.5 zabbix-agent

Die Firma Zabbix LLC wurde 2005 gegründet. Sie pflegt die Open-Source-Produkte zabbix-agent (im folgenden Abschnitt Agent genannt), zabbix-server und zabbix-frontend (siehe auch Abschnitt 5.4.3 für die Analyse der Datenvisualisierung). Alle Produkte sind unter der GPL2 lizenziert. Der Agent ist in C geschrieben. Er bietet die Möglichkeit, diverse Daten direkt aus dem lokalen System auszulesen und an den zabbix-server zu senden. Das Webinterface ist eine PHP-Applikation, welche die Daten visualisiert und direkt aus der Datenbank liest. Der Agent kann mit externen Skripten erweitert werden. In Kombination mit Coreutils kann er alle benötigten Informationen auslesen. Die Community bietet dafür fertige Skripte an (vgl. [Lee15]).

Der Agent benutzt ein eigenes Binärprotokoll zur Kommunikation. Dieses wird nicht von anderen bekannten Programmen unterstützt. Deshalb muss der Agent zwangsläufig mit dem hauseigenen zabbix-server genutzt werden. Es ist nicht möglich den Server, beziehungsweise die unterstützten Datenbanken, redundant auszulegen. Dies ist eine Anforderung für die Datenhaltungssysteme. Somit ist der zabbix-agent nicht im Projekt nutzbar (vgl. [LLC16]). [TM]

5.1.6 python-diamond

Python-diamond (auch Diamond genannt) ist ein in Python geschriebener Dienst. Er kann auf Linux Systemen diverse Daten erheben und diese über das Netzwerk austauschen. Die erste Veröffentlichung ist vom Juli 2012. Seitdem gibt es unregelmäßig neue Updates. Das letzte Release ist vom November 2016. Die Open-Source-Software steht unter der MIT Lizenz (vgl. [gro17]).

Wie im Listing 13.12 zu sehen gibt es 2922 Beiträge im Versionskontrollsystem von 292 verschiedenen Personen. Diamond kann alle benötigten Daten lokal und vom Hypervisor über seine VMs ermitteln. Über das Netzwerk werden diese im Graphite Format geschickt. Über Plugins sind auch weitere Formate möglich. Der Backlog bei Diamond ist sehr hoch. Die Software hat aktuell 88 offene Issues im Versionsverwaltungssystem. Dies ist eine Mischung aus gemeldeten Fehlern und Feature Requests. Außerdem gibt es 139 offene Pull Requests (siehe Listing 13.3.

Dies sind Änderungen am Quellcode, welche die Community erstellt hat, aber noch nicht eingepflegt worden sind. [TM]

5.1.7 sysstat

Die Softwaresammlung sysstat ist eine Kollektion an kleinen Programmen zur Performance-Analyse unter Linux. Die Sammlung steht unter der GPL2 Lizenz. Das älteste verfügbare Release ist die Version 5.0.5 vom Juni 2004. Seitdem erscheinen jedes Jahr mehrere Updates. Entwickelt wird das Projekt von (vgl. [God16a]). Die Liste der Features von sysstat ist sehr umfangreich. Neben den geforderten Kriterien zur lokalen Datenerfassung liefert es auch viele weitere Details. Hierzu zählen zum Beispiel Statistiken zum Verhalten des Arbeitsspeichers. Die Interrupt-Verarbeitung des Prozessors und detaillierte Netzwerkinformationen. Ermittelte Daten können als CSV, XML oder JSON gespeichert werden (vgl. [God16b]).

Die Software sysstat bietet keine Möglichkeit, um Daten von virtuellen Maschinen über den Host zu ermitteln. Die bereitgestellten Informationen sind aber in einer sehr guten Struktur dank des XML-Schemas. Das Einbinden in andere Programme und das Interpretieren ist somit sehr einfach. Es werden nicht alle Anforderungen von sysstat erfüllt, es kann aber eventuell als Ergänzung zu anderen Programmen genutzt werden. [TM]

5.1.8 Logstash

Logstash wird von der Firma Elasticsearch BV entwickelt. Zusammen mit Elasticsearch (siehe auch Abschnitt 5.2.2) und Kibana bilden sie den ELK-Stack. Die Open-Source-Software steht unter der Apache License. Logstash arbeitet nach dem modularen EVA Prinzip.

- Die gewünschten Daten werden in vordefinierten Formaten eingelesen.
- Die Daten werden transformiert und gefiltert.
- Das Ergebnis wird zu einem oder mehreren Endpunkten geschickt.

Alle drei Schritte können mit über 200 Plugins erweitert werden (vgl. [BV16c]). Logstash ist für die Verarbeitung von Ereignissen ausgelegt. Diese können direkt aus diversen Quellen gelesen werden. Hierzu gehören unter anderem Netzwerksockets, Textdateien, Syslog Streams und Message queues. Bei der Transformation werden die Daten neu serialisiert und in ein anderes Format übertragen. Hier können zum Beispiel zeilenweise eingelesene Ereignisse aus einer Textdatei in das JSON Format

überführt werden. Dabei werden die unstrukturierten Daten mit Filtern zerlegt und in Typen eingeteilt, um strukturierte Daten zu erhalten. Aus einem langen String wird dabei die eigentliche Nachricht, das Datum, Dringlichkeit und andere Typen erkannt. Bereits strukturierte Daten können auch einfach in eine andere Struktur überführt werden. Zum Abschluss werden die Daten ausgegeben. Dies kann passieren, indem unter anderem in einen Cache Service, eine Message Queue oder direkt in eine Datenbank geschrieben wird.

Logstash ist nicht auf das Einlesen von Metriken ausgelegt (Ganzzahlen), sondern für Logs in Textform. Über ein eigenes Plugin ließe sich aber auch das effiziente Einlesen von Metriken realisieren. Mit der Vielzahl an Transformationen und Ausgabemöglichkeiten ist Logstash sehr flexibel und passt sich an jede Situation an. Dadurch dass Daten über mehrere Ausgänge parallel verarbeitet werden können, ist die Software sehr zukunftssicher. Bei geänderten Anforderungen lassen sich die Daten zum Beispiel anstatt in eine SQL-basierte Datenbank in eine No-SQL-Datenbank schreiben. [TM]

5.1.9 Riemann

Riemann ist ein junges Softwareprojekt. Das erste Release erfolgte im März 2012. Die Open-Source-Software steht unter der Eclipse Public Lizenz Version 1. Die aktuellste Veröffentlichung trägt die Versionsnummer 0.2.12 und wurde im Dezember 2016 veröffentlicht. Das Versionsschema von Riemann hält sich an die Regeln von Semantic Versioning, somit sind erst dann Releases zum produktiven Einsatz empfohlen, wenn sie die Nummer 1.0.0 oder höher enthalten. Der Autor ist Kyle Kingsbury, welcher auch Jepsen geschrieben hat. Der Fokus der Software liegt auf der Verarbeitung von Ereignissen, auch Events genannt (vgl. [Kin17]). Dazu wurde eine eigene Struktur entworfen mit mehreren Feldern (Siehe Tabelle 13.1). In der Konfigurationsdatei werden Streams definiert. Diese laufen immer nach dem gleichen Schema ab:

- Initialisierung eines neuen Streams mit dem Schlüsselwort `streams`
- Ein Filter mit dem Schlüsselwort `where`, hier kann mit beliebig vielen Filterregeln auf alle eingehenden Events geprüft werden. Mehrere Regeln können nach den Definitionen der booleschen Algebra verknüpft werden (`and` im folgenden Beispiel).
- Zum Schluss wird eine Aktion aufgelistet. Das Schlüsselwort ist immer der Name der Aktion (`email` im folgenden Beispiel). Diese wird auf jedes Event ausgeführt, welches durch alle Filter eines Streams gekommen ist. Jeder Aktion können Optionen mitgegeben werden.

Unter Listing 13.17 wird die Temperatur des ersten Prozessorkerns überprüft. Sobald diese bei 75°C oder höher liegt, wird eine E-Mail an tim@bastelfreak.de geschickt.

Riemann besitzt eine lange Liste an vorhandenen Aktionen, die man nutzen kann. Dazu gehört auch der Export von Events in diverse Datenbanken. Dazu gehören unter anderem Elasticsearch, graphite, influxdb, kariosdb, opentsdb, prometheus und generische UDP und TCP Outputs.

Während der Analyse von Riemann hat sich herausgestellt, dass Riemann zwar besonders gut Daten verarbeiten kann, allerdings keine Möglichkeit mitbringt, diese selbst zu ermitteln. [TM]

5.1.10 Zusammenfassung

Unten aufgeführt ist eine Gegenüberstellung der verschiedenen Lösungen und deren Möglichkeiten. Verglichen wird hier die Fähigkeit der Datenermittlung für die im Abschnitt 2.4.1 definierten Typen. Betrachtet wird zuerst nur die Ermittlung von lokalen Daten in einer virtuellen Maschine oder auf einem Hostsystem.

Programme	Zugewiesener RAM	CPU Kerne	Datenträger	Netzwerk
Coreutils	✓	✓	✓	✓
atop	✓	✓	✓	✓
zabbix-agent	✓	✓	✓	✓
collectd	✓	✓	✓	✓
python-diamond	✓	✓	✓	✓
sysstat	✓	✓	✓	✓
Logstash	~	~	~	~
Riemann	✗	✗	✗	✗

Tabelle 5.1: Ermittlung von lokalen Daten

In der nachfolgenden Tabelle wird verglichen, welche Datenerfassungssysteme auf einem Hypervisor Daten über virtuelle Maschinen erfassen können, ohne in eben diesen selbst zu laufen:

Programme	Zugewiesener RAM	CPU Kerne	Datenträger	Netzwerk
Coreutils	✓	✓	✓	✓
atop	✗	✗	✗	✗
zabbix-agent	✓	✓	✓	✓
collectd	✓	✓	✓	✓
python-diamond	✓	✓	✓	✓
sysstat	✗	✗	✗	✗
Logstash	✗	✗	✗	✗
Riemann	✗	✗	✗	✗

Tabelle 5.2: Ermittlung von VM Daten

Atop erfüllt die Anforderungen nicht und fällt somit aus der weiteren Betrachtung raus. Coreutils ermöglicht zwar das Ermitteln aller benötigten Daten, dies ist allerdings mit einem sehr großen Aufwand verbunden, da Daten von Hand aufbereitet werden müssen. Außerdem muss Netzwerkfunktionalität zum Versenden der Daten nachgerüstet werden. Somit fällt Coreutils auch raus. Sysstat ist nicht in der Lage, Daten von virtuellen Maschinen zu ermitteln und fällt somit auch raus. Logstash ist äußerst flexibel, benötigt aber etwas Arbeit bei der Entwicklung eines eigenen Eingabe-Plugins.

Übrig bleiben collectd und python-diamond. Beide erfüllen alle Anforderungen. Collectd ist in C geschrieben, dies bietet gegenüber Python im Diamond Projekt einen Geschwindigkeitsvorteil. Das Team rund um collectd arbeitet schon länger am Projekt und arbeitet kontinuierlich daran. Im weiteren Verlauf des Projekts wird deshalb mit collectd gearbeitet. Sollten sich hier unerwartete Probleme ergeben, kann auf Diamond gewechselt werden. [TM]

5.2 Datenhaltungssystem

Zu Beginn des Projektes musste das Projektteam mögliche Datenhaltungssysteme in einer Liste für eine darauffolgende Evaluierung definieren. Bei der Erstellung dieser Liste wurden Systeme aufgenommen, welche den Projektmitgliedern aus vergangenen Projekten, beziehungsweise aus dem täglichen Arbeitstag bereits bekannt

waren. Zusätzlich hatten die Projektmitglieder vorab mit Experten aus dem eigenen Unternehmen und aus dem Unternehmen des Auftraggebers über mögliche weitere Systeme gesprochen. Dadurch das Herr Luis in dem Bereich der Massendatenhaltungssysteme arbeitet, sind ebenfalls Systeme aufgenommen worden, welche im Bereich der sogenannten Big-Data-Technologie arbeiten.

Das Hinzufügen von weiteren/neuen Datenbanksystemen zur späteren Evaluierung erfolgt nur nach einer Genehmigung vom Auftraggeber.

Die in der Liste aufgenommen Systeme sind folgende:

- elasticsearch
- cassandra
- Postgres
- OpenTSDB
- KNIME
- impala
- hadoop
- hive

[NL]

5.2.1 Vorbereitung der Evaluierung

Als Vorbereitung für die Evaluation der Datenhaltungssysteme wurden mehrere Kriterien definiert, welche einen Leitpfaden für die spätere Arbeit geben sollen. Während der Auswahl und Definition, der von dem Datenbanksystem zu erfüllenden Kriterien, wurde darauf geachtet, dass diese sich aus dem späteren Zielsystem ableiten lassen und ebenfalls realistisch erfüllbar sind. Anschließend wurde die Relevanz jedes Kriteriums von den Teammitgliedern beurteilt und festgelegt.

Die daraus resultierten Kriterien sind nun nach Relevanz absteigend aufgelistet:

- Bereits vorhandenes Wissen über die Datenbank. Dazu gehört, dass diese eine umfangreiche Dokumentation besitzt und eine aktive und große Community für Diskussionen und Fragen vorhanden ist. Ebenfalls fallen die vorhandenen Vorkenntnisse über das Datenbanksystem der Projektmitglieder, sowie die Mitarbeiter von dem Unternehmen des Auftraggebers unter dieses Kriterium. Es ist wichtig, dass die aufzuwendende Einarbeitungszeit während des

Projekts und im späteren Wirkbetrieb bei Mitarbeitern des Unternehmens so gering wie möglich gehalten wird.

- Das Datenhaltungssystem muss einfach, schnell und jederzeit erweiterbar sein. Dies bedeutet, dass Ressourcen (CPU/RAM/Speicherplatz) des Datenbanksystems im optimalen Fall während des normalen Betriebs (kein Neustart oder Auszeit) aufgestockt werden kann. Es ist jedoch nach Anforderung des Auftraggebers ausreichend, wenn eine Aufstockung nach einer Auszeit von maximal 45 Minuten erfolgt ist.
- Umweltschonende und effiziente Datenhaltung ist zu beachten. Dies definiert eine optimale Relation zwischen der verwendeten Hardware und dem daraus resultierenden Ergebnis. Eine umweltschonende Datenhaltung kann unabhängig der korrekten Auswahl der genutzten Stromeffizienzklasse jedes Hardwarebauteils bereits bei der im Produktivbetrieb zu benutzende Software beeinflusst werden. Dies ist im Bereich der Datenhaltungs-Software die entstehende Speicherplatzgröße für einen einzelnen Datensatz in einer Datenbank. Um so kleiner dieser ist, desto mehr kann bei Energiekosten für zusätzlichen Speicherplatz eingespart werden.
- Die Datenhaltungs-Software sollte bereits Methoden zur Datensicherung besitzen. Dabei wird unterschieden in:
  - Von extern getriggerte Methoden, um Backups zu erstellen.
  - Intern genutzte Methoden, um im Fehlerfall (zum Beispiel ein Defekt am Netzwerkkabel oder Ausfall eines Servers im Cluster) einem Datenverlust vorzubeugen.

Unter Berücksichtigung der genannten Kriterien soll jedoch die Datenhaltungs-Software mit der gleichen Hardware-Ressource das beste und schnellste Ergebnis erbringen können. Dies bedeutet, dass auch bei komplexen Analysetechniken und zusätzlich hohen Datenmengen das System weiterhin gegenüber anderen Datenhaltungssystemen ein valides und schnelles Analyseergebnis erbringen kann. [NL]

5.2.2 Durchführung der Evaluierung

Bei der Durchführung der Evaluation von den ausgewählten Datenhaltungssystemen aus der Liste im Abschnitt 5.2 wurden die Evaluationskriterien aus Abschnitt 5.2.1 verwendet. Diese dienen als Leitpfaden für jedes Datenhaltungssystem und wurden von jedem Projektmitglied beachtet.

Zu Beginn der Durchführung teilte Herr Meusel als Projektleiter die zu evaluie-

renden Datenhaltungssysteme jedem Projektmitglied zu, um eine schnellere Bearbeitung zu gewährleisten. Bei der Einteilung wurden die bereits vorhandenen Erfahrungen jeder Projektmitglieder zu dem jeweiligen Datenhaltungssystem berücksichtigt. Im zweiten Schritt sollte sich anschließend jedes Projektmitglied mit dem zugeteilten Datenhaltungssystem kurz auseinandersetzen. In diesem Schritt ist Herrn Luis aufgefallen, dass es sich bei der in der Liste aufgenommenen Software „KNIME“ nicht um ein Datenhaltungssystem handelt, sondern um ein Werkzeug zur Datenanalyse, Datenaufbereitung und Datendarstellung.

Die daraus resultierenden Ergebnisse wurden Herrn Luis anschließend von den Projektmitgliedern bereitgestellt, sodass dieser einen Überblick im Projektbereich „Datenhaltung“ erschaffen konnte. Die Ergebnisse sind im folgenden Abschnitt für jede Datenhaltungs-Software dokumentiert. [NL]

Elasticsearch

Elasticsearch ist eine in Java geschriebene, verteilte Suchmaschine mit einer RESTful Schnittstelle. Die erste Veröffentlichung gab es am 8. Februar 2010 [BV10]. Elasticsearch speichert Daten im JSON Format ab. Die komplette Interaktion mit der Suchmaschine erfolgt über REST, hierzu zählt nicht nur das eigentliche Suchen, sondern auch die Administration der Software selbst und das Eintragen neuer Daten. Elasticsearch beschreibt sich selbst als eine dokumentenbasierte No-SQL Datenbank. Dies bedeutet, das es keine Tabellen mit Relationen gibt. Jeder Datensatz ist ein Dokument, welches als JSON gespeichert wird. Ein Dokument basiert aus Key->Value Paaren, diese können auch geschachtelt sein. Elasticsearch erzeugt mit Hilfe der Lucene Bibliothek Indexe auf diesen Daten, anschließend ist dann eine Volltextsuche möglich. Die Firma Elasticsearch BV aus den Niederlanden finanziert die Entwicklung der Software. Als Lizenz hat sich die Firma für die Apache License entschieden.

Unter Listing 13.13 befindet sich eine kurze Analyse des Git-Repositorys vom 27.11.2016. Die Suchmaschine besteht hier aus 25989 Beiträgen von 869 verschiedenen Personen.

Elasticsearch erlaubt es, als verteiltes Cluster betrieben zu werden. Hierfür werden mindestens zwei Server benötigt. Eingehende Daten werden in Indexe aufgeteilt, dies ist eine logische Gruppierung basierend auf der Charakteristik der Daten. Zum Beispiel kann man alle gemessenen Metriken charakterisieren anhand des Quellservers auf dem sie ermittelt wurden oder auch anhand des Typs (Festplattenauslastung, CPU-Auslastung). Jeder Index kann in einzelne Subgruppen aufgeteilt werden (Shards genannt). Jedes Shard kann auf einem anderen Server gespeichert werden.

Das Schreiben in die Shards, sowie Lesen aus den Shards, kann parallel auf allen Servern erfolgen. Somit erreicht man eine horizontale Skalierung. Es ist auch möglich, Replikas von Shards zu erzeugen. Hierbei wird eine Kopie eines Shards auf einem anderen Server im Cluster gespeichert. Bei jeder Änderung des echten Shards wird auch das Replika geändert. Sobald ein Server im Cluster ausfällt, kann aus dem Replika ein aktives Shard gemacht werden. Die Anzahl der Replikas pro Shard und deren Verteilung kann im laufenden Betrieb geändert werden. Somit erreicht man ein hochverfügbares Setup, welches im laufenden Betrieb um neue Server erweitert werden kann (vgl. [BV16a]).

Mit Jepsen wurde Elasticsearch mehrfach untersucht (vgl. [Kin15a]). Nach dem ersten Test im Jahr 2014 hat Elasticsearch BV eine Übersichtsseite erstellt mit allen offenen Problemen und Szenarien, die zu Datenverlust führen können (vgl. [BV16b]). Die wenigsten Hersteller und Communitys gehen so offen mit Problemen um.

Der letzte Jepsen-Test zeigt leider, dass Elasticsearch immer noch Probleme bei Netzwerkausfällen und starker Systemlast hat (vgl. [Kin15b]). Aufgrund der Größe solcher Setups ist es üblich, dass jeder Node im Cluster über zwei Netzwerkverbindungen verfügt. Eine davon ist zur internen Cluster-Kommunikation und zum Verteilen der Replikas, die andere für die Kommunikation mit der Außenwelt (Bereitstellung der RESTful-Schnittstelle).

Gegeben ist ein Teilausfall des internen Netzwerks (zum Beispiel ein Kabelbruch):

- Die Nodes merken, dass einer von ihnen nicht erreichbar ist.
- Replikas der fehlenden Shards werden aktiv gesetzt.
- Das Cluster nimmt weiter Daten an, sofern Replikas zur Verfügung stehen.
- Elasticsearch meldet, dass die neuen Daten erfolgreich geschrieben wurden.
- Die neuen Daten wurden nicht geschrieben und sind verloren.

Dieses Verhalten ist mit dem Jepsen-Test reproduzierbar (vgl. [Kin15c]). Während Elasticsearch Replikas von Shards eines ausgefallenen Nodes aktiv setzt, werden neue Daten weiterhin über die RESTful-Schnittstelle angenommen und dann ohne Rückmeldung verworfen.

Das nächste Problem ist das Verhalten von Elasticsearch unter hoher Systemlast. Ein Prozess unter Linux kann pausiert und wieder gestartet werden. Währenddessen kann ein Node deaktiviert werden. Hierfür gibt es verschiedene Gründe, zum Beispiel eine defekte Netzwerkverbindung oder eine absichtliche Abschaltung durch den Administrator, weil er Wartungsarbeiten durchführen möchte.

- Elasticsearch Prozess wird pausiert

- Das Cluster detektiert ein Netzwerkproblem zwischen dem Cluster und dem pausierten Node
- Der pausierte Node wird im Cluster deaktiviert
- Der Prozess wird wieder gestartet
- Der Prozess prüft nicht, ob er noch aktiv ist für seine Shards
- Der Prozess akzeptiert eingehende Schreibanfragen über die RESTful-Schnittstelle und updated seine Shards
- Der Prozess meldet zurück, dass er die Daten erfolgreich geschrieben hat
- Sobald das Netzwerkproblem behoben ist, erhält der Node Anweisungen aus dem Cluster, um seine Shards auf den Stand der Kopien im Netzwerk zu bringen und verwirft, was er selbst geschrieben hat

Das Pausieren passiert unter zwei Umständen. Der Garbage Collector macht dies in regelmäßigen Abständen, um den belegten Arbeitsspeicher nach nicht mehr benötigten Objekten zu scannen und den Speicher dann freizugeben. Der Garbage Collector pausiert nur für wenige Millisekunden, ein Datenverlust ist hier aufgrund des Zeitfensters unwahrscheinlich. Auch der Linux Kernel pausiert Prozesse, nämlich wenn ein System unter sehr hoher Last ist. Wenn die Hardware am Rand der Leistungsgrenze arbeitet und der Kernel einen Absturz erwartet, weil die Hardware bald überfordet ist, dann kann er einzelne Prozesse kurzzeitig pausieren.

Dieser Datenverlust ist ebenfalls mit einem Jepsen-Test reproduzierbar (vgl. [Kin15d]). In großen Setups ist es sehr realistisch, dass Server in einem Rechnerverbund (auch Cluster genannt) eine starke Last aufweisen. Sofern es hier zu Lastspitzen kommt, die eine Überlast andeuten, ist ein Datenverlust sehr wahrscheinlich.

Beide Probleme sind äußerst kritisch für dieses Projekt. In der Theorie ist die Architektur sehr gut geeignet, denn sie ist skalierbar und hochverfügbar. In der Praxis zeigt sich leider das Gegenteil. Gerade in großen Setups mit vielen Nodes ist nicht garantiert, dass Daten wirklich persistent geschrieben werden. [TM]

Cassandra

Cassandra ist ein SQL und Java basiertes DBMS und wurde ausgelegt für die Datenhaltung von großen Datenmengen und dessen verbundenem Management der Arbeitsauslastung auf dem Server. Dazu nutzt die Software ein Cluster-System über mehrere Server (Nodes), um ein vollständiges Versagen durch den Ausfall von Hardwarekomponenten oder Softwareteilen auf dem System zu verhindern. Dazu wird ein Peer-to-Peer Verfahren verwendet, welches die Dateninformationen auf meh-

rere Server verteilt und verwaltet. Im Cassandra-Umfeld wird dies über das von dem Hersteller definierte Protokoll „gossip“ verwaltet. Es handelt sich dabei um ein Kommunikationsprotokoll, welches den Austausch von Statusinformationen und Metainformationen zwischen den einzelnen Nodes ermöglicht. Wenn auf einem einzelnen Node Dateninformationen erstellt, modifiziert oder entfernt werden, erhält ein weiterer Node in der Infrastruktur diese Information und baut gleichzeitig eine Replikation des Objektes. Diese Replikationen werden von dem Serveradministrator definiert, indem dieser den Datenmengen verschiedene Partitionen zuteilt. Das Cassandra-System verteilt jede dieser Partitionen an die im Netzwerk vorhandenen Nodes in einer Ringtopologie. Dies bedeutet, dass Partition eins auf dem ersten Node repliziert wird, Partition zwei auf dem zweiten Node und immer so weiter, bis das Verfahren den ersten Node wieder erreicht.

In der Shell-Ausgabe unter Listing 13.18 wird ein während der Evaluierungsphase erstelltes Cassandra-System dargestellt. Dabei besitzt der Prototyp zwei aktive Nodes, welche für Testzwecke auf dem gleichen physischen Server betrieben werden. Im Produktivbetrieb sollten diese auf zwei verschiedenen Serversystemen arbeiten, um die Ausfallsicherheit der Hardware zu gewährleisten. Es ist jedoch zu erkennen, dass beide Systeme einen Dateneigentum, in der Shellausgabe die Spalte „Owns“, von 50 Prozentpunkten besitzen. Dies bedeutet, dass jeder Node jeweils die Hälfte der Dateninformationen bereitstellt, jedoch auch die zugehörige Hälfte von dem anderen Node als Backup abspeichert. Sollte nun einer der beiden Nodes ausfallen, könnte der jeweils andere automatisch einen Dateneigentum von 100 Prozentpunkten erstellen und ohne Verluste weiterarbeiten. Dies entspricht der Shared-Nothing-Architektur.

Je nach Node-Anzahl können die vorhandenen Replikations-Modelle in der Konfiguration eingestellt und eingesetzt werden. Cassandra bietet zudem drei Modelle zur Wiederherstellung von Informationen und Datensicherungen an, welche im Gesamten ein stabiles und sicheres System ermöglichen. Neben diesen Kerneigenschaften besitzt es jedoch auch noch weitere kleinere Methoden zur korrekten Datenverarbeitung. Darunter fällt das nicht sofortige Löschen von Daten in Datenbanktabellen. Stattdessen werden Daten nur als gelöscht markiert und bei zukünftigen Abfragen zunächst nicht beachtet. Die vollständige Löschung der Daten wird dann zum Beispiel zu Zeitpunkten erledigt, wenn das System sicher ist, dass keine Fehler oder Datenleichen entstehen können. [NL]

Postgres

Postgres, oder auch PostgreSQL genannt, ist ein SQL basiertes DBMS. Die initiale Arbeit an Postgres begann 1986 (vgl. [SR86]). Mit über 30 Jahren Entwicklung ist es eines der ältesten und am weitesten verbreitetsten DBMS (vgl. [sof16]). Es steht unter einer eigenen Open-Source-Lizenz und darf frei genutzt werden (vgl. [Gro16c]). Die Community ist sehr aktiv, sie pflegt ein komplexes Wiki (vgl. [Gro16d]), sowie eine eigene Seite mit Anleitungen für Einsteiger und Optimierungshilfen für Fortgeschrittene (vgl. [Pa16]).

Wie in der Shell-Ausgabe unter Listing 13.14 zu sehen, enthält das Git-Repository des Postgres Quellcodes am 26.11.2016 41366 einzelne Commits von 41 verschiedenen Autoren. Aufgrund der Patchpolitik von Postgres werden Patches oftmals an Autoren aus dem Postgres-Team geschickt. Ein Teammitglied fügt danach den Patch in das Repository ein. Somit lässt sich nicht genau bestimmen, von wie vielen Leuten Code beigesteuert wurde.

Die Postgres-Architektur sah lange Zeit vor, dass der Dienst auf einem einzigen Server betrieben wird. Mittlerweile gibt es mehrere Möglichkeiten, die Architektur anzupassen, um Postgres:

- Hochverfügbar zu betreiben
- Vertikal zu skalieren
- Mit einer Verteilung der Anfragen den einzelnen Server zu entlasten

Postgres bietet die Möglichkeit der Streaming Replication. Hierbei wird nach dem Active-Passive Prinip 1-N gearbeitet. Dies bietet zwei Vorteile:

- Leseanfragen können an passive Server gestellt werden.
- Im Fehlerfall oder für Wartungsarbeiten kann von einem aktiven auf einen beliebigen passiven Server umgeschaltet werden.
- Hochverfügbarer Betrieb der Datenbank.

Der aktive Server muss somit nur noch Schreibanfragen beantworten. Das Hinzufügen von passiven Nodes oder das Umschwenken der Schreibanfragen auf einen von ihnen kann im laufenden Betrieb erfolgen. Die Postgres-Konfiguration lässt sich zur Laufzeit anpassen, um dem Dienst mehr Ressourcen des physischen Hosts zuzuweisen.

Mit den Erweiterungen Pgpool-II und Postgres-XC kann ebenfalls ein hochverfügbarer Betrieb realisiert werden. Außerdem ermöglichen sie eine vertikale Skalierung. Mehrere Server können hier zu einem Verbund (auch Cluster genannt) zusammenge-

schaltet werden. Schreibanfragen können von mehreren Servern beantwortet werden. Wenn es gewünscht ist, kann nach der Shared-Nothing-Architektur gearbeitet werden. In dem Fall wird nur ein funktionierender Server im Cluster benötigt. In einem Setup mit mehreren Nodes führt ein Ausfall einer oder mehrer Nodes nur zu einer verringerten Leistung, nicht aber zu Datenverlust oder Erreichbarkeitsproblemen des Verbunds. [TM]

OpenTSDB

OpenTSDB ist ein Zeitreihen-basiertes Programm, auf Englisch „Time Series Daemon (TSD)“, und arbeitet mit dem Open-Source-Programm DBMS „HBase“ zusammen. Es verfügt über die Funktionen von einem Konsolen-basierten Zugang, sowie verschiedenen Treibern für den Zugriff und Austausch von Informationen zwischen anderen Netzwerkgeräten. Es übernimmt nicht die Aufgabe der Datenspeicherung. Dazu wird das zugehörige DBMS „HBase“ zwingend benötigt. OpenTSDB wird genutzt, um einzelne und unabhängige Systeme für den Datenzugriff bereitzustellen. Dabei können beliebig viele Systeme (Deamons) an einem HBase-System angebunden sein. Es existiert keinerlei Informationsaustausch zwischen den einzelnen Deamons, sondern sie nutzen lediglich alle die gleiche Datengrundlage des HBase-Systems. Dennoch kann man durch dieses Verfahren die Analyseperformance dynamisch anpassen, da das HBase-System stark auf die Aggregation und niedrigen Speicherplatzbedarf optimiert wurde. Die einzelnen Deamons kommunizieren dabei mit dem HBase-System über einfache Protokolle wie Telnet oder HTTP (API).

Da die Dateninhalte immer in Verbindung mit einem Zeit- und Datumswert stehen, ist dieses DBMS durch die Optimierung des Verarbeitens der Zeit- und Datumsbasierten Datensätze ein Kandidat für die spätere Nutzung in der Ziellösung. [NL]

Hadoop / Hive / Impala

Das Apache-Hadoop-System wird zum aktuellen Zeitpunkt bei großen und leistungsfähigen Datenhaltungssystemen eingesetzt, sodass es auch oftmals in Verbindung mit den Worten „BIG DATA“ genannt wird.

Es handelt sich dabei um eine Vielzahl von Komponenten, bestehend aus mehreren Frameworks und Ökosystemen, also im Klartext eine Reihe von frei wählbaren Bibliotheken zur Definition der Datenhaltungslandschaft. Apache Hadoop verfolgt dabei den Ansatz, alle Dateninformationen in nur einem gebündelten System zu verwalten und keine zusätzlichen Datenhaltungssysteme, welche gegebenenfalls nur auf eine einzelne Anforderung spezialisiert sind, zu gründen.

Um diesen Ansatz zu erfüllen, wurde das Cluster-Konzept eingesetzt, Hadoop nennt es selber „Hadoop Distributed File System“ (HDFS). Dabei werden die Daten in der Datenhaltung nicht mehr auf einem einzelnen physischen System gehalten, sowie vor einem Hardwareausfall gesichert, sondern nun auf mehrere Systeme verteilt. Hadoop teilt dabei jedem in dem Cluster hinzugefügten System eine spezielle Aufgabe zu. Für die Datenspeicherung existieren dabei drei Aufgabenfelder. Nummer Eins ist der Data Node, dieser besteht oftmals aus mehreren Festplatten und einer Netzwerkanbindung. Auf diesem wird ein Datensatz mehrmals auf verschiedenen Festplatten abgespeichert, sodass bei einem Ausfall der Datensatz gesichert ist. Es können bei Hadoop beliebig viele Data Nodes in der Datenhaltungslandschaft eingesetzt werden, je mehr Data Nodes, umso mehr Speicherplatz und Rechenleistung steht zur Verfügung. Nummer zwei ist der Name Node. Bei diesem handelt es sich um ein einzelnes System, welches den Speicherort (Data Node ID) von einem Datensatz bereitstellt. Die letzte Aufgabe erfüllt der Backup Node. Auch bei diesem handelt es sich um ein einzelnes System. Das System kommt zum Einsatz, wenn der Name Node ausfallen sollte. Er synchronisiert sich dauerhaft mit dem Name Node, um im Falle eines Ausfalls des Name Nodes, sofort die Aufgaben zu übernehmen.

Für das Verwalten des Cluster-Konzepts und zeitbasierenden Aufgaben (im englischen: Job scheduling) wurde „Hadoop YARN“ gegründet. Dieses System definiert eigenständig den Austausch von Informationen zwischen den einzelnen Nodes. Es definiert dabei auch, zu welchem Tageszeitpunkt einzelne Aufgaben erfüllt werden sollen. Diese Definitionen werden automatisch erstellt und durchgeführt, jedoch kann dies auch von einer Person manuell eingestellt und verwaltet werden.

Für die Anbindung späterer Systeme und Methodiken der Datenanalyse, wird „Hadoop Common“ eingesetzt. Es handelt sich dabei um ein reines Schnittstellen-Modul, welches standardisierte Protokolle implementiert und betreibt. Es ermöglicht zum Beispiel die Anbindung der Software „Hive“ und „Impala“, welche in den nachfolgenden Abschnitten „Beschreibung Hive“ und „Beschreibung Impala“ erläutert werden.

Für die Bearbeitung und Analyse von zeitkritischen Daten kommt die Komponente „Hadoop Map Reduce“ zum Einsatz. Diese ist von dem gleichnamigen Programmiermodell abgeleitet. Es handelt sich dabei um ein Hadoop-YARN-basiertes System, welches ein paralleles Abarbeiten von Aufgaben ermöglicht. Dabei werden Analyseanfragen nun nicht mehr in mehreren Server-Threads auf einem einzelnen System aufgeteilt, sondern auf mehreren Data Nodes, welche wiederum die Aufgaben in mehreren Threads abarbeiten können. Gleichzeitig besitzt diese Hadoop-Komponente einen Data Node, welcher aus reinem Arbeitsspeicher besteht. Dieser wird genutzt, um Informationen für eine kurzfristige Zeit auf ein schnelleres Medium

zu laden, sodass die Information im Falle einer erneuten Abfrage innerhalb dieser Zeit erneut genutzt werden kann, ohne diese wieder von einem langsameren Medium laden zu müssen. Die Zeitspanne kann dabei von einem Administrator festgelegt werden. Arbeitsspeicher ist bis dato einer der schnellsten Speichermedien und kommt bei zeitkritischen Datenabfragen zum Einsatz.

Diese vier Komponenten bilden den „Hadoop Core“, welcher zwingend für den Betrieb eines Hadoop-Systems benötigt wird. Hadoop basierende Systeme zur Erfüllung von verschiedenen und spezialisierten Aufgaben werden unter dem „Hadoop Ecosystem“ zusammengefasst. Eine vollständige Liste von Hadoop basierenden Systemen und Projekten kann auf der offiziellen Webseite des Entwicklers entnommen werden (vgl. [For16]). [NL]

Beschreibung Hive

Das Apache-Hive-System ist eine Hadoop basierende Software und nutzt die im Abschnitt „Hadoop / Hive / Impala“ erläuterten Schnittstellten für den Datenzugriff. Es führt dabei eine spezialisierte Aufgabe auf dem Hadoop-System aus und fällt somit nicht unter ein eigenständiges Datenhaltungssystem. In der Hadoop-Landschaft wird es unter dem Begriff „Hadoop Ecosystem“ eingeordnet.

Die angesprochene spezialisierte Aufgabe ist dabei die Implementierung von einem sogenannten „Data Warehouse“ (DWH) in dem Hadoop Netzwerk. Ein Data Warehouse wird in Unternehmen verwendet, um eine vollständige Übersicht zu einem spezifischen Inhalt zu erhalten. Dies könnte als Beispiel die im Land verteilten Einkaufsläden des Unternehmens sein, welche alle ein eigenes Kassensystem mit einer Datenbank besitzen. Mit Hilfe des Extraktions-, Transformations- und Ladeprozesses (ETL) werden die unterschiedlich abgespeicherten Informationen von jedem einzelnen Kassensystem ausgelesen und anschließend dem Aufbau der Tabellen (Schema/Datenhaltungsstruktur) von dem Data Warehouse angepasst und somit die vollständige Speicherung im System gewährleistet (siehe auch Abschnitt 7.8.1).

Dadurch das in dem Hadoop-System keinerlei Struktur der Daten existiert, übernimmt Apache Hive mit Hilfe der Methodiken von dem Data Warehouse das Lesen, Schreiben und Managen von Dateninformationen. Es beginnt dabei mit Werkzeugen für den Zugriff auf die Daten innerhalb des Hadoop-Systems. Dies ist zum einem die Bereitstellung von einem standardisierten Treiber für den externen Zugriff auf die Daten, sowie die Vorgabe einer Abfragesprache mit dem bekannten Namen „SQL“. Jedoch implementiert Apache Hive auf dem Hadoop-System auch Funktionen wie das Reporten von Inhalten über eine wählbare Schnittstelle wie zum Beispiel dem Versand einer eMail oder Anzeige auf einer Internetwebseite.

Zusammengefasst hat Apache Hive die Aufgabe zur Bereitstellung einer strukturierten Sicht (Schema) auf die Daten innerhalb des Hadoop-Systems. Dabei kann Apache Hive mehrere Sichten auf einen einzelnen Datensatz definieren und somit auch das Rechtesystem innerhalb der Datenhaltung unterstützen, indem diese nur für ausgewählte Benutzeraccounts zur Benutzung freigeschaltet werden. Die in dem Data Warehouse bekannten „Data Marts“ fallen somit weg und werden durch den Aufbau der genannten Sichten ersetzt. Ein Data Mart ist eine von dem Datawarehouse getrennte Datenbank, welche genutzt wird, um für ausgewählte getrennte Daten Analysen und Reports zu erstellen. Diese können zum Beispiel nach Abteilungen im Unternehmen getrennt werden, sodass das Marketing nur Daten zu Produkten und Beständen erhält, jedoch keine technisch orientierten Daten. Des weiteren werden Data Marts in der dritten Normalform und ein Datawarehouse in der zweiten Normalform betrieben, entsprechend dienen sie dazu, die Daten nochmals fachlich aufzubereiten. [NL]

Beschreibung Impala

Das Apache-Impala-System ist eine Hadoop basierende Software und nutzt die im Abschnitt „Hadoop / Hive / Impala“ erläuterteten Schnittstellten für den Datenzugriff. Es führt dabei eine spezialisierte Aufgabe auf dem Hadoop-System aus und fällt somit nicht unter ein eigenständiges Datenhaltungssystem. In der Hadoop-Landschaft wird es unter dem Begriff „Hadoop Ecosystem“ eingeordnet.

Die angesprochene spezialisierte Aufgabe ist dabei eine ähnliche Implementierung, wie im Abschnitt „Beschreibung Hive“. Apache Impala stellt ebenfalls eine Reihe von Werkzeugen bereit, darunter eine spezialisierte Abfragesprache auf Basis von „SQL“. Apache Impala arbeitet auf den konsolidierten Metadaten des Hadoop-Systems, benötigt jedoch auch die Verknüpfung (Interface) eines Strukturmodells. In den meisten Fällen kommt dabei Apache Hive zum Einsatz. Apache Impala kann die Inhalte von Apache Hive Eins-zu-eins übernehmen und ausführen. Es kann jedoch auch ohne einem Apache Hive Systems ausgeführt werden.

Apache Impala wird eingesetzt, um die Schnelligkeit der Datenabfragen auf einem Hadoop-System zu verbessern. Es wird dabei auf jedem Data Node installiert. Dabei erhalten die Data Nodes neben der eigentlichen Datenspeicherung noch zusätzlich die Funktionen zur Ausübung von Datenbankabfragen (SQL). Konkret wird auf jedem Data Node ein „Query Planer“, „Query Coordinator“ und „Query Executor“ bereitgestellt. Der Query Planer hält dabei die Kommunikation mit der SQL-Applikation innerhalb des Netzwerkes. Dieser gibt definierte Datenabfragen von Benutzern an einen beliebigen Data Node weiter. Nachdem der Query Planer die Da-

tenabfrage erhalten hat, kommuniziert er dies an den Query Coordinator, welcher wiederrum alle umliegenden Data Nodes um Unterstützung bei der Datenabfrage bittet. Dies geschieht über den Query Executor, welcher, wie schon der Name sagt, die Abfrage auf dem Hadoop-HDFS-System ausführt und die Daten zurück an den Query Coordinator liefert. Die von dem Query Coordinator zusammengesetzten Daten liefert dieser anschließend an die SQL-Applikation zurück und der gesamte Prozess ist abgeschlossen.

Durch das Nutzen von dem Arbeitsspeicher auf den Data Nodes, sowie der gleichzeitigen Nutzung des Hadoop-Map-Reduce-Verfahrens, werden die Datenabfragen um ein Vielfaches beschleunigt. Da die Leistung des Clusters durch mehrere Faktoren beeinflusst wird, kann keine exakte Angabe der Performanceverbesserung geliefert werden. Diese Faktoren sind zum Beispiel die Schreibweise der SQL-Syntax, da ein nicht optimierter Code die Performanceverbesserung wieder mindern kann. Fakt ist jedoch, dass mit einem korrekt aufgesetzten Cluster System der Einsatz von Apache Impala die Performance der Datenlandschaft optimiert. [NL]

Evaluierungsbericht

Während der Evaluierungsphase wurde ein vollständiger Hadoop Core als Prototyp auf zwei Testservern installiert und in den Grundlagen konfiguriert. Während dieser Arbeit ist dem Projektteam bereits klar geworden, dass nur für die Konfiguration des ersten Systemstarts, also dem Verbindungsaufbau, sowie den allgemeinen Funktionen eines Hadoop Core Systems, zu viel Zeit benötigt wird. Des Weiteren wurde nach dem Testen des Systems die Konfigurationsmöglichkeiten im Bereich Optimierung der Performance und Geschwindigkeit studiert. Dabei wurde klar, dass durch die Größe des Systems mehr als die gesamte Projektzeit benötigt wird, um eine optimale und vollständige Konfiguration und Systemstruktur zu erarbeiten. Aufgrund dessen wurde die Verwendung eines Hadoop-Systems in diesem Projekt sehr schnell ausgeschlossen. Hauptgründe dabei waren zum einem die vielen Werkzeuge und Methoden die das Hadoop-Ecosystem bereitstellen konnte und zum anderen die Masse an Dokumentationen und dem damit verbundenen Konfigurationsaufwand. Hinzu kommt noch die darauf folgende fachliche Definition von Datenstrukturen. Diese hätte durch die breite Auswahl an Methodiken auf einem Hadoop-System ebenfalls wieder viel Zeit beansprucht. Zunächst hätte eines der verfügbaren Softwareelemente des Hadoop-Ecosystems evaluiert und ausgewählt werden gemusst. Zusammengefasst wäre das System den technischen Anforderungen an dem Projekt gewachsen, jedoch wäre der Einsatz mit einem zu großen Aufwand verbunden, wel-

chen das Projektteam in dem vorgegebenen Projektzeitraum nicht erbringen könnte. [NL]

5.2.3 Abschluss der Evaluierung

Nachdem alle Systeme von dem Projektteam auf die genannten Kriterien untersucht wurden, wurde gemeinsam abschließend eine normale Entscheidungstabelle angefertigt. Dabei kann jedes im Abschnitt 5.2.1 definierte Kriterium einen Wert zwischen Null und Zehn erhalten. Zehn ist dabei die beste und Null die schlechteste Bewertung. Die Systeme mit der höchsten Gesamtpunktzahl würden anschließend in der engeren Auswahl beurteilt werden. Der Punkt Wissen bezieht sich auf vorhandenes Wissen innerhalb des Projektteams.

Diese Entscheidungstabelle wurde wie folgt aufgebaut:

System	Wissen	Skalierbarkeit	Umwelt	Datensicherung	Gesamt
elasticsearch	8	7	6	3	24
cassandra	7	5	6	8	26
Postgres	10	10	7	8	35
OpenTSDB	6	5	7	3	21
KNIME	0	0	0	0	0
Hadoop Core	8	10	4	8	30

Tabelle 5.3: Gesamtbewertung der Datenbanksysteme

Wie in der Entscheidungstabelle abzulesen ist, wurde im Anschluss über die Systeme Postgres und dem Hadoop Core diskutiert. Es wurde bereits während der Evaluierungsphase erkannt, dass bei der Verwendung eines Hadoop-Core-Setups die eingeplante Realisierungs-Zeit zu gering ist. Aufgrund dessen und einer höheren Gesamtpunktzahl wird das System Postgres für dieses Projekt verwendet. [NL]

5.3 Schnittstelle - Datenbereitstellung

Neben der herkömmlichen Methode zur Datenabfrage von Datenbank-Systemen über einen Datenbanktreiber, existiert noch die Möglichkeit einer API.

Die Verwendung von einer API-Schnittstelle besitzt Nachteile sowie Vorteile, beziehungsweise verbirgt gewisse Gefahren in Hinsicht auf den späteren Einsatz im Produktivsystem. Eine dieser Gefahren ist das Vernachlässigen der Dokumentation einer API. Dadurch dass während der Entwicklung einer API bereits definiert wird, welche Fähigkeiten und Methodiken die API besitzen soll, ist eine nicht vorhandene oder unvollständige Dokumentation suboptimal. Darum muss bei der Auswahl des genutzten API-Systems darauf geachtet werden, dass die Dokumentation von Softwarekomponenten für den Entwickler so einfach wie möglich erfolgt.

Wenn die Dokumentation einer API vollständig und das ausführende System den Performance-Anforderungen gerecht ist, bringt die Nutzung eines API-Systems Vorteile. Darunter ist eine bessere Verwaltung von Zugriffsrechten und somit auch gleichzeitig eine Verbesserung der Systemsicherheit. Durch den vorab festgelegten Funktionsumfang der API kann der Systeminhaber gezielt kontrollieren, welche Inhalte ein Benutzer sehen oder modifizieren kann. Dabei können grobe Einschränkungen, wie das Lesen oder Schreiben auf der gesamten Datenbank, oder auch detaillierte Einschränkungen, wie das Lesen oder Schreiben von einer einzelnen Tabelle in der Datenbank, kontrolliert werden. Anders als bei der Verbindung über einen Datenbanktreiber kann bei der API der potentielle Angreifer keine eigenen Datenbankbefehle definieren und ausführen. Somit wird das Risiko der Datenmanipulation durch eine zusätzliche Schicht (Layer) verringert.

Oftmals benötigen die Systeme, welche auf die Daten des Datenhaltungssystems zugreifen, die Daten in einer anderen Struktur als diese abgespeichert sind. Durch die Programmiermöglichkeiten einer API können während der Abfrage der Informationen auf dem Datenhaltungssystem bereits Datenanpassungen vorgenommen werden. Diese Anpassungen können zum Beispiel Funktionen, wie das anonymisieren oder pseudonymisieren sein. Die API kann jedoch auch zum Beispiel die Daten vor der Auslieferung beliebig anpassen und somit auf ein weiteres System optimal zuschneiden, ohne dabei die Ursprungsdaten zu verändern. [NL]

5.3.1 Vorbereitung der Evaluierung

Anders als bei der Evalierung der Datenbanksysteme wurde bei der Evaluierung der möglichen API-Systeme keine Liste im Voraus erstellt. Dies lag daran, dass keines der

Projektmitglieder oder keiner der Interessenten an dem Projekt Erfahrungen mit API-Systemen im Vorhinein besaß und dies erst während der Projektphase angeeignet werden musste.

Aufgrund dessen wurde entschieden, eine Liste von Anforderungen an das API-System zu erstellen und im Anschluss während der Evaluierungsphase Softwarelösungen zu suchen, welche die Anforderungen am besten erfüllen können.

Die festgehaltenen Anforderungen sind nun nach Relevanz absteigend aufgelistet:

- Die API Software muss auf einer selbst administrierten Hardware-Umgebung betrieben werden können. Das Verwenden eines fertigen API-Softwarepakets eines Cloudanbieters, bei welchem der Administrator alle Konfigurationen über ein Web-Interface des Providers vornimmt und der Cloudanbieter die vollständige Hardware-Umgebung administriert, ist ein absolutes Ausschlusskriterium. Es ist vorgesehen, dass das API-System möglichst nah an dem Datenhaltungssystem liegt, um mögliche Latenzen zwischen den beiden Systemen so niedrig wie möglich zu halten. Zudem sollen externe Cloudanbieter keinen Zugriff auf die teilweise kundenbezogenen Daten der Datenhaltung erhalten, auch wenn das System nur Performance-Daten ohne Kundenbezug ausliefern soll. Außerdem kann während des Projekts entschieden werden, dass das Datenhaltungssystem nicht von externen Systemen erreichbar sein soll, sodass die damit verbundene Lösung über einen Cloudprovider nicht möglich wäre. Externe Systeme sind in diesem Projekt Geräte, welche außerhalb des Unternehmens betrieben werden.
- Die API-Software muss vollständig konfigurierbar und administrierbar sein. Entsprechend soll die Software nur eine leere Hülle mit Standardfunktionen einer API zur Verfügung stellen. Eine Konfiguration von einzelnen Parametern ist dabei erwünscht. Zu diesen Parametern gehört als Beispiel der Port, auf welchem das API-System betrieben und von anderen Systemen erreicht werden kann. Zur Hilfe der Admnistration soll das API-System ein Framework für Entwickler bereitstellen. Mit diesem können wir als Projektmitglieder anschließend die Logik in das System implementieren und somit definieren, welche Funktionen die API ausüben darf. Ein Framework in einer bekannten Entwicklersprache ist dabei erwünscht. Dies hat den Vorteil, dass der zuständige Entwickler im Projekt keine weitere Einarbeitungszeit für die neue Entwicklersprache benötigt.
- Das API-System sollte innerhalb des Projektes keine Kosten verursachen. Dazu gehören Lizenzkosten sowie laufende Betriebskosten bei externen Cloudanbietern. Spätere Kosten bei dem Auftraggeber sind in dieser Anforderung nicht

mit inbegriffen. Grundsätzlich gilt für das ganze Projekt, dass nur Software mit einer Open-Source-Lizenz genutzt werden darf.

- Die Erstellung von späteren Anwendungshandbüchern der API soll dem API Entwickler so einfach wie möglich fallen. Eine sehr beliebte Dokumentationsmöglichkeit ist dabei das Dokumentieren von wichtigen Informationen direkt in dem Quellcode des API-Systems. Entwickler können dabei über die Kommentierungsfunktion der jeweiligen Entwicklersprache eine Beschreibung sowie Ein- und Ausgabe von Werten festlegen. Diese werden im Anschluss während des Kompiliervorgangs in einem strukturierten Dokument festgehalten und neben der eigentlichen API-Software mit ausgeliefert. Dieses separate Dokument ist oftmals in Form einer HTML oder PDF und kann auf gängigen Computersystemen eingesehen werden. Die beschriebene oder ähnliche Dokumentationsmöglichkeit sollte die API-Software zur Verfügung stellen. Alternativ kann auch erst die Dokumentation geschrieben werden und darauf basierend Quellcode generiert werden. Beide Verfahren nennt man „One Source of Truth Prinzip“. Hierbei wird sichergestellt, dass es nur eine autoritative Quelle gibt und der zugehörige Pendant auf der Quelle beruht. Dies vermeidet Dokumentation die nicht mit den eigentlichen Funktionen der Software übereinstimmen.
- Bereits vorhandenes Wissen über das API-System. Dazu gehört, dass diese eine umfangreiche Dokumentation besitzt und eine aktive und große Community für Diskussionen und Fragen vorhanden ist. Die dazugehörige Programmiersprache der API sollte ebenfalls für die Projektmitglieder nachvollziehbar sein sowie eine umfangreiche Dokumentation und aktive Community für Diskussionen und Fragen besitzen.
- Das API-System sollte nicht von einem einzelnem Betriebssystem abhängig sein. Das bedeutet, dass die API auf jedem beliebigen Betriebssystem vollständig und zuverlässig ausgeführt und arbeiten kann.

[NL]

5.3.2 Durchführung der Evaluierung

Bei der Evaluierung der API-Systeme wurden die im Abschnitt 5.3.1 festgehaltenen Anforderungen der API als Leitlinie für die Durchführung verwendet. Als ersten Schritt wurde zunächst Recherche betrieben, damit das Projektteam einen allgemeinen Eindruck über die aktuelle Marktsituation zu API-Systemen erhält. Dabei wurden im Internet mehrere Blogs, Foren sowie Artikel von Fachzeitschriften gele-

sen. Die dort angesprochenen API-Systeme wurden anschließend mit den bereits definierten Anforderungen abgeglichen und aussortiert. Dabei stellten die Projektmitglieder fest, dass die meisten API-Systeme von Cloudprovidern nur in bereits vorkonfigurierten Paketen mit einem monatlichen Festpreis angeboten werden. Beides ist ein absolutes Ausschlusskriterium für den späteren Produktivbetrieb sowie Nutzung der Software. Ebenfalls wurde bei den vorkonfigurierten Paketen eine eigene Definition von Funktionen komplett verboten. Lediglich ein einzelner Cloudprovider hatte ein Angebotsmodel, bei dem die Kunden das betriebene API-System in Hinsischt der Funktionen modifizieren durften.

Aufgrund dieser Erkentnisse hat sich das Projektteam während der Evaluierung für ein API-System mit dem Namen „Swagger“ entschieden. Es ist aktuell neben den API-Systemen „RAML“ und „API-Blueprint“ das einzige API-System auf dem Markt, welches den Betrieb auf einer selbst administrierten Hardware-Umgebung zulässt. Für den Vergleich der drei Systeme wurde eine Entscheidungstabelle mit gewichteten Faktoren erstellt und ausgewertet. Die in der nachfolgenden Entscheidungstabelle genannten Faktoren wurden aus dem Abschnitt 5.3.1 abgeleitet und können dort im Detail nachgeschaut werden. Jeder der definierten Kriterien kann einen Wert zwischen 0 und 10 erhalten. Zehn ist dabei die beste und Null die schlechteste Bewertung. Die Faktoren bezüglich der selbst administrierbaren Hardwareumgebung, und die damit verbundenen externen Kosten sind bei allen drei genannten API-Systemen gegeben und werden nicht berücksichtigt. API-Systeme bei denen dies nicht gegeben ist, wurden von dem Projektteam aus der Bewertung ausgeschlossen.

System	Modifizierbar	Doku	Wissen	Abhängigkeit	Gesamt
Swagger	10	9	10	10	39
RAML	10	8	7	10	35
API-Blueprint	8	6	5	9	28

Tabelle 5.4: Gesamtbewertung der API-Systeme

[NL]

5.3.3 Zusammenspiel API und Frontend

Nachdem die Evaluierung der Datenvisualisierung im Abschnitt 5.4.3 abgeschlossen war, wurde dessen Zusammenspiel mit der API überprüft. Dabei schaute sich Herr Luis speziell die Kommunikation zwischen den beiden Softwareprodukten an. Mit der Kommunikation ist der Aufbau einer Datenanfrage und die daraus resultierende Datenzulieferung gemeint. Der von der Datenvisualisierungs-Software definierte Kommunikations-Aufbau muss die API eins zu eins übernehmen, andernfalls wäre eine vollständige Kommunikation, und somit die Übertragung der benötigten Informationen, der beiden Softwareprodukte nicht möglich.

Neben der technischen Kommunikation wurde ebenfalls die Informationsinhalte, welche die API von der Datenvisualisierung erhält und zurück sendet, auf den fachlichen Inhalt bewertet. Dabei konnte das Projektteam erkennen, dass bei der Verwendung eines API Systems, der Datenvisualisierungs-Administrator keine optionalen Datenfilterungen an die API übergeben kann. Speziell in diesem Projekt entstand dabei die Problematik, dass bei der Datenanfrage an der API nicht ersichtlich war, um welchen Kunden, beziehungsweise eingeloggten Benutzer es sich handelt. Entsprechend würde die API nicht wissen ob Daten an einen Administrator, welcher alle Server sehen darf, oder ob Daten an einen Kunden, welcher nur seine eigenen Server sehen darf, zurück gesendet werden sollen.

Um diese Problematik zu lösen, hat das Projektteam den „Header“-Bereich (auf Deutsch Kopf-Bereich), welcher bei jeder HTTP- oder HTTPS-Anfrage vorhanden ist, genauer untersucht. In diesem steht die Herkunft der Anfrage und weitere Informationen wie der genutze Browser des Internetbenutzers. Speziell bei der Anfrage von der Datenvisualisierung an die API steht in dem Header-Bereich immer die Organisations-ID von dem anfragenden Benutzer. Die Organisations-ID ist bei der für das Projekt ausgewählten Datenvisualisierungs-Software eine Abgrenzung der Zugriffsrechte. Das heißt sie definiert, welche Daten, beziehungsweise Dashboards der eingeloggte Benutzer sehen darf und welche nicht.

Für die Ziellösung wurde entsprechend von dem Projektteam festgelegt, dass die Datenhaltung neben den eigentlichen Daten auch eine Mapping Tabelle besitzen soll. Diese Mapping Tabelle besitzt entsprechend die Information, welche Organisations-ID zu welcher virtuellen Instanz, beziehungsweise Host-Server gehört. Sollte die Datenvisualisierung Daten bei der API anfordern, würde diese vor der eigentlichen Datenabfrage bei der Datenhaltung die vorhandene Organisations-ID in den Server-Namen übersetzen. Mit dieser Übersetzung kann die API anschließend regulär die Daten von der Datenhaltung anfordern und an die Datenvisualisierung liefern.

Diese Lösung hat zusätzlich noch den Vorteil einer weiteren Datensicherheit. Die

genannte Organisations-ID kann ausschließlich im Backend der Datenvisualisierung verändert werden. Einem Benutzer ist es nicht möglich den Wert dieser ID in irgendeiner Weise zu verändern und somit andere Daten zu erhalten. [NL]

5.4 Datenvisualisierung

Bevor das Projekt begann, musste eine Softwareliste mit möglichen Datenvisualisierungslösungen erarbeitet werden. Bereits vorab wurden folgende Softwarelösungen in das Pflichtenheft aufgenommen:

- Grafana
- Graphite
- Zabbix-Frontend

Während den ersten Recherchen für die oben aufgelisteten Datenvisualisierungslösungen, wurden nach Absprache mit dem Auftraggeber noch folgende Systeme in die Evaluierungsliste aufgenommen:

- Datadog
- Gridster-D3
- Netdata

Das Hinzufügen von weiteren Softwarelösungen während der Evaluierungsphase muss durch den Auftraggeber zugestimmt werden. Erst nachdem dieser zustimmt, kann eine weitere hinzugefügt werden. Zusätzlich zu der Liste mit den einzelnen Softwarelösungen, wurden verschiedene Kriterien, welche die Datenvisualisierung erfüllen muss, in einem Kritierienkatalog definiert. Diese werden für die eigentliche Durchführung der Evaluierung und zusätzlich für die Anfertigung einer Nutzwertanalyse für den Abschluss der Evaluierung benötigt. Die zu evaluierende Softwareliste besteht zum Teil aus Softwarelösungen, welche im Unternehmen von einzelnen Projektmitgliedern bereits erfolgreich eingesetzt werden und zum anderen aus Recherchen, welche während der Evaluierung aufgefallen sind. [MR]

5.4.1 Definition eines Graphen

Ein Graph besteht im Wesentlichen aus mehreren Knoten und Kanten, welche auf einem Koordinatensystem angelegt sind. Mit Hilfe von zwei Knoten, bestehend jeweils aus einer Koordinate (x/y), werden diese eindeutig zu einer Kante definiert.

Ein linearer Graph wird genau dann visualisiert, wenn zwei Knoten miteinander verbunden werden. Zusätzlich gibt es die Unterscheidung zwischen einem gerichteten und ungerichteten Graphen. Ein gerichteter Graph besteht aus einem Anfangsknoten A und einem Endknoten B, welche mit einer Kante verbunden sind. Bei einem ungerichteten Graphen ist sowohl der Knoten A mit dem Knoten B, als auch der Knoten B mit Knoten A als Kante miteinander verbunden (vgl. [KK08]).

Innerhalb der Datenvisualisierungslösung werden dem Benutzer verschiedene Graphen von beispielsweise CPU-Auslastung, RAM-Auslastung oder SSD-Schreibzugriffe visualisiert. Diese Graphen können entweder in Linerar- oder Balkenform dargestellt werden. Lineare Graphen haben eine X- und Y-Achse und besitzen zusätzlich einen Anfang und einen wachsenden Endpunkt. Er wird mittels Metrik (siehe Abschnitt 5.1.1), welche die Webseite aus den Datenhaltungssystemen abgreift, erstellt. Diese kann vom Administrator ebenfalls nur für eine definierte Zeitspanne angezeigt werden. Balkendiagramme haben nur eine X-Achse, bei der der gewünschte Analysewert eingetragen ist und werden zum Beispiel für ausgefallene SSDs in einem bestimmten Zeitraum verwendet. Diese werden über die Metrik aus dem Datenhaltungssystem abgegriffen und dem Administrator über eine Übersicht angezeigt. Ein Graph- beziehungsweise Balkendiagramm visualisiert immer eine oder mehrere Timeseries (siehe Abschnitt 5.1.1). [MR]

5.4.2 Vorbereitung der Evaluierung

Damit die Evaluierungen mittels einer Nutzwertanalyse der einzelnen Softwarelösungen durchgeführt werden können, müssen zuvor im ersten Schritt Kriterien mit dem Projektteam in einen Kriterienkatalog aufgenommen werden. Anschließend werden diesen Kriterien entsprechende Gewichtungen zugewiesen. Bei den Kriterien muss während der Beurteilung beachtet werden, dass diese innerhalb der im Projekt zur Verfügung stehenden Zeit realisierbar sind. Dabei wird das Prüfen, Testen, sowie Realisieren in den zeitlichen Abschätzungen der einzelnen Kriterien beachtet. Nachdem die einzelnen Kriterien zusammen mit dem Projektteam erstellt worden sind, wurden diese im Anschluss Herrn Reuter zur abschließenden Überprüfung zur Verfügung gestellt. Erst hiernach wurden diese in einem Kriterienkatalog aufgenommen.

Die einzelnen Kriterien wurden im späteren Verlauf der Evaluierung zusammengefasst und gewichtet. Im nachfolgenden können die einzelnen Kriterien entnommen werden. Der Oberbegriff definiert jeweils immer die Kategorie zu der das Kriterium zugewiesen ist. Folgende Kategorien wurden der Nutzwertanalyse hinzugefüt: Kosten, Sicherheit, Skalierbar, Performance und Dokumentation.

Kosten

- In der Grundanforderung unter Abschnitt 2.4 ist definiert, dass die zu evaluieriende Softwarelösung eine Open-Source-Lizenz besitzen muss. Damit der Quellcode angepasst werden und keine (Folge-)Kosten innerhalb des Projekts und somit beim Kunden anfallen können.

Sicherheit

- Um den Zugriff und damit auch die Sicherheit der Webseite zu schützen, muss eine Authentisierung, Authentifizierung und Autorisierung seitens von der Datenvisualisierung unterstützt werden. Diese dienen in erster Linie für den Schutz der Daten. Speziell die Anmeldung an der Webseite schützt den unautorisierten Zugriff durch Dritte. Der Zugriff auf die Webseite soll mit einer Transportverschlüsselung durch HTTPS durchgeführt werden und je nach Kundenanforderung auch eigene Client-Zertifikate zulassen. Die Anmeldung an der Datenvisualisierung kann entweder über einen lokalen Administrator, als auch über eine LDAP-Anbindung verfügen, um zum Beispiel Unternehmensverzeichnisse der Kunden anbinden zu können. Über diese soll ebenfalls eine Rechtevergabe innerhalb der Datenvisualisierung möglich sein.
- Um die Sicherheit der Datenvisualisierung zusätzlich schützen zu können, muss diese mit einem API-System interagieren können, damit eigenmächtige Modifizierungen der Daten wie das verändern, anpassen oder sogar löschen unterbunden sind.

Skalierung

- Die Datenvisualisierung muss eine Plugin-Erweiterung unterstützen. Es dient dazu, dass im späteren Verlauf, wenn das Produkt bereits beim Kunden eingesetzt wird, auf bestimmte Kundenwünsche direkt eingehen zu können. Es sollte bereits schon eine Auswahl von Pluginerweiterungen in der Community für die Datenvisualisierung geben. Ebenso sollte es möglich sein, eigene Plugins entwickeln und implementieren zu können.
- Eine leichte Anpassung, Konfiguration und direkte Ausgabe der Visualisierung von CPU, RAM, Netzwerk oder ähnlichen Datenwerten. Hiermit ist gemeint, dass gewährleistet werden muss, dass die einzelnen Graphen Endanwenderspezifisch schnell und einfach angepasst werden können. Ebenso soll die Anbindung an eventuell schon vorhandenen Datenbanksystemen beim Kunden realisiert werden können, welche keinen direkten Bezug zu diesem Projekt haben.

Performance

- Da auf das Datenhaltungssystem nicht nur die Datenvisualisierung zugreift, sondern auch die Datenerfassung, um Daten abspeichern zu können, kann es zu massiven Performance-Problemen kommen, die durch die Schnittstelle der Datenvisualisierung verursacht werden können. Ein wichtiges Kernkriterium ist, dass der Administrator eine Zeitspanne vordefinieren kann, in welcher die Graphen neu visualisiert und Daten aus dem Datenhaltungssystem abgerufen werden sollen. Dies dient dazu, dass das Datenhaltsungssystem nicht mit Anfragen überlastet wird. Ein kurzes Rechenbeispiel soll dies verdeutlichen. Acht unterschiedliche Benutzer greifen gleichzeitig auf die Datenvisualisierung zu und visualisieren im Fallbeispiel acht Graphen mit unterschiedlichen Werten. Dies sind insgesamt 64 Graphen, welche sekündlich eine Abfrage an das Datenhaltungssystem senden werden.

Dokumentation

- In der Grundanforderung unter Abschnitt 2.4 ist definiert, dass eine aktive und qualitativ hochwertige Community vorrausgesetzt wird. Der Auftraggeber und auch die Projektmitglieder selbst, benötigen keine große Schulung für den Umgang der Datenvisualisierung. Dies erleichtert zum einen den späteren Betrieb, als auch die Wartung im späteren Verlauf, wenn das Produkt bei einem Kunden eingesetzt wird.

[MR]

5.4.3 Durchführung der Evaluierung

Für die Durchführung der Evaluierung der Datenvisualisierungslösungen aus der in Abschnitt 5.4 definierten Liste, wurden die entsprechenden Kriterien zur Evaluierung aus Abschnitt 5.4.2 verwendet.

Bevor die Evaluierung der einzelnen Softwarelösungen durchgeführt werden konnte, wurden vorab bereits eine Vielzahl an Informationen über die einzelnen Softwarelösungen mittels Internetrecherche und Fachzeitschriften eingeholt. Zusätzlich wurden die einzelnen Softwarelösungen auf zwei Projektmitglieder aufgeteilt, um die Evaluierung und damit auch die Projektplanung zeitlich zu verkürzen.

Nachdem die Analyse der Softwarelösungen von den Projektmitgliedern abgeschlossen wurde, wurden die gesammelten Resultate Herrn Reuter zur Verfügung gestellt. Mit diesen Resultaten konnte Herr Reuter bereits eine Vorauswertung aller Softwarelösungen durchführen. Um die Nutzwertanalyse durchführen zu können, wurde jede Softwarelösung gegenübergestellt und auf den zuvor erstellten Kriterienkata-

log geprüft. Hierfür wurde ein Punktesystem von Null bis Fünf festgelegt, wobei Null die schlechteste Punktzahl und Fünf die beste Punktzahl ist. Diese vergebenen Punkte wurden anschließend mit der Gewichtung in eine Tabelle eingetragen und multipliziert, welches dann die Gesamtpunktzahl ergibt. Die Tabellen werden im Abschnitt 5.4.4 genauer erklärt.

Während der Evaluierung der einzelnen Softwarelösungen ist aufgefallen, dass die Softwarelösung Gridster-D3 keine direkte Visualisierungssoftware ist. Die entsprechende Evaluierung, von diesem Produkt, kann im Abschnitt 5.4.3 vollständig entnommen werden.

Im Nachfolgenden sind die einzelnen Datenvisualisierungslösungen mit einer kurzen Zusammenfassung, sowie Vor- und Nachteile aufgeführt: [MR]

Grafana

Grafana ist eine von Torkel Ödegaard & Raintank Inc entwickelte Open-Source-Visualisierungssoftware. Sie bietet neben einfachen Visualisierungsgraphen noch weitere Ereignisfunktionen. Um die Anmeldung an der Webseite durchführen zu können, kann hier wahlweise die Anmeldung mit einem Active Directory Konto oder lokalen Benutzer plus Kennwortkombination durchgeführt werden. Hinzu können verschiedene Berechtigungen vergeben werden. Somit können zum Beispiel Mitarbeiter aus der Führungsebene eingeschränkte Zugriffe auf das System erhalten, um hier zum Beispiel Analysen durchführen zu können. Grafana arbeitet mit fast jedem Timeseries Datenhaltungssystem, wie z.B. InfluxDB zusammen. Zusätzlich kann es über Plugins erweitert werden. Damit Grafana API Systeme (siehe auch Abschnitt 5.3) ansprechen kann, wird hier ein solches Plugin benötigt. Plugins werden zur Erweiterung der Software in der Programmiersprache Java entwickelt und installiert. Diese werden zum Beispiel dann benötigt, wenn weitere Graphen, Analysen oder unterstützte Datenquellen erweitert werden sollen. Dies bedeutet, dass das Datenhaltungssystem API Statements zur Verfügung stellen kann und hiermit Graphen visualisieren kann. Grafana kann über einen Datenbanktreiber oder eine Schnittstelle in Form eines Treibers einfache Statements auf das Datenhaltungssystem durchführen, um die Daten abzugreifen. Es ist zusätzlich eine Open-Source-Software, wodurch der Quellcode angepasst und verändert werden darf. Dies bedarf zum Beispiel bei Anpassungen an bestimmte Kundenumgebungen. Dadurch entstehen keine Support-Kosten, da die Software nicht als ein Dienst angeboten wird. Grafana hat eine stetig wachsende Community mit sämtlichen Dokumenten zur Einrichtung oder Erweiterung des Systems. Der gesamte Quellcode ist auf GitHub veröffentlicht, somit hat jeder die Möglichkeit, das Produkt zu verbessern und neue

Funktionen zu implementieren. Die vom Anwender erstellten Dashboards mit allen Graphen und Statements können vom System, wie auch vom Anwender selbst, auf einen externen Dateifreigabeserver oder lokal abgespeichert werden (vgl. [Öde16]).

Zusammenfassung:

Positiv:

- Authentisierung, Authentifizierung und Autorisierung
- Gute Community und eine Vielzahl an Dokumentationen
- Leichte Anpassungen an Graphen und Einstellungen möglich
- Dashboards können auf externen Server oder lokal gesichert werden
- API-Unterstüzung nach Plugin installation
- Plugin-Erweiterung möglich (Java)
- Open-Source-Softwarelösung

Negativ:

- Plugin-Erweiterung nur in Programmiersprache Java möglich, weshalb Abhängikeit zu speziellen Entwicklern besteht
- Webdesign kann nicht angepasst werden
- Nur zwei Design Webtemplates verfügbar (dark/white)

[MR]

Graphite

Die Softwarelösung Graphite wurde von Chris Davis im Jahr 2006 entwickelt und gehört zu einer der am meisten eingesetzten Softwarelösungen im Bereich Monitoring und Visualisierung. Graphite ist wie auch Grafana eine Open-Source-Lösung und ermöglicht es dem Anwender, die Software bearbeiten und anpassen zu können. Es verfügt über eine inhaltlich gute und umfangreiche Community mit einem eigenem Wiki-Board, wo alle relevanten Dokumentationen enthalten sind. Graphen können leicht mittels Statements über eine API-Datenabfrage oder direkt an das Datenhaltungssystem durchgeführt werden. Eine Anmeldung an der WebOberfläche mittels Benutzername und Kennwort oder LDAP Anbindung ist ebenfalls möglich. Es können einzelne Graphen, wie auch das gesamte Dashboard, auf externe Server oder lokal abgespeichert werden, um einem Verlust von einzelnen Graphen vorbeugen zu können. Graphite ist, wie auch Grafana, kompatibel zu Timeseries Datenhaltungssystemen. Es bietet keine einfache Möglichkeit, weitere Plugins hinzufügen

zu können, wie es beispielsweise Grafana kann. Hierzu muss der Quellcode von Graphite massiv angepasst werden, wodurch gute Kenntniss der einzelnen Dienste erforderlich sind. Graphite ist ein System, welches aus mehreren Diensten besteht. Der Graphite-Webdienst ist für die Anzeige der Webseite zuständig. Der Carbon-Cache-Dienst wird verwendet, um die Daten, welche vom Datenerfassungsystem ermittelt worden sind, aufzuarbeiten. Dies wird benötigt, damit Graphite diese für die Visualisierung verwenden kann. Grundsätzlich installiert sich mit der Installation von Graphite noch eine Whisper-Datenbank, welche jedoch durch eine beliebige, wie OpenTSDB oder InfluxDB, ausgetauscht werden kann (vgl. [Dav16]).

Zusammenfassung:

Positiv:

- Authentisierung, Authentifizierung und Autorisierung
- Gute Community und eine vielzahl an Dokumenten innerhalb eines Wiki
- API-Unterstützung
- Dashboards und einzelne Graphen können auf externen Servern oder lokal abgespeichert werden
- Open-Source-Softwarelösung
- Dienste können hochverfügbar und auf unterschiedliche Server aufgeteilt werden

Negativ:

- Keine Möglichkeit, leicht Plugins hinzuzufügen
- Webdesign kann nicht angepasst werden
- Besteht aus mehreren Diensten, wie graphite-web und carbon-cache

[MR]

Zabbix-Frontend

Zabbix-Frontend ist eine von Zabbix LCC entwickelte Softwarelösung, welche dem Benutzer eine leichte und einfache Konfiguration von einzelnen Graphen ermöglicht. Es bietet zudem eine inhaltlich gute Wissensdatenbank, in der viele Dokumente zur Konfiguration von einzelnen Funktionen bis hin zur Erläuterungen von jedem Button enthalten sind. Das Zabbix-Frontend benötigt keinerlei Plugins, da hier bereits viele Funktionen in der eigentlichen Software enthalten sind. Plugins können dennoch in der Programmiersprache Java entwickelt und implementiert werden. Es

ist darauf ausgelegt bei verschiedenen Events, beispielsweise das Erreichen eines kritischen Festplattenstands, eine E-Mail an den Benutzer zu versenden, weshalb Zabbix sehr häufig gerade in großen Infrastrukturen zum Einsatz kommt. Die Anmeldung an der Webseite kann wahlweise mit lokalen Benutzer mit Benutzername und Kennwort oder mit einer LDAP-Anbindung durchgeführt werden. So benötigt der Administrator beziehungsweise der Benutzer lediglich seine Active-Directory-Benutzerdaten zur Authentifizierung. Zabbix ist zudem mit API 5.3 Abfragen kompatibel. So können hier Graphen mit API-Abfragen visualisiert werden. Sollte keine Abfrage über eine API gewünscht sein, so kann dies ebenfalls mittels Statement direkt auf dem Datenhaltungssystem durchgeführt werden. Um die Datensicherheit und Ausfallsicherheit gewährleisten zu können, bietet Zabbix die Möglichkeit, einzelne Graphen mit Statements, wie auch das gesamte Dashboard exportieren und importieren zu können (vgl. [SIA16]).

Zusammenfassung:

Positiv:

- Authentisierung, Authentifizierung und Autorisierung
- Gute inhaltliche Community und eine Vielzahl an Dokumenten innerhalb eines Wikis
- Leichte Anpassungen an Graphen und Einstellungen möglich
- API-Unterstützung
- Dashboards und einzelne Graphen können auf externe Servern oder lokal abgespeichert werden
- Bringt bereits eine Softwarelösung mit, womit Daten von virtuellen Maschinen exportiert werden können
- Benötigt keine Plugins
- E-Mail-Benachrichtigung bei bestimmten vordefinierten oder selbstdefinierten Events
- Open-Source-Softwarelösung

Negativ:

- Keine Möglichkeit, das Webdesign anzupassen

[MR]

Datadog

Datadog ist eine von Oliver Pomel entwickelte Analysesoftware und wird bereits in großen Firmen, wie zum Beispiel EA oder Citrix erfolgreich eingesetzt. Es ist nicht wie die anderen Software-Lösungen ein Open-Source-Produkt und wird in verschiedenen Lizenzen von der Anzahl der entsprechenden virtuellen Instanzen, welche abgegriffen werden sollen, gestaffelt. Die Basis-Lizenz gibt es hier bereits kostenlos, jedoch können hier nur maximal 5 virtuelle Instanzen abgefragt werden und die Daten werden maximal einen Tag zurückgehalten. Die teuerste Lizenz kostet 23 Dollar und beinhaltet alle Hosts, welche der Kunde anbinden möchte. Zusätzlich bietet diese Lizenz einen E-Mail-, Chat- oder Telefonsupport. Dieser ist nicht in der Basis-Lizenz vorhanden. Graphen und auch Dashboards können in Datadog einfach vom Administrator erstellt werden. Um die Anmeldung an der Webseite durchführen zu können, wird ein Benutzer und Kennwort benötigt. Ebenfalls besteht die Möglichkeit ein LDAP an Datadog anbinden zu können. Innerhalb eines Graphen können bestimmte Teilabschnitte mit zum Beispiel Soziale Netzwerke, wie Facebook geteilt werden. Datadog bietet zudem eventbasierte Mitteilungen, welche vom Administrator festgelegt werden können. Datadog hat eine vollständige API Unterstützung und Dashboards können vom Administrator exportiert werden. Eine Möglichkeit, Graphen zu exportieren, bietet Datadog hier nicht. Auf die Community, sowie für den IRC Channel haben nur Kunden Zuriff, welche eine Pro-Lizenz gekauft haben (vgl. [16b]).

Zusammenfassung:

Positiv:

- Authentisierung, Authentifizierung, Autorisierung
- Leichte Anpassung an Graphen, Dashboard und Einstellungen möglich
- API-Unterstützung
- Gute inhaltliche Community mit IRC-Channel und Telefonsupport (steht nur in der Kauflizenz zur Verfügung)
- Benötigt keine zusätzlichen Plugins
- Teilabschnitte von Graphen können in Social-Media-Plattformen geteilt werden

Negativ:

- Keine Open-Source-Lizenz. Lizenzen sind hier nach Anwendungsfall gestaffelt

[MR]

Gridster-D3

Gridster ist eine von Anmol Koul entwickelte Java-Bibliothek, welche auf GitHub einsehbar und dokumentiert ist. Während der ersten Recherchen stellete sich heraus, dass es sich hierbei nicht direkt um eine Visualisierungssoftware handelt, sondern lediglich um eine Funktion, nämlich das Drag & Drop von einzelnen Graphen oder Elementen innerhalb einer Webseite. Mit dieser besteht die Möglichkeit, ein Dashboard, welches selbst entwickelt worden ist, individuell anpassen zu können (vgl. [Kou16]).

Somit fällt Gridster-D3, da es sich hierbei nicht um eine Visualisierungssoftware handelt, aus der weiteren Evaluierung heraus. [MR]

Netdata

Netdata ist eine Open-Source-Monitoring-Softwarelösung. Es wurde von Costa Tsaousis entwickelt und bietet eine umfangreiche Möglichkeit, Daten in Form von Graphen oder anderen Visualisierungen darzustellen. Der vollständige Code steht auf GitHub, worüber auch die Community gepflegt wird. Andere Mitglieder der GitHub-Community haben hier die Möglichkeit, Quelltext anzupassen und zu verändern, jedoch müssen diese von anderen Mitgliedern bestätigt werden. Es bietet nicht die Möglichkeit, eine Authentifizierung auf der Webseite durchzuführen. Ebenso werden alle Graphen, welche abgefragt werden, auf einer Seite angezeigt, weshalb die Übersicht stark eingeschränkt ist. Graphen können bei Netdata nicht einfach angepasst werden, da diese im Quellcode der Webseite verändert und angepasst werden müssen. Ein klarer Nachteil ist allerdings, dass diese Softwarelösung jeweils nur eine Instanz gleichzeitig überwachen kann und der Vorteil bei Netdata ist, dass kein weiteres Datenhaltungssystem oder ähnliches benötigt wird (vgl. [Tsa16]).

Zusammenfassung:

Positiv:

- Es werden sämtliche Daten unmittelbar ohne Datenbank bereits visualisiert
- Gute inhaltliche Community
- Vollständige Open-Source-Softwarelösung

Negativ:

- Sehr unübersichtlich vom Aufbau der Webseite
- Authentisierung, Authentifizierung und Autorisierung kann nicht gewährleistet werden

- Bietet keine API-Unterstützung
- Kann keine zusätzliche Datenbank ansprechen, somit ist eine Datensicherung nicht gewährleistet

[MR]

5.4.4 Abschluss der Evaluierung

Nach Abschluss der Evaluierung der einzelnen Softwarelösungen für die Datenvisualisierung wurden diesen entsprechende Bewertungspunkte für die Kriterien von Null bis Fünf vergeben. Damit eine Softwarelösung in die nähere Auswahl aufgenommen werden kann, muss diese mindestens 3,00 Gesamtpunkte in der Nutzwertanalyse erhalten. In nachfolgender Tabelle kann die Nutzwertanalyse der Softwarelösungen Datadog, Gridster-D3 und Netdata entnommen werden.

Kriterien	Gewicht	Datadog		Gridster-D3		Netdata	
		Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt
Skalierbar	25%	3	0,75	0	0,00	2	0,50
Kosten	20%	1	0,20	0	0,00	5	1,00
Sicherheit	30%	3	0,90	0	0,00	1	0,30
Performance	15%	4	0,60	0	0,00	1	0,15
Dokumentation	10%	5	0,50	0	0,00	4	0,40
Gesamt	100%		2,95		0,00		2,35

Tabelle 5.5: Nutzwertanalyse Datadog, Gridster-D3 und Netdata

In der oben stehenden Tabelle erhält die Softwarelösung Datadog nur 2,95 Gesamtpunkte, da diese Softwarelösung eine Kostenpflichtige Lizenz benötigt. Dies ist innerhalb des Projektes ein Kernkriterium, weshalb es als Softwarelösung nicht weiter verwendet werden kann. Gridster-D3 ist bereits während der Evaluierungsphase ausgeschieden, da es sich hierbei nicht direkt um eine Softwarelösung handelt sondern lediglich um eine Funktion. Aus diesem Grund hat Gridster-D3 0,00 Gesamtpunkte in der Nutzwertanalyse erhalten. Im Laufe der Evaluierung hat Netdata 2,35 Gesamtpunkte erhalten, da es sehr unübersichtlich aufgebaut ist und keine Sicherheit gewährleisten kann. Alle drei Softwärelösungen sind damit ausgeschieden, da eine Gesamtpunktzahl von 3,00 nicht erreicht worden ist.

Nachfolgend bleiben drei Datenvisualisierungslösungen für die nähere Auswahl:

- Grafana
- Graphite
- Zabbix-Frontend

Ebenfalls wie auch bei den Softwarelösungen Datadog, Gridster-D3 und Netdata, wurde ebenfalls eine Nutzwertanalyse durchgeführt, die wie folgt aufgebaut ist:

Kriterien	Gewicht	Grafana		Graphite		Zabbix-Frontend	
		Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt
Skalierbar	25%	4	1,00	3	0,75	2	0,50
Kosten	20%	5	1,00	5	1,00	5	1,00
Sicherheit	30%	4	1,20	4	1,20	4	1,20
Performance	15%	3	0,45	2	0,30	2	0,30
Dokumentation	10%	5	0,50	5	0,50	5	0,50
Gesamt	100%		4,15		3,75		3,50

Tabelle 5.6: Nutzwertanalyse Grafana, Graphite und Zabbix-Frontend

Nachdem die Nutzwertanalyse für Grafana, Graphite und Zabbix-Frontend durchgeführt worden ist, hat das Zabbix-Frontend eine Gesamtpunktzahl von 3,50 erhalten. Während der Evaluierung der Zabbix-Frontend fiel auf, dass es ohne einen Zabbix-Server nicht verwendet werden kann. Der Zabbix-Server ist ein von Zabbix LLC entwickelter Netzwerkdienst. Er empfängt Metriken von Agents, bereitet diese auf und speichert die Daten anschließend in einem Datenhaltungssystem ab. Somit besteht bei der Zabbix-Frontend Softwarelösung eine Abhängigkeit zu anderen Diensten, weshalb diese im Fehlerfall Probleme verursachen könnte. Zusätzlich ist aufgefallen, dass das Zabbix-Frontend maximal nur mit einer Datenbank gleichzeitig kommunizieren und verwendet werden kann. Anders als bei Grafana, welches mit mehreren Datenbanken gleichzeitig kommunizieren kann. Innerhalb der einzelnen Graphen werden hier mittels einer Abfrage die Datenbanken direkt angesprochen, nachdem diese in der Konfiguration hinzugefügt wurde. Bei der Evaluierung der Graphite Softwarelösung, hat Graphite 3,75 Gesamtpunkte in der Nutzwertanalyse erhalten. Während der Implementierung von Graphite wurde klar, dass es deutlich komplexer ist als Grafana. Graphite besteht aus mehreren Diensten, welche bei der

Installation konfiguriert und angepasst werden müssen, damit Graphite funktional ist. Wenn ein Dienst ausfallen sollte, betrifft dies das gesamte Graphite-System und die Datenvisualisierung wäre nicht mehr erreichbar. Zusätzlich ist gerade in der Wartung der Dienste ein Mehraufwand, da der Administrator alle Dienste in Form von Abhängigkeit zueinander und Konfiguration kennen muss. Da Graphite aus mehreren Diensten besteht, benötigt Graphite auch mehr Systemressourcen als ein einzelner Dienst, wie es bei Grafana der Fall ist. Hierdurch bedingt, wird Graphite für die weitere Implementierung nicht verwendet. Grafana hat in der Nutzwertanalyse die meisten Gesamtpunkte mit 4,15 erhalten. Es erfüllt alle von den Projektmitgliedern zuvor definierte Kriterien. [MR]

6 Userstories

Im Abschnitt 3.2 wurde bereits kurz auf Userstories eingegangen. Im Folgenden wird die generelle Vorgehensweise erklärt, sowie die Implementierung und Realisierung der Userstories in diesem Projekt.

Zu jeder Userstory wird ein Drahtgittermodell (Englisch: Wireframe) erstellt. Dies ist eine schematische Zeichnung der angefragten Änderung. Bei Änderungen an einer Webseite wird eine Nachbildung der Seite gezeichnet, bei Datenbankänderungen eine Tabelle. Im Anschluss darauf erfolgt die Zerlegung (Englisch: Decomposition) des Wireframes in die einzelnen Informationsbestandteile. Als Beispiel kann angenommen werden, dass eine Userstory einen zusätzlichen Graphen auf einer Webseite anfordert. Ausschlaggebend für die Zerlegung sind folgende Fragen:

- Werden zusätzliche Daten benötigt, um die Userstory zu implementieren oder reichen die vorhandenen Daten, die bereits auf der Webseite genutzt werden?
- Wo kommen diese Daten her?
- Wer liefert die Daten bzw. wird dafür eine Genehmigung benötigt?
- Werden bereits genutzte Daten nicht mehr benötigt?
- Ist die Änderung atomar oder kann sie weiter zerlegt werden?

Daraus ergeben sich mehrere Änderungen, die am Projekt gemacht werden müssen, um die Userstory zu implementieren. Für jede Änderung wird ein Ticket erstellt. Ziel des Ganzen ist es, eine komplexe Userstory, die einen einzelnen erfahrenen Entwickler lange mit der Implementierung beschäftigt, möglichst weit herunterzubrechen. Die entstandenen Teilaufgaben erfordern größtenteils keinen erfahrenen Entwickler. Dieser ist nur noch für einige, wenige Tickets notwendig. Ein weiterer Vorteil ist, dass nun mehrere Leute parallel an den einzelnen Tickets arbeiten können. [TM]

6.1 SSD Userstory

Name der Story: Der Administrator möchte eine Anzeige der SSDs, welche einen Hardwaredefekt haben könnten.

Beschreibung: In der heutigen Zeit werden vermehrt langsame Hard Disk Drives durch schnellere Solid State Disks ausgetauscht. Gerade im Serverbetrieb sorgt dies für einen schnelleren Datenzugriff. SSDs haben jedoch, wie auch HDDs, einen Lebenszyklus, der bei SSDs schneller erreicht werden kann, als bei handelsüblichen HDDs. Der Administrator soll darüber benachrichtigt werden, sobald von einer SSD ein kritischer Schreib- oder Lesezyklus erreicht worden ist, um die SSD bereits vor dem Ausfall proaktiv austauschen zu können. Ebenfalls soll eine Statistik ausgegeben werden können, welche SSD von welchem Hersteller in der Wahrscheinlichkeit am häufigsten ausfällt. Im Folgenden finden sich die Akzeptanzkriterien:

- Die Ausgabe und Visualisierung in einem Balken- oder Graphendiagramm für jede einzelne Instanz
- Auflistung einer Statistik in Form eines Balkendiagramms über die Hersteller, welche am wahrscheinlich häufigsten in den Instanzen ausfallen
- Eine Benachrichtigung, ob per E-Mail oder auf anderen Kommunikationswegen, wenn ein kritischer Schreib- oder Lesezyklus erreicht worden ist

Das dazugehörige Wireframe, kann dem Anhang 13.15 entnommen werden. [MR]

6.2 CPU Userstory

Name der Story: Der Administrator möchte eine Auswertung der CPU-Auslastung jeder Instanz, um Engpässe zu erkennen.

Beschreibung: Die CPUs der heutigen Zeit werden in der Taktrate und Verarbeitungsgeschwindigkeit der Prozesse immer schneller und besser. Bei virtuellen Maschinen werden aus einem physischen CPU-Kern des Host-Systems mehrere virtuelle CPU-Kerne emuliert und einer Instanz zugewiesen. So kann eine VM Zwei virtuelle CPU-Kerne mit zum Beispiel 2,5GHz Taktrate zugewiesen haben. Auf jeder Instanz laufen andere Dienste, weshalb jede VM unterschiedliche CPUs zugeteilt haben kann. Hier kann es vorkommen, dass eine VM mehr CPU-Leistung benötigt, als ihr selbst zugewiesen ist. Um dies frühestmöglich erkennen zu können, benötigt der Administrator eine detaillierte Übersicht in Form eines Graphen, wie viel CPU-Leistung von einer virtuellen Instanz verwendet wird und auf dem System verfügbar ist. Im Folgenden finden sich die Akzeptanzkriterien:

- Visualisierung der Auslastung und Verfügbarkeit der CPU-Leistung für jede einzelne Instanz
- Alle Werte der CPU, wie beispielsweise Soft-IRQ,iowait oder Interrupt müssen visualisiert werden
- Eine frühestmögliche Erkennung von einem Engpass einer virtuellen Maschine

Das dazugehörige Wireframe, kann dem Anhang 13.16 entnommen werden. [MR]

6.3 Memory Userstory

Name der Story: Der Administrator möchte eine Auswertung der Arbeitsspeicherauslastung von jeder Instanz, um Engpässe zu erkennen.

Beschreibung: Vergleichbar wie auch bei der CPU-Auslastung, gibt es auch eine ähnliche Thematik zur Auslastung bei dem verwendeten Arbeitsspeicher. Meist ist hier jedoch nicht die Taktrate des Arbeitsspeichers entscheidend, sondern eher wie viel Arbeitsspeicher einer virtuellen Maschine zugewiesen ist. Hier kann es ebenfalls wie auch bei der CPU Userstory 6.2 vorkommen, dass es zu Engpässen auf der virtuellen Maschine bei der Verteilung des Arbeitsspeichers kommen kann. Im Folgenden finden sich die Akzeptanzkriterien:

- Ausgabe und Visualisierung in Form eines Graphen für den verwendeten, freien und zwischengespeicherten Arbeitsspeicher für jede einzelne virtuelle Maschine
- Eine frühestmögliche Erkennung von einem Engpass des Arbeitsspeichers einer virtuellen Maschine

Das dazugehörige Wireframe, kann dem Anhang 13.17 entnommen werden. [MR]

6.4 Zeitdefinierte Analyse Userstory

Name der Story: Der Administrator möchte eine Auswertung über die im vergangenen Jahr verwendeten Ressourcen einer virtuellen Maschine.

Beschreibung: Ein Onlineshop für zum Beispiel saisonale Ware, wie Adventskalender, benötigt eine Analyse über die verwendeten Ressourcen aus dem vergangenen Jahr, um hier im kommenden Jahr die Ressourcen bereits bei Beginn richtig skalieren zu können. Dies dient in erster Linie, um die Erreichbarkeit des Onlineshop

gewährleisten zu können, wie auch die Performance im Allgemeinen zu verbessern. So können zum Beispiel bestimmten virtuellen Maschinen zu einem gewissen Zeitraum mehr Ressourcen zur Verfügung gestellt werden. In den Monaten in dem keine große Nachfrage der Webseite besteht, können hier weniger Ressourcen der virtuellen Maschine zugewiesen werden. Dies hat den Vorteil, dass spezielle Vertäge dem Kunden angeboten werden können. Im Folgenden finden sich die Akzeptanzkriterien:

- Es soll eine Ausgabe in visualisierter Form über einen bestimmten Monat dargestellt werden
- Ressourcen, welche benötigt werden, können direkt bereits ab Beginn zugewiesen werden

Das dazugehörige Wireframe, kann dem Anhang 13.18 entnommen werden. [MR]

6.5 Webinterface Userstory

Name der Story: Der Administrator möchte eine Visualisierung der erhaltenen Daten zur weiteren Verwendung.

Beschreibung: Dem Administrator oder der Benutzer soll es ermöglicht werden, eine visualisierte Ausgabe in Form von Graphen zu erhalten. Diese sollen Aufschluss über die Auslastung der einzelnen Systeme geben können. Im Folgenden finden sich die Akzeptanzkriterien:

- Visualisierung erfolgt für jeden einzelnen abgegriffenen Wert
- Zeitspanne der Visualisierung kann über den Benutzer angepasst werden
- Die Graphen, der Visualisierung kann über den Benutzer angepasst werden

Das dazugehörige Wireframe, kann dem Anhang 13.19 entnommen werden. [MR]

7 Realisierung

In der Evaluierungsphase wurde für jeden Bereich die optimale Software gesucht und gefunden. Hier wurde mit Absicht jeder Bereich getrennt betrachtet. Es wurde nicht geprüft, ob die gewählte Software in einem Bereich mit den Alternativen in den anderen beiden Bereichen kompatibel ist. Aufgrund der hohen Auswahl an Möglichkeiten pro Bereich gibt es zu viele Kombinationsmöglichkeiten. Diese können zeitlich nicht alle im Rahmen des Projektes evaluiert werden. Es wurde darauf geachtet, dass jede Software mit offenen und bekannten Schnittstellen arbeiten kann. Diese werden in der Realisierungsphase miteinander kombiniert. Alle arbeiten wurden auf dem Betriebssystem Archlinux durchgeführt, da hier die meiste Erfahrung im Team vorhanden ist. Um eine große Kompatibilität sicherzustellen wurden allerdings alle Tests nicht nur für Archlinux, sondern auch für die Distributionen Debian, Ubuntu und CentOS durchgeführt. [TM, NL, MR]

7.1 Prototyp 1

Entschieden wurde sich für Postgres als Datenbank, collectd zur Datenerfassung und Grafana zur Visualisierung. Hinzu kommt eine selbst entwickelte API für komplexe Abfragen und Fragestellungen. Die erste Idee der Umsetzung ist unter 13.4 visualisiert. Die Architektur ist in drei Teile gegliedert. Im ersten Teil befinden sich alle Hypervisors und virtuellen Maschinen, welche Daten via collectd verschicken. Jede Instanz von collectd soll ihre Daten in der zentralen Postgres Datenbank speichern. Für Tests wird mit einer einzelnen Postgres Datenbank gearbeitet, diese kann bei Bedarf um weitere Server erweitert werden, um einen hochverfügbaren Betrieb zu realisieren. Komplexe Datenbankabfragen über die API können, gerade bei großen Datenmengen, eine unerwartet hohe Last erzeugen. Es ist möglich, dass in dieser Zeit eingehende Daten von collectd nicht direkt gespeichert werden können. Die Daten würden dann verloren gehen. Sofern die Postgres Installation nicht hochverfügbar ausgelegt ist, kann es auch zu einem Datenverlust bei regulären Wartungsarbeiten oder Ausfällen an dem Datenbankserver kommen. Um den

genannten Problemen entgegen zu wirken, kam die Idee auf, einen Message Bus (auch Message Queue genannt) zu installieren. [TM]

7.2 Message Bus

Ein Message Bus ist eine Applikationen, welche auf einem oder mehreren Servern betrieben wird und als Middleware agiert. Über diverse Netzwerkschnittstellen ist es möglich, Daten auf diesen Bus zu senden. Jedes Datenpaket kann mit Tags versehen werden. Ein Tag kann ein beliebiger Text sein, zum Beispiel der Name des Quellservers, Uhrzeit an dem das Paket erstellt wurde oder eine Retention Period.

Auf der anderen Seite des Busses können Applikationen über die gleichen Schnittstellen auf die Daten zugreifen und diese dem Bus entnehmen. Hierbei kann auf die Tags gefiltert werden, um nur bestimmte Daten zu entnehmen.

Im Worst Case (dem schlimmsten möglichen Zustand) ist es möglich, dass die Datenbank für mehrere Stunden oder Tage nicht erreichbar ist. Hierzu unterstützen Message Busse eine Retention Period. Dieser Wert beschreibt wie lange ein Datensatz gültig ist. Wenn er zu alt ist, wird er automatisch gelöscht und vom Bus genommen. Der Bus kann somit nicht über die Zeit volllaufen. Im Projektteam gab es bereits Erfahrung mit der Message-Bus-Software RabbitMQ, weshalb diese für den ersten Prototypen gewählt wurde.

Der Messagebus wird zwischen collectd und der Datenbank installiert. Collectd ist nicht in der Lage, direkt SQL zu sprechen. SQL ist allerdings die einzige Schnittstelle, welche von Postgres bereitgestellt wird. Die Daten müssen also transformiert werden. Hierfür bietet sich das bereits zuvor evaluierte Logstash an (5.1.8). Logstash besitzt ein Plugin, um vom Bus zu lesen (vgl [BV17]). Zusätzlich existiert ein Plugin, um über JDBC mit Datenbanken zu kommunizieren (vgl. [Sou16]). [TM]

7.3 JVM

Die JVM (Java Virtual Machine) ist eine virtuelle Maschine zum Ausführen von Java Bytecode. Sie stellt eine abgekapselte Umgebung (Sandbox) bereit in der der Code ausgeführt wird. Dies bringt nicht nur eine erhöhte Systemsicherheit, da die Applikation in der JVM nicht direkt mit dem Hostsystem kommunizieren kann. Es hilft auch beim Multithreading. Die JVM ist in der Lage, beliebig viele Prozesse zu starten und zu verwalten.

Logstash selbst ist in Ruby geschrieben. Die Ausführung selbst erfolgt aber innerhalb einer JVMNormalerweise wird Ruby Code mit dem MRI Interpreter in Maschinencode übersetzt und direkt auf dem Betriebssystem ausgeführt. Der Interpreter JRuby übersetzt den Ruby Code in Java Bytecode, dieser wird dann von der JVM in einem Prozess ausgeführt und nicht direkt auf dem Betriebssystem. Dies ermöglicht es, Multithreading bei der Entwicklung des Ruby Codes zu ignorieren. Hierfür muss keine Funktionalität programmiert werden, da dies automatisch von der JVM übernommen wird. Ein normales Ruby Script wird nur auf einem Prozessorkern ausgeführt. Ein einzelner Kern kann aber die erwartete Datenmenge nicht aus dem Message Bus abfragen und verarbeiten. Hier kann die Architektur stark von der JVM profitieren. Sie kümmert sich darum, dass alle zur Verfügung stehenden CPU-Kerne belastet werden und startet entsprechend viele Prozesse mit Logstash. Somit erhält man eine sehr einfache Möglichkeit der Datenverarbeitung mit mehreren Prozessorkernen (vgl. [Tea16b]). [TM]

7.4 JDBC

JDBC (Java Database Connectivity) ist die Schnittstelle der JVM zu Datenbanken. Sie stellt generische Funktionen bereit, um mit Datenbanken zu kommunizieren. Für jede Datenbank mit der kommuniziert werden soll, wird ein Treiber benötigt. Dieser beschreibt die spezifischen Eigenschaften eines bestimmten DBMS für die JDBC. Er kommuniziert auf der einen Seite mit dem DBMS und auf der anderen Seite mit der JDBC. Für Logstash gibt es ein Plugin, welches die JDBC-Schnittstelle innerhalb des JRuby Bytecodes nutzbar macht. Somit ist Logstash in der Lage, in eine Datenbank zu schreiben. Hierfür musste ausschließlich die Kommunikation mit der JDBC programmiert werden. Postgres bietet einen Treiber für JDBC an (vgl. [Gro16b]). [TM]

7.4.1 JDBC Treiber Typen

Es gibt vier verschiedene Typen von JDBC-Treibern. Typ vier ist besonders effizient, dieser wird auch von den Postgres Entwicklern bereitgestellt. Im Folgenden sind die einzelnen Typen erklärt (vgl. [Tea16a]). [TM]

7.4.2 JDBC-ODBC Treiber

Hierbei spricht die JDBC-Schnittstelle einen sehr generischen ODBC Treiber an. Dieser kommuniziert mit einer Bibliothek des DBMS. Der ODBC-Treiber muss auf jedem System vorhanden sein, welches die JVM-Applikation ausführt. Die Bibliothek kann, zusammen mit der Datenbank selbst, auf einem anderen System im Netzwerk oder auch lokal betrieben werden.

Die Vorteile sind:

- Sehr einfache Implementierung der JDBC, da ein Großteil der Arbeit von der ODBC übernommen wird.
- Sehr einfacher Austausch der ODBC, da die Schnittstelle generisch ist.
- Jede Datenbank, die SQL nutzt, bietet eine ODBC-Schnittstelle, somit ergibt sich eine lange Liste an alternativen DBMS, welche genutzt werden können.

Die Nachteile sind:

- Durch die große Anzahl an Komponenten, durch die die Daten fließen, ergeben sich viele Fehlerquellen.
- Da alle Schnittstellen sehr generisch sind, kann dort keine Optimierung auf ein spezifisches DBMS erfolgen.

[TM]

7.4.3 Nativer API-Treiber

Die JDBC-Schnittstelle spricht hier über den Treiber direkt die Bibliothek des DBMS an. Dafür muss der Treiber als auch die Bibiliothek auf dem gleichen System vorhanden sein, wie die JVM-Applikation. Die Datenbank selbst kann auf dem gleichen Server oder über das Netzwerk angesprochen werden.

Die Vorteile im Vergleich zum JDBC-ODBC-Treiber:

- Eine höhere Performance, da Teile direkt in Java geschrieben sind.
- Geringere Fehleranfälligkeit, da eine Komponente weniger vorhanden ist.

Die Performance ist besser als beim JDBC-ODBC-Treiber, aber noch nicht optimal. Außerdem ist es nicht immer möglich, Zusatzsoftware, wie die Bibliotheken, auf dem gewünschten System zu installieren. Es gibt Firmen, die dies untersagen. Dies trifft allerdings nicht in diesem Projekt zu, weshalb es nicht als Negativpunkt gewertet wird. [TM]

7.4.4 Netzwerkprotokolltreiber

Datenbankoperationen werden an eine weitere, auch in Java geschriebene, Netzwerkapplikation geschickt (auch Middleware genannt). Diese spricht dann die Datenbank an, welche oft auf einem dritten System installiert ist. Hier muss keine Drittsoftware auf der eigentlichen Applikation installiert werden.

Die Vorteile sind:

- Die Middleware kann sich dediziert um SQL kümmern und von Fachleuten auf dem Gebiet entwickelt werden.
- Datenbanklogik muss nicht in der eigentlichen Applikation implementiert werden. Die Entwickler der Applikation können sich auf die Features der Applikation konzentrieren.
- In der Middlerware kann sehr einfach Auditing erfolgen.

Die Nachteile sind:

- Die eigentliche Applikation benötigt zwingend eine Netzwerkfunktionalität.
- Die Datenbankoperationen erfolgen über zwei Netzwerkverbindungen, was zu einem Overhead führt.
- Der Entwicklungsaufwand ist bedeutend komplexer als bei anderen Treibern, sofern man die Middleware selbst implementiert und nicht auf fertige Lösungen zurückgreifen kann.

[TM]

7.4.5 Nativer JDBC-Treiber

Hier liefern die Datenbankentwickler einen nativen Treiber. Dies bedeutet, dass er komplett in Java geschrieben ist. Er wird beim Starten der Applikation geladen und erlaubt es der JDBC-Schnittstelle, direkt mit der Datenbank zu kommunizieren. Es werden keine weiteren Bibliotheken benötigt. Die Datenbank kann auf dem gleichen System wie die Applikation laufen oder auch über das Netzwerk angesprochen werden.

Die Vorteile sind:

- Dieser Treiber bietet die größtmögliche Performance, da er Anfragen von der JDBC auf das spezifische DBMS optimieren kann.
- Der Overhead wurde auf ein Minimum reduziert.

- Es werden keine zusätzlichen Schnittstellen benötigt, was die Fehleranfälligkeit reduziert.

Diese Art von Treiber bietet aus Sicht des Nutzers keine bekannten Nachteile. Die Entwicklung benötigt für diese Art die meiste Arbeit, weshalb nicht jedes DBMS solche Treiber zur Verfügung stellt. Für Postgres existieren solche Treiber. Logstash ist somit in der Lage, besonders effizient auf die Datenbank zuzugreifen. [TM]

7.5 JDBC in Kombination mit der JVM

Die Kombination aus JVM und JDBC bietet einen großen Vorteil für das Projekt. Logstash ist in der Lage, die Auslastung der Datenbank über JDBC zu ermitteln. Bei einer zu hohen Last werden die Anzahl der Prozesse limitiert was in weniger Anfragen zur Datenbank resultiert. Sobald die Datenbank wieder eine geringere Auslastung hat, kann die JVM eigenständig mehr Prozesse starten und die Datenmenge erhöhen. Somit verhindert man Performanceprobleme auf der Datenbank bei komplexen Abfragen, welche über die API gestellt werden.

Anstatt die Datenbank als limitierenden Faktor zu nutzen (dem Ziel von Logstash), kann dies auch mit der Quelle erfolgen. Der Message Bus ist in diesem Setup die Quelle für Logstash. Die JVM kann anhand der vorhandenen Datenmenge im Bus beliebig viele Prozesse starten, um die Datenmenge möglichst schnell abzufragen. Dies wäre in einem Szenario sinnvoll, in dem der Speicher im Bus besonders gering ist und die Datenbank eine garantierte Leistung erbringen kann. Dies ist nur möglich, wenn Logstash der einzige Dienst ist, welcher mit der Datenbank interagiert. Dieses Szenario ist innerhalb des Projektes nicht möglich, weshalb es nicht weiter verfolgt wurde bei der Implementierung. [TM]

7.6 Prototyp 2

Die Implementierung des Message Busses stellte sich als sehr komplex heraus. RabbitMQ ist zwar die Software in dem Bereich mit der größten Verbreitung, aber auch sehr komplex zu betreiben und aufzusetzen. Es war dem Projektteam nicht möglich, RabbitMQ an collectd anzubinden. Hierfür sind diverse Einstellungen über die RabbitMQ CLI notwendig. Diese konnte sich nicht gegenüber dem RabbitMQ Server authentifizieren. Es wurde keine sinnvolle Fehlermeldung ausgeben. Die Entwickler von RabbitMQ konnten ebenfalls keine Lösung für das Problem liefern. Außerdem ist dem Projektteam aufgefallen, dass der erste Prototyp unnötig komplex

ist. Das Konzept eines Message Busses passt sehr gut in das Projekt, allerdings muss dafür keine eigenständige Software genutzt werden. Während der Suche nach Alternativen fiel auf, dass auch Postgres als Bus genutzt werden kann.

Bei der Analyse von Grafana hat sich herausgestellt, das dort keine granulare Rechteverwaltung möglich ist, wenn direkt mit einer Datenbank interagiert wird. Da die selbst entwickelte API bereits Endpunkte für Analysen bereitstellen muss, kann diese simpel erweitert werden, um auch Daten an Grafana zu liefern. Somit liefert nun die API Daten an Grafana und Grafana kann nicht mehr direkt mit der Datenbank sprechen. Die API ist in der Lage, alle Anfragen von Grafana passend zu filtern und zu validieren. Eine Visualisierung des Prototypen findet sich unter 13.5. [TM]

7.7 Automatisierung mit Puppet

Die Firma Puppet Inc. entwickelt die Software puppet. Es ist eine quelloffene Konfigurationsmanagement- und Automatisierungssoftware. Sie erlaubt es in einer deklarativen DSL, den gewünschten Systemzustand zu beschreiben. Die Software realisiert dies. In Modulen wird der puppet Code gebündelt. Jedes Modul enthält Code, welcher zur Automatisierung einer bestimmten Software benötigt wird. Über die Plattform `https://forge.puppet.com/` können diese Module ausgetauscht werden. Es existieren bereits Module zur Verwaltung von Postgres, collectd, Grafana und Logstash. Jedes Modul hat seinen eigenen Namespace und bündelt ein oder mehrere Klassen. Informationen über ein Modul werden in der `metadata.json` Datei gespeichert. Diese befindet sich in jedem Modul und enthält folgende Informationen:

- Name des Modules
- Autor
- Lizenz
- Link zum Quellcode
- Eine Kurzbeschreibung
- Liste der unterstützten Betriebssysteme
- Eine Liste an anderen Modulen, welche für den Betrieb dieses Moduls benötigt werden
- Die unterstützten Puppetversionen

[TM]

7.7.1 Strukturierung im puppet Code

In der DSL werden immer Ressourcen beschrieben. Es gibt verschiedene Typen von Ressourcen. Die wichtigsten sind:

- Package - Verwaltet die Installation und Deinstallation von Paketen
- Service - Managt Dienste auf dem System
- File - Erstellt konfigurationsdateien und Verzeichnisse mit passenden Rechten

Eine einzelne Ressource benötigt nur sehr wenig Code. Jede Deklaration einer Ressource besteht aus drei Teilen (Beispiel unter Listing 13.19).

Erst wird der Typ genannt, in diesem Fall `package`. Darauf folgt ein Name für die Deklaration und zum Schluss eine Liste an Parametern. Der Name der Ressource muss einzigartig sein. In diesem Fall ist es der Name des Paketes, welches in der Version 2.0.2-1 installiert wird. Puppet bietet 48 verschiedene Typen. Diese sind fest in puppet verankert und werden von der Firma Puppet gepflegt. Außerdem kann puppet mit eigenen Typen erweitert werden. Für jeden Typen gibt es einen Provider. Dieser implementiert den Typen auf einer bestimmten Plattform. Das Grafana-Modul besitzt eigene Typen, welche Dashboards und Datenquellen verwalten. Die Konfiguration von eben jenen erfolgt nicht über normale Textdateien, sondern über eine API oder Webseite. Über die eigenen Typen kann Puppet auch hiermit interagieren und nicht nur die Installation, sondern auch die komplette Konfiguration automatisieren (vgl. [Inc17]). Da puppet deklarativ und nicht imperativ arbeitet, ist die Anordnung der Ressourcen egal. Der Interpreter analysiert alle Ressourcen, erkennt automatisch Abhängigkeiten und sortiert diese zu einem gerichteten Graphen. Die Ressourcen werden deshalb nicht in der Reihenfolge abgearbeitet, in der sie notiert sind.

Dieses modulare Konzept ermöglicht es, puppet auf einer großen Anzahl von verschiedenen Betriebssystemen und Hardwaretypen zu betreiben. Durch die Abstraktion kann Code sehr häufig wiederverwendet werden.

In einem Puppet-Profil werden mehrere Module und Ressourcen deklariert, um eine bestimmte Aufgabe zu realisieren. Dieses Profil ist aus Sicht des Codes eine normale Klasse. Mehrere Profile werden in einem Modul gebündelt. Ein Profil wird einem oder mehreren Nodes zugewiesen. Im Rahmen des Projekts werden diverse Profile erzeugt, um die einzelnen benötigten Komponenten gebündelt zu automatisieren. Ein beispielhaftes Profil für Grafana findet sich unter 13.4. Dies installiert Grafana und richtet außerdem ein Dashboard und eine Datenquelle ein. [TM]

7.7.2 Ablauf eines Puppet Runs

Puppet kann nach dem Client-Server-Prinzip arbeiten. Der Agent läuft dabei auf jedem System, welches verwaltet werden soll. Unterstützt werden neben Linux-Distributionen auch Windows, BSD, sowie diverse Hersteller von Netzwerkequipment. Periodisch ermittelt der Agent Facts und sendet diese über eine HTTPS-Verbindung zum Puppetserver. Anhand dieser Facts wird entschieden, welche Module zu einem Katalog zusammenkompiliert werden. Der Katalog wird dem Agent zurückgeschickt. Dieser analysiert sein vorhandenes System und gleicht es mit den Beschreibungen im Katalog ab. Sobald es hier zu Differenzen kommt, wird das System dem Katalog angeglichen. Jede durchgeführte Aktion kann reportet werden. Reports können lokal gespeichert oder zu einer zentralen Datenbank gesendet werden. Die Firma Puppet bietet außerdem die quelloffene PuppetDB an. Diese kann Reports speichern. Über eine RESTful API kann man alle Reports analysieren und Abfragen. Zusätzlich können alle Facts dort gespeichert werden. Ein „Puppet Run“ beschreibt den kompletten Ablauf vom Auslesen der Facts bis zum Erstellen des Reports.

Puppet kann auch ohne Puppetserver genutzt werden. Hier erfolgt der komplette Ablauf dann lokal. Die Module müssen auf dem Server vorhanden sein auf dem auch der Agent läuft. Er kompiliert sich dann seinen Katalog selbst. Diese Vorgehensweise bietet sich gerade während der Entwicklung an. In großen Produktivumgebungen mit vielen Modulen ist es oftmals sinnvoller, Puppetserver zu nutzen und nach dem Client-Server Prinzip zu arbeiten. [TM]

7.8 Postgres

Für die Implementierung der Postgres Datenbank wurde zunächst ein einzelner Server für die Bearbeitung der Anforderung aufgebaut und mit vier Nutzern ausgestattet. Ersterer ist der Administrator, oder auch root genannt, welcher nur genutzt wird um Änderungen (Abschnitt 7.8.3) am System vorzunehmen. Mit diesem werden ebenfalls die Zugriffsrechte der einzelnen Nutzer geregelt. Der zweite Nutzer besitzt die Berechtigungen in einem definierten Bereich innerhalb des Postgres-Systems die Daten und dessen Struktur zu managen. Mit diesem Benutzer wurden z.B. die Prozesse aus Abschnitt 7.8.1 und Abschnitt 7.8.4 durchgeführt. Dritter Nutzer wird von dem in Abschnitt 7.9 definierten API-System verwendet. Dieser besitzt keine direkten Lese- oder Schreibrechte auf die in der definierten Bibiliothek vorhandenen Tabellen. Er erhält lediglich ausführende Rechte auf die im Abschnitt 7.8.4 erstellten Funktionen. Vierter Nutzer wird für den im Abschnitt 7.8.1 beschriebenen ETL-Prozess im Wirkbetrieb genutzt. Er besitzt ausschließlich schreibende Rechte.

Neben den durchgeführten Aufgaben an dem Postgres-Server wurde bei den Teammitgliedern jeweils die benötigte Entwicklerumgebung aufgesetzt und eingerichtet. Mit dieser und zusammen mit einem berechtigten Nutzer, können die Teammitglieder an den Inhalten der Datenbank arbeiten. [NL]

7.8.1 ETL

Durch das Fehlschlagen der Anbindung eines Message Busses an die Datenerfassung musste eine alternative Möglichkeit der Informationsintegration zwischen der Datenhaltung und Datenerfassung entwickelt werden. Dazu wurde ein vollständig eigener Extraktions-, Transformations- und Lade-Prozess (ETL) in das System implementiert. Die konkrete Umsetzung erfolgte durch das Hinzufügen von einem eigenen Arbeitsbereich zwischen der Datenquelle und der Zieldatenbank.

Dabei findet der Extraktions-Teil zwischen dem erstellten Arbeitsbereich und der Datenquelle statt. Es werden dabei die neuen Daten in ein allgemeines Datenobjekt eingefügt und kurzzeitig zwischengespeichert. Dieses Datenobjekt hat dabei keinerlei Anforderungen an die gelieferten Daten und nimmt sie zunächst einfach auf, sodass diese nicht verloren gehen. Sollten dabei Daten nicht vollständig übermittelt werden oder während der Übertragung die Datenquelle ausfallen, beziehungsweise andere Fehler entstehen, ist dies zunächst für den eigentlichen Wirkbetrieb, also das Halten der vollständigen Daten und das Ausliefern der Daten an andere Systeme/APIs, nicht relevant. Durch die Trennung der Systeme, beziehungsweise die einzelnen Aufgabenfelder, ist nur der Verlust von neuen Daten das größte Risiko. Ein Systemabsturz durch dauerhafte Fehler beim Datenschreiben ist somit vermieden.

Sollten nun Daten in dem genannten allgemeinen Datenobjekt vorhanden sein, beginnt der Transformations-Teil. Dabei greift ein weiterer Algorithmus auf die zwischengespeicherten Daten zu und vergleicht die Inhalte von jedem neuen Datensatz mit einem Repositorium, beziehungsweise einem von dem Projektteam definierten Metadatensatz. Dieser dient dazu, eine spezifische Definition von allen in dem Datenhaltsungssystem vorhandenen Daten zu besitzen. Er wird entsprechend als Dokumentationsbasis genutzt, sodass ein Nutzer genau nachvollziehen kann, in welcher Struktur und an welchem Ort die Daten in dem Datenhaltsungssystem abgespeichert sind. Gleichzeitig besitzt das Projektteam und der spätere Administrator im Wirkbetrieb die Möglichkeit, den Dateninput über den ETL-Prozess zu kontrollieren. Darüber können zum Beispiel die Länge von einzelnen Attributen oder der Datentyp angepasst werden. Es kann jedoch sogar tiefer ins Detail gegangen werden und der genaue Inhalt eines Attributes bestimmt werden. Ist ein solches Reposi-

torium definiert, greift der Transformations-Algorithmus darauf zu und püft die neuen Daten nach den Definitionen. Sollte bei der Prüfung ein Konflikt entstehen, so kann der Prozess diesen eigenständig lösen und die neuen Daten an die Definition anpassen. Bei schwerwiegenden Fehlern, welche nicht von dem Algorithmus gelöst werden können, wurde von dem Projektteam ein Fehlerreport definiert. Dabei werden die auf Fehler gelaufenen Daten nicht entfernt und ein beliebiger Verteiler darüber informiert. Anschließend kann ein Administrator den Fehler beseitigen oder die Daten manuell in das Datenhaltsungssystem überführen.

Neben den angesprochenen Inhalten wurde im Transformations-Teil eine Aggregation der Daten zugunsten des Speicherplatzes definiert. Sofern bei aufeinanderfolgenden Datensätzen innerhalb einer Stunde der Ressourcentyp, das System und der Wert identisch ist, wird nur der erste Datensatz gespeichert. Oftmals laufen Systeme über Stunden ohne eine messbare Auslastung, somit kann bei der Speicherung auf diese Datensätze verzichtet werden. Das selbe Prinzip wird für Systeme angewandt, welche eine konstante gleiche Auslastung besitzen. Bei diesen muss ebenfalls nur der erste Datensatz innerhalb der Stunde gespeichert werden. Bei der Datenabfrage werden die fehlenden Datensätze dynamisch errechnet. Dadurch das jede Stunde mindestens ein Datensatz von jedem angebunden Server in die Datenhaltung gespeichert wird, kann ebenfalls zwischen einem ausgeschaltenden Server und einem Server ohne Nutzung unterschieden werden.

Ist der Transformations-Teil bei einem Datensatz vollständig durchgelaufen, beginnt der Lade-Teil. Dabei übergibt der Transformations-Algorithmus den vollständig korrekten Datensatz an den Lade-Algorithmus. Neben dem eigentlichen Datensatz übermittelt dieser auch die Zieldatenbank und Zieltabelle, sodass der Lade-Algorithmus nun nur noch die Daten in das entsprechende Objekt einfügen muss. Dabei wird speziell auf die Auslastung des Zielobjektes geachtet. Ist die Tabelle zum Beispiel durch einen anderen Prozess belegt, wird dies von dem Lade-Algorithmus erkannt und das Einfügen der Daten hinten angestellt. Für diesen Teil besitzt die Postgres Datenbank bereits vorhandene Frameworks, welche von dem Projektteam genutzt werden. Somit muss das Management der Prozessabarbeitung innerhalb des Datenhaltsungssystem nicht selber entwickelt, sondern nur eingebunden werden. Desweiteren prüft der Lade-Teil, ob eine in dem Abschnitt 7.8.3 erläuterte Partionierung existiert. Sollte dies nicht der Fall sein, so wird vor dem eigentlichen Einfügen der Daten, eine neue Partition angelegt und der Haupttabelle mitgeteilt. [NL]

7.8.2 Repositorium

Der im Abschnitt 7.8.1 angesprochene Metadatensatz, welcher in diesem Projekt für die Metric-Daten jedes virtuellen Servers genutzt wird, wurde zusammen mit allen Projektmitgliedern definiert und abgesprochen. Dabei wurde sich auf eine spezielle Abspeicherung der Daten geeinigt. Es handelt sich um die Verwendung von einer einzelnen Datentabelle, welche nur zwei Attribute besitzt. Bei beiden Attributen handelt es sich um ein JSON Objekt. Ersteres beinhaltet Metadaten zu dem einzelnen Datensatz. Das ist zum Beispiel eine generierte ID, um den Datensatz spezifisch selektieren zu können, oder die Informationen, seit wann der Datensatz in der Datenhaltung existiert, beziehungsweise wann und von welchem System angefasst wurde. Zweites JSON Objekt/Attribut beinhaltet den eigentlichen Datensatz, also die werthaltigen Informationen, welche auch später analysiert werden. Die Inhalte des Attributes werden klar in dem Metadatensatz definiert. Der Vorteil dieser Datenspeicherung ist zum einen die Effizienz des Speicherplatzes sowie Geschwindigkeit und zum anderen das Zusammenhalten der zugehörigen Informationen. Des Weiteren können in diesem Modell sehr einfach neue Informationen hinzugefügt werden, indem diese in dem Metadatensatz definiert werden. Dies ist ein Vorteil der Skalierbarkeit und es müssen anders als bei Datenbanken in der dritten Normalform die voneinander abhängigen Daten nicht in jeweils eine eigene Tabelle ausgelagert werden.

Da sich bei diesem Modell die Geschwindigkeit der Datenabfragen mit der Menge der Daten in einer Tabelle verlangsamt, wird die automatische Partitionierung angewandt, welche nachfolgend beschrieben wird. [NL]

7.8.3 Partitionierung

Bei der Partitionierung in dem Postgres-System muss zwischen zwei Arten von Partitionierung entschieden werden. Ersteres ist die manuelle Partitionierung durch den Administrator. Dabei werden über einen Index Datenbereiche definiert, welche logisch voneinander getrennt werden. Mit nachfolgendem Code-Prototyp kann solch eine Trennung im groben vom Administrator definiert werden.

Unter dem im Anhang 13.5 beigefügten Prototypen wird eine Haupttabelle in zwei Kinder-Tabellen aufgeteilt. Diese Kinder-Tabellen halten dabei bestimmte definierte Datenbereiche. In diesem Prototypen enthält die Kinder-Tabelle `measurement_xMINUS12Hour` nur Datensätze bei denen der Zeitstempel der beinhalteten Daten zwischen dem aktuellen Zeitpunkt und dem aktuellen Zeitpunkt minus 12 Stunden liegt. Die zweite Kinder-Tabelle beinhaltet wiederum nicht die Daten von der ersten

Kinder-Tabelle, sondern nur alle Daten, welche einen Zeitstempel innerhalb eines Tages besitzen. Diese Art von Partitionierung kann auf jedem beliebigen Attribut angewendet werden, sodass zum Beispiel auch noch CPU-Daten von RAM-Daten getrennt werden. Der Anwender oder Analyst der Haupttabelle merkt dabei nicht, dass die verwendete Tabelle partitioniert ist. Für die vollständige Implementierung wurden von dem Projektteam jedoch auch noch weitere Funktionen programmiert und implementiert. Darunter ist zum Beispiel das Verhalten der Datenbank bei dem Einfügen von Daten mit einem älteren Zeitstempel.

Bei der manuellen Partitionierung kann ein Administrator jedoch auch durch die Stärke der Partitionierungs-Intervalle an Optimierung verlieren oder sogar das System verschlechtern. Hinzu kommt ein starker administrativer Aufwand, umso mehr Partitionen eingefügt werden. Dabei kann auch die Übersichtlichkeit des Codes verloren gehen. Entsprechend bietet Postgres auch die Methode der automatisierten Partitionierung an. Dabei werden die Partitionen von dem Postgres System selbständig angepasst und die Datenspanne definiert. Dies bringt den Vorteil, dass eine schnelle Geschwindigkeit der Datenabfrage zu jedem Zeitpunkt gewährleistet ist. Der administrative Aufwand wird mit dieser Methode jedoch nicht verringert. Die manuelle Partionierung hat einen kontinuierlich gleich bleibenden administrativen Aufwand. Bei der automatisierten Partionierung muss vor allem zu Begin mehr Zeit investiert werden. Erst bei der optimalen Einstellung der Konfigurationsmöglichkeiten verringert sich der administrative Aufwand deutlich.

Das Projektteam hat sich entschieden, die automatische Partionierung zu wählen. Dies gewährleistet nicht nur eine dauerhaft schnelle Datenhaltung, sondern ist auch kundenfreundlicher, da dieser nicht dauerhaft einen Administrator für die Partionierung einstellen muss. Speziell in diesem Projekt wurde sich zuletzt auf die Partitionierung in Bezug auf die Zeit des Datensatzes geeinigt. Dabei wird für jede Stunde am Tag eine eigene Partionierung erstellt. Somit wird ein CPU-Wert um 20:13 Uhr am 22.04.2017 in der Partition `measurement_2017_04_22t20` gespeichert. Die genaue Implementierung der Methode kann aus dem Postgres-Code im Projektanhang entnommen werden. [NL]

7.8.4 Datenschutz und Analysen

Für den Zugriff auf die in dem Datenhaltsungssystem vorhandenen Daten wurde von dem Projektteam eine weitere Sicherheitsebene eingefügt. Diese definiert, dass ein Systembenutzer, also ein Account, welcher Daten von dem System abfragt, keinen direkten Zugriff auf Tabellen im Datenhaltsungssystem erhält. Stattdessen wird für jeden Anwendungsfall, also jede Datenbankabfrage, beziehungsweise analytische

Fragestellung, eine eigene Postgres-Funktion bereitgestellt. Der Systembenutzer kann dabei ausschließlich die für ihn freigeschalteten Funktionen ausführen. Bei dem Systembenutzer für die API sind dies zum Beispiel nur Funktionen zur Abfrage von definierten Daten.

Dieses Verfahren hat den großen Vorteil, dass wenn einer der Systembenutzer entfremdet wird, dieser keinen Schaden an dem Datenhaltsungssystem ausüben kann. Des Weiteren kann der Systemadministrator genau kontrollieren, wer auf welche Daten Zugriff haben soll. Ein weiterer Vorteil ist, dass durch die vordefinierten, analytischen Fragestellungen bei jedem Benutzer die gleichen Daten ausgegeben werden und somit eine falsche Aussage, aufgrund von verschiedenen Implementierungsweisen, vermieden wird. Zu diesen analytischen Fragestellungen gehört ebenfalls die beschriebene Trendgenerierung aus Abschnitt 5.1.1 und das in Abschnitt 7.8.1 erläuterte dynamische Errechnen der im ETL verzichteten Datensätze.

Zusätzlich wurde ein Benutzer mit vollständigen oder kritischen Zugriffsrechten auf der Datenbank eingerichtet, sodass diese sich nur lokal auf dem Datenbankserver verbinden können. Das bedeutet, dass mit diesen Benutzern nicht von anderen Systemen direkt auf die Datenbanktreiber zugegriffen werden kann. Dazu ist zunächst eine SSH-Verbindung auf dem Server vonnöten. SSH ist ein speziell geschütztes Protokoll für die Fernadministration von Rechnersystemen. Man kann dabei einrichten, dass das Einloggen nicht mit einem Passwort, sondern ausschließlich mit einem, von dem System-Administrator bereitgestellten, Key-Dokument möglich ist. Dies bringt einen weiteren Sicherheitsfaktor zum Schutz der Daten auf dem Datenbanksystem.

[NL]

7.9 API-System

Bei der Entwicklung des API-Systems wurde sich zunächst auf die Implementierung der Postgres und Grafana-Anbindung konzentriert. Voraussetzung für die eigentliche Entwicklung ist eine funktionierende und auch möglichst dem Entwickler vertraute Entwicklungsumgebung. Bei der Swagger API wurde dazu die Umgebung mit dem Namen „Spring Tool Suite“ verwendet. Diese basiert auf dem Entwicklerstudio „Eclipse“ und war dem Projektteam bereits bekannt. Die Spring Tool Suite ist dabei für die Entwicklung einer Swagger API optimiert und liefert alle benötigten Grundlagen mit. Für die Einbindung weiterer Funktionen wurden die entsprechenden Maven-Abhängigkeiten eingefügt. Diese ermöglichen die einfache Integration von benötigten Bibliotheken oder Code-Algorithmen. Das Spring Tool Suite liefert

ebenfalls eine eigene Testumgebung mit, in welcher der Entwickler für das Projekt angepasste Tests definieren und durchführen kann.

Nach der Einrichtung des Entwicklerstudios wurde zunächst das API Modell definiert. Es handelt sich dabei um die Datenstruktur, welche das API-System von einem anderem System zugeschickt bekommt, beziehungsweise zuschickt (siehe auch Abschnitt 5.3.3). Da sich das Projekt auf die Zusammenarbeit mit dem Grafana-System konzentriert, wurde mit der Modellierung dieser Datenstrukturen begonnen. Dabei arbeitet Grafana nur mit vier URLs (Endpunkte) der API. Die erste ist dafür zuständig zu prüfen, ob das API-System überhaupt erreichbar, beziehungsweise vorhanden ist. Die zweite wird genutzt, um bei dem API System anzufragen, welche Daten es besitzt. Die dritte führt die eigentlichen Datenanfragen, basierend auf der zweiten URL, aus und liefert die zu anzeigenden Daten zurück. Die letztere ermöglicht es, einzelne Informationen in einem Graphen hervor zu heben. Alle vier URLs besitzen eine eigene Anfrage-Methode und Datenstruktur an der API und erwarten auch unterschiedliche Antworten von der API in Struktur und Inhalt. Neben diesen vier URLs wurde ebenfalls das Extrahieren der in Abschnitt 5.3.3 erläuterten Organisations-ID implementiert.

Nachdem diese URLs und Modellstrukturen definiert wurden, schaute das Projektteam auf die genaue Zusammenarbeit zwischen Grafana und der -. Dies hatte den Vorteil, dass die darauf folgende Modellierung der Datenabfrage bei dem Datenhaltsungssystem besser nachvollziehbar war. Entsprechend wurden die im Abschnitt 7.8.4 definierten Postgres-Funktionen nun auch in dem API-System integriert.

Zusätzlich wurde die in Abschnitt 7.8.1 erläuterte Rückrechnung der Daten integriert.

[NL]

7.10 Realisierung Grafana

Nachdem nun die Evaluierung aus Abschnitt 5.4.3 abgeschlossen ist, konnte mit der Installation und Konfiguration von Grafana begonnen werden. Die Installation von Grafana wird nachfolgend für die Linux Distribution ArchLinux beschrieben. Grafana unterstützt ebenfalls auch andere Distributionen, jedoch können hier die entsprechenden unten aufgeführten Befehle abweichen. Die zu unterstützen Distributionen können der Grafana Evaluierung aus Abschnitt 5.4.3 entnommen werden. Damit die eigentliche Konfiguration durchgeführt werden konnte, musste zuvor das Installationspaket von Grafana heruntergeladen und installiert werden. Auf einem ArchLinux-Betriebssystem wird dieses Paket mit folgendem Befehl installiert:

```
sudo pacman -S grafana
```

Pacman ist ein Paketinstaller mit dem weitere Software aus einem Repository heruntergeladen und installiert werden kann. Dabei durchsucht der Paketinstaller die gesamten vordefinierten Repository URLs nach Grafana und installiert hier die aktuellste vorliegende Version auf der entsprechenden Distribution.

Nachdem Grafana erfolgreich heruntergladen und installiert worden ist, muss dieses mit folgendem Befehl gestartet und falls gewünscht im Autostart von Archlinux hinterlegt werden. Mit folgendem Befehl wird der Grafana-Dienst gestartet:

```
systemctl start grafana.service
```

Um Grafana im Autostart zu hinterlegen, wird folgender Befehl benötigt:

```
systemctl enable grafana.service
```

Nachdem der `grafana.service` gestartet wurde, kann die Webseite über einen Browser bereits aufgerufen werden. Der Aufruf hierzu lautet wie folgt: `http://<Adresse-des-grafana-servers>:3000`. Bei der Installation von Grafana wird bereits ein vordefinierter Administrator-Benutzer mit Passwort festgelegt. Die Zugangsdaten hierzu lauten:

- Benutzername: admin
- Kennwort: admin

Diese Logindaten können nach dem erfolgreichen Login angepasst und verändert werden. Anschließend wird bereits das Home Dashboard angezeigt, von wo aus die Datenvisualisierung konfiguriert werden kann.

Das Dashboard ist wie folgt aufgebaut: Links gibt es ein Grafana-Symbol, worüber alle Konfigurationen, wie Plugin-Verwaltung, Datenquellen oder Benutzereinstellungen konfiguriert werden können. Rechts neben dem Grafana-Symbol wird immer die aktuelle Ebene angezeigt auf der der Administrator sich befindet. Wenn der Administrator sich hier auf dem Home Dashboard befindet, so wird hier „Home“ angezeigt und die Übersicht der einzelnen Dashboards. Sollte der Administrator in den Einstellungen unter „Data Sources“ sein, so wird an der vorgesehenen Stelle „Data Sources“ eingeblendet. Dies soll dem Administrator zusätzlich helfen, eine bessere Übersichtlichkeit zu erhalten. Auf der rechten Seite des jeweiligen Dashboards hat der Administrator oder Benutzer die Möglichkeit, den Aktualisierungstimer der anzuzeigenden Daten zu verändern. Standardmäßig steht dieser auf sechs Stunden, kann aber beliebig angepasst werden. Eine Darstellung des Dashboards, kann dem Anhang 13.6 entnommen werden.

Zu Beginn sollte hier zuerst die Datenquelle von der Grafana die ermittelten Daten erhält, konfiguriert werden. Hierzu navigiert der Administrator auf das Grafana-Symbol und wählt hier anschließend „Data Sources“ aus. Nun kann hier unter „Add data source“ eine neue Datenquelle definiert werden. Grafana bietet bereits eine Vielzahl an Auswahlmöglichkeiten von Datenhaltungssystemen an. Diese können nach Belieben durch Plugins erweitert werden. Hierauf wird im weiteren Verlauf der Implementierung eingegangen. Nachdem hier alle Werte ausgefüllt worden sind, kann die Datenquelle über die Schaltfläche „Add“ hinzugefügt werden. Grafana prüft bereits beim Speichern einer jeden Datenquelle, ob diese erreichbar ist und Daten abgerufen werden können.

Sobald die Datenquelle hinterlegt wurde, muss zunächst ein neues Dashboard angelegt werden, damit die Graphen definiert werden können. Das neue Dashboard kann unter dem Reiter Dashboard angelegt werden. Anschließend erhält der Administrator bereits eine vordefinierte Auswahlmöglichkeiten von Graphen, Tabellen oder auch Singlestats. Diese können je nach Verwendungszweck hinzugefügt werden. Eine Beschränkung, wie viele Graphen maximal auf einem Dashboard angelegt werden können, ist laut der Dokumentation von Grafana nicht definiert, jedoch kann dies die Performance der Webseite für den Benutzer negativ beeinflussen.

Nachdem der gewünschte Graph ausgewählt worden ist, kann dieser angepasst und bearbeitet werden. Hierzu wird der zu editierende Graph ausgewählt und anschließend auf „edit“ geklickt. Unterhalb des Graphen öffnet sich dann ein Menü, in dem der grundsätzliche Aufbau, die Metrik, die Axen und diverse weitere Konfigurationsmöglichkeiten für diesen Graphen definiert werden können. Zunächst wird die Metrik des Graphen angepasst, damit die ersten Graphen visuell dargestellt werden können. Unter dem Abschnitt 7.10.1 wird genauer auf die Erstellung von Graphen eingegangen. Im Nachfolgenden ist eine Beispielabfrage für die Visualisierung von dem verwendeten Arbeitsspeicher einer Instanz. Die in dem Bild zu sehenden, abgefragten Daten wurden aus einer InfluxDB erhalten:

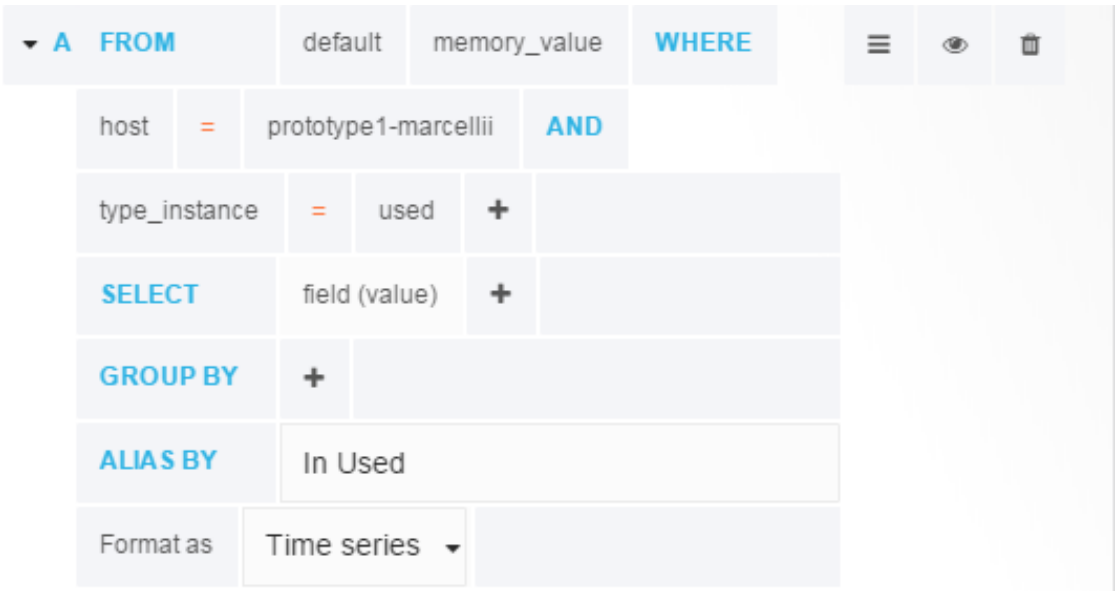

Abbildung 7.1: Beispielabfrage für eine Graphenvisualisierung

Nachstehend können noch diverse Feinkonfigurationen für einen Graphen spezifisch vorgenommen werden. Dieses Vorgehen wird nun für jeden Graphen durchgeführt, dessen Werte visualisiert werden sollen. Nachdem alle Graphen erstellt worden sind, muss das Dashboard abgespeichert werden, damit die Konfigurationn nicht verloren gehen. Sollte das Speichern vom Administrator vergessen werden, so warnt Grafana bei dem Wechsel eines Dashboards, das die Konfigurationen verloren gehen, falls diese nicht abgespeichert werden. Damit wie in dem Projekt gefordert eine API Schnittstelle angesprochen werden kann, benötigt dies eine Plugininstallation. Es stehen viele verschiedene Plugins auf der Grafana Webseite zur Verfügung. Zu jedem Plugin gibt es eine kurze Erläuterung, sowie eine Schritt für Schritt Anleitung zur Installation dieses Plugins. Die Plugins werden über einen Kommandozeilenbefehl installiert und sind anschließend über die Grafana-Oberfläche verfügbar.

Die Grundkonfiguration sind nun abgeschlossen und Grafana kann verwendet werden. Um weiteren Benutzern Zugriff zu dem Dashboard zu gewähren, können unter dem Reiter Admin > Global Users weitere Benutzer angelegt werden und Zugriff auf die Webseite von Grafana erhalten. Insbesondere hier kann beschränkt werden, welche Benutzer welche Einstellungen und Graphen sehen können. Dies wird in Grafana mit der Organisations-ID realisiert. Ein neu angelegter Benutzer wird dabei mindestens einer Organisation zugeordnet. Dies ist in diesem Projekt zum Beispiel für einen einzelnen Kunden von einer virtuellen Instanz vorgesehen. Durch die in Punkt 5.3.3 erläuterte Mapping-Tabelle wird sichergestellt, dass der Benutzer

nur die zu ihm zugeordneten Daten sehen kann. Administratoren oder andere Mitarbeiter eines Unternehmens, wie zum Beispiel ein Kundenberater, erhalten mehrere Zuordnungen von Organisations-IDs. Somit erhalten diese Zugriff auf die Dateninhalte des Kunden und können somit ihre Aufgaben, wie die Beratung für ein neues Produkt oder Verbesserung der Gesamtperformance, durchführen. Andersherum muss für einen Kunden mit mehreren gemieteten virtuellen Instanzen kein neuer Benutzer angelegt werden, stattdessen wird seinem bestehenden Benutzer eine weitere Organisations-ID hinzugefügt.

Weitere Administrator Benutzerrollen können an zum Beispiel lokale Benutzer oder auch für LDAP-Benutzer vergeben werden. Sollte ein Unternehmen die LDAP-Anbindung für die Authentifizierung an Grafana wünschen, so muss in dem Verzeichnis von der Grafana-Installation die Datei `grafana.ini` angepasst werden. Alle Einstellungen, die in dieser Datei vorgenommen werden, beziehen sich auf die Servereinstellungen, welche vom Kunden individuell angepasst werden können. Nachdem die Änderungen abgeschlossen sind, muss der Grafana-Dienst neugestartet werden. Dies wird mit folgendem Befehl durchgeführt:

```
systemctl restart grafana.service
```

Diese Konfiguration beschreibt die manuelle Installation und Konfiguration von Grafana. Um hier jedoch nicht bei jedem einzelnen Kunden diese Anpassung vornehmen zu müssen, kann die Installation, wie auch die Konfiguration, automatisiert werden.

Die Automatisierung erfolgt mit einem Puppet-Profil. Das Puppet-Profil hat verschiedene Vorteile. So ist dieses Betriebssystem unabhängig, Versionen von Grafana können vorgegeben und Konfigurationen werden direkt während der Installation in die Software implementiert. Um das Puppet-Profil aufrufen zu können, wird folgender Befehl benötigt:

```
sudo puppet apply grafana.pp
```

Innerhalb des Profils wird eine „grafana.pp“ Datei erstellt und anschließend mit dem Puppet-Profil aufgerufen. In dieser ist definiert mit welcher Version Grafana installiert, welche Datenquelle verwendet oder welches Dashboard bereits vordefiniert übergeben werden soll. Hier können zusätzlich noch verschiedene andere Parameter, wie LDAP-Anbindung übergeben werden. Dies erleichtert dem Administrator die Konfiguration, da dieses Profil nur einmal auf der Distribution ausgeführt werden muss, um Grafana zu installieren und zu konfigurieren. Eine zusätzliche Konfiguration nach der Installation von Grafana wird nur dann benötigt, wenn spezielle Anforderungen vom Kunden gewünscht sind. Dashboards werden in der

Konfigurationsdatei des Puppet-Moduls für Grafana als JSON definiert und können im Nachhinein angepasst werden.

Im Anhang 13.4 befindet sich ein Beispiel für eine solche Konfigurationsdatei.

Während der ersten Tests für die Automatisierung der Installation stellte sich heraus, dass es hier ein Problem gibt, die Konfiguration an Grafana zu übergeben. Das Puppet Profil prüft den Login auf die Webseite nicht, da es davon ausgeht, dass die Anmeldemaske der Webseite unmittelbar nach dem Neustart von Grafana erreichbar ist. Das Problem liegt nicht in der grafana.pp, sondern in dem eigentlichen Puppet-Modul, welches die Ressourcentypen `grafana_dashboard` sowie `grafana_datasource` bereitstellt. Hierzu wurde ein Bugreport beim Entwickler des Puppet-Moduls erstellt (vgl. [Mar17]). [MR]

7.10.1 Graphen definieren

Sobald die Grundkonfiguration von Grafana wie in Abschnitt 7.10 beschrieben, durchgeführt worden ist, können als nächstes die Graphen konfiguriert werden. [MR]

General

Zu Beginn wird unter dem Reiter `General` ein aussagekräftiger Titel, sowie Beschreibung des Graphen hinterlegt. In dem Beispiel, welches im Anhang 13.7 enthalten ist, wird der Graph für die Prozessorauslastung definiert. Unter dem Reiter `General` können alle Konfigurationen für das Aussehen des Graphen, sowie Verlinkungen zu anderen Dashboards oder direkten URL-Adressen zu Webseiten hinterlegt werden. Diese werden anschließend, wie im Anhang 13.8 auf der Dashboard-Übersicht dargestellt. Dies dient der besseren Übersichtlichkeit, wenn ein Administrator oder die Führungsebene diese Graphen analysieren möchte. Im zweiten Schritt werden nun die einzelnen Graphen definiert. [MR]

Metrics

Hierzu muss unter dem Reiter `Metrics` zunächst ausgewählt werden, von welcher Datenquelle die erhaltenen Daten stammen und anschließend über die Schaltfläche `Add Query` hinzugefügt werden. Grafana unterstützt unter anderem viele verschiedene Datenquellen, wie zum Beipsiel OpenTSDB (siehe Abschnitt 5.2.2) oder Elasticsearch (siehe Abschnitt 5.2.2). In dem Projekt wird für die Datenerhaltung

eine HTTPS-API Schnittstelle verwendet, welche im Laufe des Projektes entwickelt und entworfen worden ist. Die HTTPS-API liefert dabei ein JSON-Objekt. Sie wird nicht mittels einer SQL-Abfrage abgerufen, sondern über eine HTTPS Schnittstelle. Innerhalb der API werden bereits vordefinierte Werte zurückgegeben. Wenn als Datenquelle eine Datenbank angesprochen werden soll, so kann zunächst eine Art `SELECT` SQL-Abfrage hinterlegt werden. Innerhalb der SQL-Abfrage wird definiert, von wo die erhaltenen Daten stammen, sowie welche Werte abgerufen werden sollen. Ebenfalls kann diese SQL-Abfrage mit anderen SQL Abfragen innerhalb einer Gruppe zusammengefasst werden. Ein Beispiel hierzu ist dem Anhang 13.9 beigefügt. [MR]

Axes

Im nebenstehenden Reiter `Axes` kann definiert werden, ob eine Beschriftung für die X- oder Y-Achse benötigt wird. Zur Verdeutlichung ist dem Anhang 13.10 ein Bild hierzu beigefügt. Zusätzlich kann auch die entsprechende Einheit für diese Werte definiert werden, denn die erhaltenen Werte aus der Datenbank oder der API sind meist nur Ganz- oder Kommazahlenwerte, welche keine Einheiten übergeben bekommen. [MR]

Legend

Unter dem Menüpunkt `Legend` wird die Legende für diesen Graphen definiert. Es kann dort eingestellt werden, welcher Minimal-, Maximal-, Aktuell- und Durschnittswert innerhalb der Analyse ausgegeben worden ist und ob diese Werte als Tabelle ausgegeben werden sollen. Diese Ausgabe erfolgt direkt unterhalb des Graphen und soll für eine bessere Übersichtlichkeit für den Administrator dienen, da dieser die Minimal- und Maximalwerte, sowie Aktuelle- und Durchnittswerte, welche innerhalb der Analyse ausgegeben worden sind, sehen kann. Ein Beispiel hierzu ist dem Anhang 13.11 beigefügt. [MR]

Display

Nachdem nun die Axenbeschriftung und Legende angepasst worden sind, gibt es unter dem Punkt `Display` drei Einstellungsmöglichkeiten, `Draw options`, `Series overrides` und `Thresholds`. Der Aufbau kann dem Anhang 13.12 entnommen werden. Unter dem Punkt `Draw options` kann das Aussehen des Graphen angepasst werden. So kann bei jedem Schnittpunkt oder Veränderung des Graphen ein Punkt

hinterlegt werden. Ebenso kann definiert werden, ob der Graph zur besseren Übersichtlichkeit mit Farbe gefüllt werden soll oder die Breite der Linien des Graphen. Hier gibt es ebenfalls noch die Option, den Graphen als Balkendiagramm zu visualisieren. Diese Einstellungen werden für alle Graphen innerhalb der Visualisierung durchgeführt. Um dies für jeden einzelnen Graphen anpassen zu können, gibt es den Punkt `Series Overrides`. In diesem können die Einstellungen, welche ebenfalls in `Draw options` definiert werden können, für jeden einzelnen Graphen festgelegt werden. Abschließend gibt es zu diesem Menüpunkt noch den Punkt `Thresholds`. Dies sind Markierungen, innerhalb des Graphens, wenn ein kritischer Wert erreicht worden ist. Diese dienen dazu, auf einen Blick zu erkennen, ob ein Problem der Instanz vorliegt. Im Anhang 13.13 ist ein Bild beigefügt, welches dies einmal verdeutlicht. [MR]

Alert

Im vorletzten Punkt `Alert` können bestimmte Graphen überwacht werden, wodurch bei einem kritischen Wert eine Benachrichtigung versendet wird. Damit diese Funktion verwendet werden kann, muss diese Funktion zum einen in der allgemeinen Konfiguration unter dem Reiter `Alerting` konfiguriert werden und zum anderen muss dies die Datenquelle unterstützen. Die Benachrichtigung hierzu kann anschließend beispielsweise über Email oder den Messenger Telegram oder Threema erfolgen. Innerhalb der `Alert` Konfiguration, können für jeden Graphen individuell die Kriterien definiert werden, bei denen eine Benachrichtigung ausgelöst werden soll. Ein Beispiel hierzu ist dem Anhang 13.14 beigefügt. Die Bedingung muss entweder einen Min- oder Maximalwert eines Graphen beeinhalten, ab wann diese Bedingung erfüllt ist. Diese wird entweder mit einem Standardtext versendet oder kann über den Konfigurationsmenüpunkt `Notifications` individuell angepasst werden. Zudem wird definiert, über welchen Kommunikationsweg die Benachrichtigungen erfolgen sollen. [MR]

Time range

Im letzten Punkt `Time range` kann das Intervall der anzuzeigenden, visualisierten Graphen definiert werden. Innerhalb der Dashboardübersicht in der rechten, oberen Hälfte von Grafana kann die allgemeine zu visualisierende Zeit definiert werden. Diese gilt für alle Graphen innerhalb eines Dashboards. Diese Einstellung kann individuell für jeden einzelnen Graphen unter `Time range` überschrieben werden und gilt dann nur für den aktuell bearbeiteten Graphen. [MR]

7.11 Logstash und collectd

Die Implementierung beider Komponenten gestaltet sich als sehr einfach. Für beide existiert ein fertiges Puppetmodul. Außerdem sind die Konfigurationsoptionen überschaubar. Für Logstash wurde ein eigenes Puppet-Profil geschrieben. Dieses befindet sich unter 13.10. Es installiert Logstash und anschließend das JDBC Plugin für die Kommunikation mit der Datenbank. Hierfür wird noch der Postgres Treiber benötigt, dieser wird von der Postgres Webseite heruntergeladen. Zum Schluss wird die Logstash-Konfiguration aus 13.2 erzeugt und Logstash gestartet. Hierbei stellte sich heraus, dass das Puppet-Modul für Logstash nicht zu 100% auf Archlinux funktioniert. Dies wurde vom Projektteam behoben. Der Fix wurde den Entwicklern in einem Pull Request zugeschickt (vgl. [Meu17b]).

Die Konfiguration von collectd läuft sehr ähnlich ab. Es wurde wieder ein Puppet-Profil erzeugt, dieses befindet sich unter 13.9. Dies installiert collectd und richtet dann das Netzwerk-Plugin ein, um Daten zu Logstash schicken zu können. Im Anschluss werden die verschiedenen Plugins zum Sammeln der Metriken aktiviert. Hierbei wurde festgestellt, dass das Puppet-Modul für collectd keine sinnvollen Standardwerte für das Konfigurieren des collectd Repositorys besitzt. Dies wurde behoben und der Fix wieder den Entwicklern zur Verfügung gestellt (vgl. [Meu17a]).

[TM]

8 Tests und Qualitätssicherung

Der genutzte sowie der selbst entwickelte Code ist äußerst komplex. Es muss von Anfang an sichergestellt werden, dass dieser Code nicht nur funktioniert, sondern auch den gängigen Best Practices der jeweiligen Programmiersprache entspricht. Dadurch wird der Code strukturierter und ist für Außenstehende leichter zu verstehen. Im folgenden Kapitel werden die einzelnen Testverfahren vorgestellt und begründet. [TM]

8.1 Peer Review

Alles an entwickelter Software befindet sich in einem Git Repository. Die Repositories sind unter anderem auf der Webseite `https://github.com/` gespeichert. Im Branch `master` befindet sich ausschließlich getesteter Code. Es wird nie direkt in diesem Branch gearbeitet. Jedes Teammitglied erstellt sich einen `Feature Branch`. Dies ist ein kurzlebiger Branch, welcher auf `master` aufbaut. Im `Feature Branch` wird ein einzelnes Feature von einem einzelnen Teammitglied implementiert. Im Anschluss wird ein Pull Request eröffnet. Dies ist eine Möglichkeit von GitHub, einen Branch mit einem anderen zusammenzuführen. Über die Webseite werden die Unterschiede der beiden Branches gezeigt. Ein weiteres Teammitglied liest diese Änderungen (Review). Man kann nun Korrekturen anfordern oder den Code mergen. Dann wird der `Feature Branch` mit `master` zusammengeführt. Ein Teammitglied darf nie seinen eigenen Code mergen. [TM]

8.2 Continuous Integration

Continuous Integration beschreibt eine Vorgehensweise in der Softwareentwicklung. Hierbei wird für jede Änderung eine zuvor definierte Testmatrix durchlaufen, um sicherzustellen, dass die Software weiterhin intakt ist. Dies setzt voraus, dass Tests vor dem eigentlichen Code entwickelt werden (Test Driven Development) oder parallel zur Entwicklung des Codes. Ersteres stellt sicher, dass ausschließlich das

implementiert wird, das gefordert ist. In den Tests wird definiert, wie eine Software zu funktionieren hat, danach wird so lange am Code entwickelt bis die Tests erfolgreich sind. Wenn man die Tests parallel zum Code entwickelt, ist es möglich, dass der Code mehr als die geforderte Funktionalität aufweist.

Im Rahmen des Projekts werden Tests immer parallel zum Code entwickelt. Test Driven Development erfordert ein enormes Fachwissen über die genutzten Frameworks für Tests. Dieses lässt sich aufgrund der Komplexität nicht während des Projektes aneignen. [TM]

8.2.1 Travis

Travis ist eine Plattform für Continuous Integration. Sie ist kostenlos nutzbar und stellt verschiedene virtuelle Maschinen und Container bereit, in denen Tests ausgeführt werden können. GitHub kann bei neuen Pull Requests eine Benachrichtigung an Travis schicken. Daraufhin startet Travis eine neue VM und klont das entsprechende GitHub Repository und den `Feature Branch`. Danach wird eine definierte Testmatrix ausgeführt. Travis schickt eine Benachrichtigung an GitHub, nachdem die Tests beendet wurden. Somit ist in der Weboberfläche ersichtlich, ob die Tests erfolgreich waren oder nicht.

Dieses Verfahren ermöglicht es, eine reproduzierbare Testumgebung zu nutzen. Diese wird immer auotmatisch gestartet. Es ist nicht mehr notwendig, jeden Test lokal auszuführen. Travis stellt VMs bereit, in denen nur das nötigste installiert ist. Dadurch ist sichergestellt, dass keine Side Effects auftreten. Bei lokalen Tests tritt dies häufig auf. Der Entwickler hat diverse Bibliotheken installiert, welche von der entwickelten Software benötigt werden. Innerhalb der entwickelten Software wird eine Abhängigkeit nicht definiert, es funktioniert aber trotzdem, da die Bibliothek zufällig vorhanden ist. Innerhalb von Travis fällt dies aber auf. Außerdem ist es jedem Teammitglied möglich, die Travis Logs von überall aus zu begutachten. Alternativ müsste jeder Entwickler die Tests selbst lokal ausführen. [TM]

8.3 Linter und Syntax Validator

Linter- und Syntax-Tests werden immer genau dann benötigt, wenn der Quellcode immer komplexer wird. Ein Linter-Test prüft von einem Quellcode die Formatierung und macht diesen anschließend lesbarer, damit auch Laien diesen Quellcode gut verstehen können. Der Sinn hinter einem Linter-Test ist es, das best Practice als auch die Darstellung auf dem Quellcode nach einer hinterlegten Liste von Formatierungen

zu erzwingen. Diese Liste an Formatierungen beeinhaltet jede Programmiersprache und deren Formatierungen, da jede Programmiersprache eigene Formatierungen besitzt. Ein Syntax-Test macht eine statische Code-Analyse und prüft, ob der gesamte Code syntaktisch in Ordnung ist. Dies bedeutet, dass zum Beispiel geprüft wird, ob eine Klammer (öffnen oder schließen) vergessen wurde oder ein Simikolon an der richtigen Stelle ist. [MR]

8.4 Unit-Tests

Für die Tests und Qualitätsicherung wurden unter anderem Unit-Tests verwendet. Unit-Tests sind Software-Tests und werden immer dann verwendet, wenn jeweils die Kernfunktion von einem Quellcode geprüft werden soll. Rspec ist das Framework und stellt eine DSL zur Verfügung, in der Testszenarien definiert werden können. Die Puppet-Module, welche in diesem Projekt verwendet werden, wie beispielsweise Grafana, besitzen schon einige Tests, welche erweitert worden sind. Mit einem Test kann beispielsweise geprüft werden, ob das Profil auf unterschiedlichen Betriebssystemen ausgeführt wird oder ob bestimmte Ressourcen im kompilierten Katalog enthalten sind. Zu Beginn müssen diese vorab mit rspec in dem Katalog kompiliert und definiert werden, damit diese verwendet werden können. Mit folgendem Befehl können die Tests durchgeführt werden:

```
bundle exec rake spec
```

Die vollständigen Tests für das Grafana-Profil befinden sich unter dem Anhang 13.20. Zuerst werden verschiedene Betriebssysteme simuliert, um das Grafana-Profil auf diesen zu testen. Jedes Betriebssystem besitzt betriebssystemspezifische Facts. Diese werden von rspec simuliert und mit Hilfe einer Liste der unterstützten Betriebssysteme auf dem entsprechenenden Profil getestet. Diese Liste an Betriebssystemen wird aus der `metadata.json` ausgelesen und hierfür immer ein Katalog erstellt. Dieser zuvor erstellte Katalog wird anschließend geprüft. Unter anderem kann das rspec prüfen, ob benötigte Ressourcen wie `grafana_dashboard` oder `grafana_datasources` in dem Profil enthalten sind. Diese Tests können für jedes Profil einzeln definiert werden, da jedes Profil unterschiedliche Ressourcen benötigt. Im Anhang befindet sich ein Auszug aus einem bestandenen Test. Es wurde bewusst darauf geachtet, dass jedes Modul nur seine eigenen Ressourcen auf Fehler prüft. Zusätzlich wurde jede einzelne Software von den Entwicklern auf Funktionalität und Sicherheit geprüft.

Die Vorteile von solchen Unit-Tests mit rspec sind Folgende:

- Verschiedene Tests, wie Funktionalität des Profiles, können gleichzeitig geprüft werden
- Direkte Ausgabe mit detaillierter Fehlerbeschreibung
- Tests können für alle Profile gleich definiert werden
- Die Tests können gleichzeitig für unterschiedliche Profile durchgeführt werden
- Das Framework erlaubt es auf einfache Art und Weise, viele verschiedene Betriebssysteme mit einem Test zu testen

[MR]

8.5 Akzeptanz Test

Ein Akzeptanz Test prüft den zuvor erstellten Code in einer eigens dafür gestarteten virtuellen Maschine oder abgesichertem Container mit einem Betriebssystem auf die Funktionalität der einzelnen Module. Hierzu wurden schon bereits vorhandene Test Szenarien von testkitchen und serverspec verwendet und diese auf die im Projekt verwendeten Module angepasst. Der Code wird innerhalb dieser Tests ausgeführt und geprüft, ob die Grundfunktion, sowie die Funktionalität der einzelnen Module und das Puppet Modul idempotent und fehlerfrei läuft. Diese Szenarien prüfen beispielsweise, ob die Erhaltung der Daten, sowie das Speichern dieser Daten in der Datenbank und Visualisierung der Graphen, funktioniert. Die Ausgabe erfolgt hier über einen booleschen Wert, welcher entweder `true` oder `false` sein kann. Anschließend kann der Entwickler diesen Code erneut anpassen und testen. Die vollständigen Tests für das collectd Modul sind dem Anhang 13.7beigefügt.

Wie auch auch bei Unit-Tests aus Abschnitt 8.4 werden Akzeptanztests für verschiedene Betriebsysteme programmiert. Anders als bei Unit-Tests werden diese nicht in einer `metadata.json` hinterlegt, sondern jedes Betriebssystem erhält eine eigene Datei, in der definiert ist, mit welcher virtuellen Maschine oder Container dieser Code ausgeführt werden soll. Im Anhang 13.8 ist eine solche Datei beigefügt. [MR]

8.6 API Test

Während der Entwicklung des API Systems wurde bei der Implementierung von einem neuen Algorithmus immer der zugehörige Test definiert. Dies wurde, wie in Abschnitt 8.4 bereits beschrieben, mit Komponententests durchgeführt. Speziell bei der API wurden die Komponententests in einem parallelen Verzeichnis zu dem

Source Code und in der gleichen Programmiersprache wie die API, also Java, verfasst. Das in dem in Abschnitt 7.9 genannte Entwicklerstudio „Spring Tool Suite“ bietet dabei eine benutzerfreundliche Darstellung und Verwendung. Es stellt den zu einem Algorithmus zugehörigen Komponententest direkt im Programm bereit, sodass jeder Entwickler die zugehörigen Tests sofort erkennen kann und hierzu nicht danach suchen muss.

Für das Ausführen der definierten Tests muss keine eigene Laufzeit Infrastruktur bereitgestellt werden. Echte Komponententests werden normalerweise extrem schnell ausgeführt und steigern durch die Integration in das genannte Entwicklerstudio die Produktivität der API Entwicklung um ein Vielfaches. Durch die geringe Laufzeit konnten die Tests auch beliebig oft, in diesem Fall nach jeder Änderung am Code, ausgeführt werden, ohne das längere Wartezeiten entstehen.

Die von Swagger bereitgestellten Testbibliotheken sind dabei sehr umfangreich und bieten zu jeder API Funktion auch die passenden Testdefinitionen. Die bereitgestellten Bibliotheken sind dabei von Mock-Objekten abgeleitet. Mock-Objekte sind in Testumgebungen Platzhalter für die im echten Code verwendeten Objektdefinitionen. Ein konkretes Beispiel dafür ist die „Bean“ Definition von Swagger. Mit dieser können verschiedene Definitionen von Objekten in eigenen Laufzeitumgebungen einfach simuliert und ausgewählt werden. In dem Projekt war dies zum Beispiel das Nutzen von im Arbeitsspeicher gehaltenen Daten zum Testen während der Entwicklung versus dem Zugriff auf die Produktivdaten der Datenhaltung. Ein weiteres Beispiel, welches ebenfalls in diese Art von Testbibliothek gehört, war das Registrieren von verschiedenen API Profilen, welche alle unterschiedliche Zugriffsrechte besitzen, um sicher zu stellen, dass jedes dieser Profile so funktioniert, wie von dem Entwickler vorgegeben.

Neben den vorgestellten Komponententests kann der Entwickler ebenfalls Integrationstest in der Swagger Testumgebung definieren. Bei diesen werden nicht einzeln definierte Algorithmen auf die Funktionsweise getestet, sondern eine aufeinander abgestimmte Reihe von Einzeltests. Dabei wird geprüft, ob die voneinander abhängigen Komponenten optimal miteinander zusammenarbeiten und vorhanden sind (vgl. [Spr17]).

Bei der Swagger Testbibliothek kann zum Beispiel das korrekte Caching von Daten getestet werden. Das Anfragen und Abholen von Daten bei einem Datenhaltungssystem kann je nach Abfrage sehr lange dauern. Um diese Zeit zu verkürzen, nutzt Swagger die Methoden des Caching, welche mit diesem Test überprüft werden. Ein weiterer Testfall wäre die Prüfung von Abhängigkeiten. Wenn zum Beispiel die API eine eigene Konfigurations-Datei besitzt, kann damit geprüft werden, ob diese vollständig und ohne Fehleinträge ist. Jedoch kann der Integrationstest auch für

komplexere Inhalte verwendet werden. In diesem Projekt ist es die Prüfung der korrekten Transaktion von Daten mit der Nutzung von Backup-Transaktionen. Diese helfen vor allem, wenn die API Daten in dem Datenhaltungssystem modifiziert, da nach der Modifizierung die Daten wieder auf den vorherigen Stand zurückgesetzt werden. Diese Tests sparen nicht nur Administrations-Zeit, sondern geben dem Entwickler auch einen umfangreichen Bericht zu den von dem Programm getätigten Inhalten, um zu prüfen, ob dies auch vollständig das gewollte Ergebnis ist. [NL]

8.7 ETL Test

Für den in Abschnitt 7.8.1 beschriebenen Extraktions-, Transformations- und Ladeprozess von der Datenhaltung wird neben den in Abschnitt 8.4 erläuterteten Komponententests und in Abschnitt 8.6 beschriebenen Integrationstest zusätzlich noch ein Verhaltenstest durchgeführt. Bei diesem wird geprüft, wie das System reagiert, wenn eine vorher nicht definierte Situation eintritt. Dies könnte zum Beispiel das Erhalten von anderen Daten sein, welche keine definierten Metadaten-Einträge besitzen. [NL]

9 Ausblick

Im Projekt wurde eine solide und modulare Architektur erstellt. Sie erfüllt alle Anforderungen, bietet aber auch noch diverse Möglichkeiten für weitere Optimierungen oder Erweiterungen. [TM]

9.1 Alerting

Es ist nicht Bestandteil des Projektes, bei bestimmten Schwellwerten Benachrichtigungen zu verschicken. Das komplette Setup ist darauf ausgelegt, reaktiv und nicht aktiv zu arbeiten. Grafana bietet die Möglichkeit des Alertings. Dabei können Schwellwerte für bestimmte Metriken definiert werden. Sobald diese erreicht werden, wird eine Benachrichtigung verschickt. Dies kann über verschiedenste Kommunikationswege erfolgen. Denkbar ist es, Kunden zu informieren, wenn Ihr virtueller Server am oberen Ressourcenlimit läuft. Es könnte auch der Vertrieb informiert werden, welcher dann dem Kunden eine größere Instanz verkaufen könnte. [TM]

9.2 Quality of Service

Logstash arbeitet im `Batch-Mode`. Dies bedeutet, das es mehrere Werte pro Prozess cached und diese dann gebündelt in einem SQL-Statement zur Datenbank schickt. Der Standardwert liegt bei 125 Werten und ist konfigurierbar. Es ist in bestimmten Konstellationen möglich, das Logstash mehrere Minuten lang Werte zwischenspeichert, bevor sie zur Datenbank geschickt werden. Dies passiert, wenn besonders viele Logstash Prozesse gestartet worden sind und wenig Daten von den Hypervisors kommen (zum Beispiel weil gerade viel VMs deaktiviert sind oder Wartungsarbeiten an Teilen der Infrastruktur durchgeführt werden). Wenn ein Kunde live seine Auslastung visualisieren möchte, dauert dies eventuell zu lange. Zur Optimierung gibt es zwei Möglichkeiten. Zum einen kann der Zwischenspeicher verringert werden. Dies erhöht allerdings die Last auf der Datenbank, da dies zu wesentlich mehr SQL-Statements führt. Zum anderen kann mit Logstash QoS gemacht werden.

Hierbei wird auf die eingehenden Daten ein Grok Filter gelegt. Dieser erlaubt es, Daten zu filtern und in Kategorien einzuordnen. Für jede Kategorie kann die Größe des Zwischenspeichers definiert werden. Diese Filter kann man auch dynamisch aktivieren. Sobald ein Kunde eine Live-Visualisierung im Grafana startet, kann ein Grok Filter aktiviert werden. Dieser filtert Daten dieses Kunden heraus, kategorisiert diese in eine QoS Queue mit einem Zwischenspeicher von 0. Somit wird jeder Datensatz von diesem Kunden direkt an die Datenbank weitergeleitet. [TM]

9.3 Cache Invalidation

Neben dem eingesetzten `Batch-Mode` gibt es noch eine weitere Arbeitsmethode in Logstash. Hierbei wird mit `Cache Invalidation` gearbeitet. Es wird ein Schwellwert gesetzt, der das maximale Alter des Caches beschreibt. Sobald die Daten den Schwellwert erreicht haben (zum Beispiel 5 sekunden), werden sie zur Datenbank geschickt, auch wenn das Limit des Zwischenspeichers noch nicht erreicht ist. [TM]

10 Fazit

In vielen Monaten Arbeit wurde ein neues Softwareprojekt geschaffen, welches eine sehr gute Alternative zu Telemetry aus dem OpenStack-Projekt darstellt. Die erarbeitete Lösung erfüllt alle definierten Anforderungen aus Abschnitt 2.4. Nicht nur können alle Userstorys über Grafana visualisiert werden, auch die API stellt einen passenden Placement Algorithmus zur Verfügung und bietet somit mehr Features als Telemetry. Das Softwareprojekt ist äußerst flexibel gehalten und kann in beliebige Umgebungen mit Linux Hypervisors integiert werden. Aufgrund des modularen Konzepts und den offenen Schnittstellen gibt es auch mehrere Möglichkeiten für zukünftige Erweiterungen oder Spezialisierungen für bestimmte Umgebungen, wie im Abschnitt 9 dargestellt.

Das Projektteam hat nicht nur viel über Software Engineering, sondern auch über agiles Projektmanagement und kollaboratives Arbeiten gelernt. Währen der Entwicklung ergaben sich diverse Probleme. Durch das konsequente Arbeiten mit Prototypen konnten Probleme im ersten Architektur-Prototyp schnell erkannt werden. Gemeinsam wurde stets eine Lösung gefunden. Diverse kleinere Bugs in der benutzten Software wurden vom Team behoben und mit zusätzlichen Unit Tests versehen. Alle Änderungen wurden an die jeweiligen Entwickler geschickt.

Bei der Durchführung des Projektes wurde bei jedem Projektmitglied der praktische Wissensschatz erweitert. Zum einem wurden neue Wissensinhalte zu Techniken und Arbeitsmethoden erschlossen, zum anderen Vorteile weiter fortgebildet und Nachteile erneut auf die Probe gestellt. Entsprechend wurde zum Beispiel die Belastbarkeit jedes Einzelnen erforscht und die Selbsteinschätzung somit weiter fortgebildet oder sogar korrigiert. [TM, NL, MR]

11 Glossar

Active Directory Ist ein von Microsoft entwickelter Server-Dienst, welcher auf Windows Server Maschinen installiert werden kann, um in einem Unternehmensverzeichnis über das Protokoll LDAP eine Abfrage oder Änderung durchführen zu können. 54

Active-Passive Prinip Eine Möglichkeit des Serverbetriebs. Ein aktiver Server erhält alle Anfragen von Clients, ein oder mehrere passive Server sind verfügbar. Der aktive Server kann deaktiviert werden und ein passiver wird dafür aktiv geschaltet. 38

Agents Ist meist ein Programm, welches bestimmte Abläufe selbständig ohne weitere Eingriffe des Benutzers überwachen kann. Sobald hier ein bestimmter Wert errreicht wurde, kann auf verschiedene Aktionen zurückgegriffen werden. 61

API Application Programming Interface, ermöglicht eine standardisierte Anbindung von externen Softwaresystemen über öffentliche Methoden einer Bibliothek [DJ06]. 7, 10, 15, 20, 21, 39, 45–49, 52, 54–58, 60, 67, 72–76, 80, 81, 84, 87, 93–95, 98

Backlog Begriff aus der Softwareentwicklung und aus dem Projektmanagement. Bezeichnet die Menge an gefundenen Fehler, die noch nicht behoben worden sind, sowie die Menge der unbearbeiteten Änderungswünsche (Featurerequests). 18, 27

Bidirektional Eine Punkt-zu-Punkt-Verbindung, in der Daten in beide Richtungen fließen. 19

Carbon Carbon ist eine Software aus dem Graphite-Projekt. Sie empfängt Metriken aus dem Netzwerk und schreibt diese in eine Datenbank. 27, 56

CLI Command Line Interface, eine Schnittstelle zu einem Service, welche über eine Kommandozeile genutzt wird. 72

Client-Server Begriff aus der Computer Netzwerkstruktur. Es beschreibt die Verbindungsart zwischen zwei oder mehreren Computern in einem Netzwerk.

Dabei kommuniziert ein oder mehrere Teilnehmer mit einem zentral definierten Server. In einem Client-Server-Modell sind die Teilnehmer immer auf den Server angewiesen, da er benötigte Dienste im Netzwerk anbietet. 75

Cloud Eine Plattform, die nach dem Client-Server-Modell arbeitet. Eine oder mehrere bestimmte Dienstleistungen werden hier zur Verfügung gestellt (z.b. Mailhosting). Die Verteilung von Ressourcen erfolgt dynamisch. 8–10, 13, 103

Commit Aus der Softwarentwicklung: Ein Commit bündelt eine oder mehrere Änderungen in einem Repository. Aus der Datenbanktechnik: Ein Commit beendet eine Transaktion. 38

CSV Comma Seperated Values, Kommaseparierte Daten. Ursprünglich wurde nur das Komma als Delimiter genutzt, mittlerweile sind verschiedene Trennzeichen zulässig. Eine Kopfzeile, welche die Daten beschreibt, ist optional [Sha05]. 25, 28

Dashboard Ein Dashboard ist eine Oberfläche auf einer Webseite, um Graphen oder Metriken einem User zur Verfügung stellen zu können. Dem Anwender werden diese Graphen oder Metriken visualisiert dargestellt. 49, 55–59, 74, 82–86, 88

Datenbanktreiber Wie auch bei Treibern, wird ein Datenbanktreiber benötigt, um eine Verbindung mit einer Datenbank aufbauen zu können. Zusätzlich wird hier das Schema der Inhalte dieser Datenbank überprüft und für die Verwendung aufbereitet. 45, 54

DBMS Datenbankmanagementsystem, dieses System ist die Schnittstelle zwischen gespeicherten Daten und den Anwendern. Es stellt Operationen wie Datenzugriff und Datenmanipulation bereit. Außerdem verwaltet es das Datenbankschema. Das DBMS ermöglicht parallele Zugriffe, bietet Datenintegrität sowie administrativen Zugriff. 36, 38, 39, 69–72, 102, 103, 105

Deklarative Programmiersprache Hierbei wird in der Programmiersprache das zu lösende Problem beschrieben. Der Interpreter oder Übersetzer der Sprache analysiert dies und erarbeitet eigenständig einen Lösungsweg. Die bekanntesten Sprachen sind hier SQL und puppet. Ein alternativer Programmierstil ist imperativ. 73

DSL Domain Specific Language, eine Programmiersprache für einen sehr spezifischen Einsatzzweck. 73, 74, 92

EVA Eingabe, Verarbeitung, Ausgabe. Ein Prinzip aus der Datenverarbeitung. 28

Fact Ein Fact ist ein Key-Value-Wert, welcher mit dem Programm Facter aus dem laufenden System ermittelt wurde. Die Facter Notation erfolgt in JSON. Ein Beispiel findet sich unter 13.3. 75, 92

Garbage Collector Oft als GC abgekürzt, übersetzt: Müllabfuhr/Müllman. Es beschreibt die automatische Speicherwaltung eines Programms. Ein Programm allokiert Speicher dynamisch. Der GC prüft periodisch den belegten Speicher und gibt nicht mehr benötigte Teile wieder frei. 36

Git Freie Software zur Versionsverwaltung von Dateien. 34, 38, 90, 101

GitHub GitHub ist ein Filehoster, welcher für die Veröffentlichung und Sicherung von sogenannten Software-Repositorys verwendet werden kann. GitHub basiert auf Git und Entwickler können hier entweder alleine oder im Team an einem Softwareprojekt zusammenarbeiten. Jeder Entwickler kann sein eigenes Repository erstellen und dieses verändern, bearbeiten oder weiterentwickeln. 54, 59, 90, 91

Gnocchi Datenbank aus dem OpenStack-Projekt zum Speichern von Metriken. 13

Grok Ein Grok ist ein Filter zum extrahieren und strukturieren von Daten. Dieser besteht aus einem oder mehreren regulären Ausdrücken. Logstash nutzt diese zum filtern. 97

HDD Hard Disk Drive, mechanisches Speichermedium, bei dem die Daten auf Magnetscheiben abgespeichert werden. 64

Hochverfügbarkeit Ein IT-System gilt als verfügbar, wenn es seine definierte Aufgabe abarbeiten kann. Als hochverfügbar gilt es, wenn es diese Aufgabe trotz Ausfall einer oder mehrerer Komponenten noch erfüllen kann. 38

HTTP HyperText Transfer Protocol ist ein Protokoll, welches für die Übertragung von Inhalten wie z.B. Webseiteninhalte verwendet wird. 39, 49, 101

HTTPS Hypertext Transfer Protocol Secure. Basiert auf HTTP mit zusätzlicher Transportverschlüsselung. 21, 49, 52, 75, 87, 102

Hypervisor Eine Software welche Hardware partitioniert und virtuellen Maschinen zur Verfügung stellt. 25, 27, 30, 67, 96, 98

Imperative Programmiersprache Die Sprache besteht aus einer Reihe von spezifischen Anweisungen, welche der Reihe nach abgearbeitet werden. Die meisten Programmiersprachen arbeiten nach diesem Paradigma. 100

Interrupt Ein Interrupt ist die vorübergehende Unterbrechung eines bestimmten

Prozesses, damit andere Prozesse, welche wichtiger sind, Vorrang bekommen. Dieser Interrupt wird durch die Hardware ausgelöst. 14, 28, 65, 105

iowait Ein iowait beschreibt, ob ein Prozessor mit der Verarbeitung von Aufgaben warten muss, weil der Datenspeicher (HDDs, SSDs) beim Lesen oder Schreiben nicht hinterherkommt. 14, 65

IRC Channel Internet Relay Chat ist genau wie HTTPS oder LDAP ein Netzwerkprotokoll. Es ermöglicht die Kommunikation über einen Chat. Hier hat der Anwender die Möglichkeit, sowohl privat als auch öffentlich mit anderen Mitgliedern zu schreiben. Um den Chat verwenden zu können, wird eine Client-Software, wie zum Beispiel weechat, benötigt. 58

JDBC Java Database Connectivity, generische Schnittstelle der JVM zur Interaktion mit einem DBMS. 68–72, 89

Jepsen Jepsen ist eine in Clojure geschriebene Bibliothek mit der verteilte Datenbanken auf Persistenz und Zuverlässigkeit getestet werden können [Kin13]. 12, 29, 35, 36

JSON JavaScript Object Notation, ein Datenformat mit dem Ziel, von Menschen und Maschinen verarbeitet werden zu können. Standardisiert in [Bra14]. 12, 20, 28, 34, 78, 86, 87, 101

JVM Java Virtual Machine, eine Laufzeitumgebung für Java Bytecode. 68–70, 72

Kibana Webinterface zum Visualisieren von Logs aus einer Elasticsearch-Datenbank. 28

LDAP Lightweight Directory Access Protocol, ist ein Netzwerkprotokoll, welches zur Abfrage und Änderung von Informationen aus einem Unternehmensverzeichnis verwendet wird. LDAP besitzt keine Transportverschlüsselung, diese wurde bei LDAPS nachgerüstet. 52, 55, 57, 58, 85, 99, 102

Lucene Lucene ist eine Bibliothek der Apache Software Foundation zur Volltextsuche. Sie ist in Java geschrieben und erzeugt Indizes, welche im Anschluss durchsucht werden können. Lucene implementiert mehrere Suchalgorithmen. Die Indexgröße beträgt 20-30% der Größe der Nutzdaten [Fou16a]. 34

Maintainer zu Deutsch: Erhalter, Verantwortlicher. Im Kontext der Softwareentwicklung eine Rolle, die alle Rechte in einem Repository besitzt. Sie entscheidet, welche Änderungen von externen Personen in das Projekt aufgenommen werden. Oftmals leiten Maintainer auch ein Softwareprojekt. Projekte können mehrere Maintainer haben. 26

Middleware Eine Softwarekomponente, welche Daten zwischen zwei oder mehreren verschiedenen Schnittstellen oder Diensten transportiert. Oftmals erfolgt auch eine Transformation der Daten zwischen zwei verschiedenen Protokollen. 68, 71

MRI Referenzimplementierung eines Ruby Interpreters in der Programmiersprache C. Die Abkürzung steht für „Matz's Ruby Interpreter", Yukihiro „Matz" Matsumoto ist der Erfinder von Ruby. 69

Namespace Zu deutsch: Namensraum. Variablen sind nur innerhalb eines Namespaces gültig. Der gleiche Variablenname kann in einem anderen Namespace wiederverwendet werden. 73

ODBC Open Database Connectivity, Standardisierte Schnittstelle für ein DBMS via SQL. 70

Partitionen Eine Partition in einem Datenbankmanagementsystem definiert einen Umgang mit großen Datenmengen, indem eine einzelne Tabelle in Tochtertabellen aufgeteilt wird. Dabei werden die Datenmengen mit Hife von Vererbungen auf mehrere abhängige Tabellen geteilt. 37, 79

Peer-to-Peer Begriff aus der Computernetzwerkstruktur. Es beschreibt die Verbindungsart zwischen zwei oder mehreren Computern in einem Netzwerk. Dabei kommunizieren zwei oder mehr Teilnehmer über eine direkte Verbindung miteinander. In einem Peer-to-Peer Netzwerk sind alle Teilnehmer gleichgestellt, die Alternative ist das Client-Server-Modell. 36

Pgpool-II Eine Erweiterung für Postgres. Sie ermöglicht das Wiederverwenden von Datenbankverbindungen, das hochverfügbare Replizieren der Daten sowie das Verteilen von Leseanfragen [Gro16a]. 38

Postgres-XC Eine Erweiterung für Postgres. Sie ermöglicht den vertikal skalierenden Cluster. Hierbei skaliert die Leistung sowohl beim Lese- als auch beim Schreibzugriff. Je mehr Nodes man in den Cluster stellt, desto höher wird die Schreibleistung [Gro16e]. 38

Private Cloud Siehe auch Cloud, Hier wird jedem Nutzer eine dedizierte Infrastruktur bereitgestellt. 12

Public Cloud Siehe auch Cloud, auf dieser Plattform teilen sich mehrere Benutzer die gleiche physische Infrastruktur. 12

Quality of Service Begriff aus der Datenverarbeitung und der Netzwerktechnik. Daten werden in verschiedene Klassen kategorisiert und mit unterschiedlicher

Priorität behandelt. Besonders bekannt ist dies bei der Behandlung von Telefoniedaten im Internet. Diese werden immer mit höchster Priorität transportiert. 96

Repositorium Eine Bibliothek über Schemadaten von Datenbanktabellen und den notwendigen Informationen zu den Datenbereinigungs- und Transformationsregeln in einem Datenhaltungssystem. 76

Repository Verzeichnis zum Speichern von Objekten. Im Git Kontext: Alle logisch zusammengehörenden Dateien eines Projekts. 34, 38, 82, 89–91, 100–102

RESTful REST steht für Representational State Transfer. Es ist ein Architekturprinzip für Schnittstellen auf Basis von HTTP. RESTful bedeutet, dass eine Applikation das REST-Prinzip implementiert [Fie00]. 34–36, 75

Ringtopologie In einer Ringtopologie ist jede Station mit ihrern direkten Nachbarstation verbunden. Entsprechend empfängt eine Station im Ring die Daten und reicht sie an den Nachbarn weiter. Der Nachrichtenumlauf ist dabei vorgegeben. 37

Semantic Versioning Ein anerkanntes Verfahren zum Definieren von Versionsnummern. Es nutzt das Schema Major.Minor.Patch. Nur Releases ab 1.0.0 sind für den produktiven Betrieb ausgelegt [Pre13]. 29

serverspec Hiermit werden RSpec Tests geschrieben, um einen Server auf eine Misskonfiguration zu prüfen. Hierzu werden keine zusätzlichen Agents auf den einzelnen Server benötigt. Tests können über SSH oder WinRM abgesetzt werden. 93

Shared-Nothing-Architektur Eine mögliche Architektur für verteilte Systeme. Hier besitzt jeder Teilnehmer im Cluster alle notwendigen Daten, um den Ausfall eines anderen Teilnehmers zu kompensieren. Das System ist somit weiterhin komplett erreichbar, allerdings mit eingeschränkter Leistung. 37, 39

Side Effect In Deutsch: Seiteneffekt. Der Ablauf eines Programmes ändert sich durch unerwarteten externen Einfluss. Dies tritt sehr häufig bei der Validierung von Nutzerdaten auf. Es wird ein bestimmer Datentyp erwartet aber ein anderer übergeben. 91

singlestat Ein singlestat ist die Zusammenfassung einer Statistik aus mehreren Werten. Sie zeigt hier wahlweise den Minimal- oder Maximalwert. Diese können in einem Panel visuell dargestellt werden. 83

Skalierung Beschreibt das Erhöhen der Ressourcen in einem komplexen Softwaresetup. Bei horizontaler Skalierung wird die Leistung eines einzelnen Servers

durch ein Hardwareupgrade erhöht, bei vertikaler Skalierung wird das Softwaresetup auf mehrere Server verteilt. In großen Setups wird die vertikale Vorgehensweise bevorzugt, da sie mehr Leistung bieten kann. 35, 38

Soft-IRQ Ein Soft-IRQ ist ähnlich wie ein Interrupt für die Unterbrechung eines bestimmten Prozess zuständig, jedoch wird dieser durch einen Software-Interrupt ausgelöst. 14, 65

SQL Structured Query Language, SQL ist ein Medium, um mit einem DBMS zu kommunizieren [SE07]. 29, 34, 36, 38, 41–43, 68, 70, 71, 87, 96, 100, 103, 105

SSD Solid State Disk, nichtflüchtiges Speichermedium welches schnellere Lese- und Schreibzugriffe unterstützt als HDDs. 51, 64

Statement Ein Statement ist eine Abfolge von Befehlen für eine SQL Abfrage der Datenbank. Mit dieser können Werte abgerufen, verändert oder gelöscht werden. 54, 55, 57, 96

Streaming Replication Hierbei schreibt ein Server seine Änderungen an einer Datenbank in eine Logdatei. Die Änderungen an der Datei werden kontinuierlich an alle passiven Server übertragen, diese wenden die Änderungen ebenfalls an. 38

Templates Vorlage für Dokumente, die mit Inhalt gefüllt und auch nach Belieben angepasst werden können. Dies kann zum Beispiel die Planung eines Graphen sein oder einer Weboberfläche. 55

testkitchen Testkitchen ist ein Tool, womit der zu testende Code auf mehreren unterschiedlichen Plattformen in einem Container getestet werden kann. Hier können Tests, welche mit RSpec oder Serverspec geschrieben worden sind ausgeführt und getestet werden. 93

Transaktion Begriff aus der Datenbanktechnik, eine Transaktion bündelt ein oder mehrere Datenbankabfragen. 95, 100

Treiber Ein Treiber definiert die Schnittstelle zwischen einer Software und einer Hardware. Damit eine gewisse Software auf eine Hardware zugreifen kann, wird ein Treiber verwendet, damit hier die volle Unterstützung zwischen Hard- und Software gegeben ist. 15, 20, 21, 39, 41, 54, 69–72, 89, 100

Unidirektional Eine Punkt-zu-Punkt-Verbindung, in der Daten nur in eine Richtung fließen. 19

XML Extensible Markup Language, eine erweiterbare Auszeichnungssprache für strukturierte Daten [Bra+08]. 20, 28

12 Literatur

[16a] *Dokumentenorientierte Datenbank*. Technische Hochschule Köln, Campus Gummersbach. Nov. 2016. URL: http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.DokumentenorientierteDatenbank.

[16b] *Modern monitoring & analytics*. Datadog, Inc. Dez. 2016. URL: https://datadoghq.com.

[Bec+01] Kent Beck u. a. „Manifesto for agile software development“. In: (2001). URL: http://nlp.chonbuk.ac.kr/SE/ch05.pdf.

[Bel+15] T Bell u. a. „Scaling the CERN OpenStack cloud“. In: *Journal of Physics: Conference Series* 664.2 (2015), S. 022003. URL: http://stacks.iop.org/1742-6596/664/i=2/a=022003.

[Bra+08] Tim Bray u. a. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Nov. 2008. URL: https://www.w3.org/TR/2008/REC-xml-20081126/.

[Bra14] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. http://www.rfc-editor.org/rfc/rfc7159.txt. RFC Editor, März 2014. URL: http://www.rfc-editor.org/rfc/rfc7159.txt.

[BV10] Elasticsearch BV. *First Elasticsearch Release v0.4.0*. Feb. 2010. URL: https://github.com/elastic/elasticsearch/releases?after=v0.11.0.

[BV16a] Elasticsearch BV. *Elasticsearch Reference Guide*. Basic Concepts. Nov. 2016. URL: https://www.elastic.co/guide/en/elasticsearch/reference/5.x/_basic_concepts.html.

[BV16b] Elasticsearch BV. *Elasticsearch Resiliency Status*. Nov. 2016. URL: https://www.elastic.co/guide/en/elasticsearch/resiliency/current/index.html.

[BV16c] Elasticsearch BV. *Logstash - Centralize, Transform and Stash Your Data*. Dez. 2016. URL: https://www.elastic.co/products/logstash.

[BV17] Elasticsearch BV. *RabbitMQ Plugin for Logstash*. Jan. 2017. URL: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-rabbitmq.html.

[Dav16] Chris Davis. *Graphite Documentation*. The Graphite Project. Dez. 2016. URL: http://graphite.readthedocs.io/en/latest/.

[DJ06] Danny Dig und Ralph Johnson. „How Do APIs Evolve&Quest; A Story of Refactoring: Research Articles“. In: *J. Softw. Maint. Evol.* 18.2 (März 2006), S. 83–107. ISSN: 1532-060X. DOI: 10.1002/smr.v18:2. URL: http://dx.doi.org/10.1002/smr.v18:2.

[Fie00] R.T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000. URL: https://books.google.de/books?id=1xN9NwAACAAJ.

[For16] Apache Forrest. *Welcome to Apache Hadoop!* Apache Forrest. Dez. 2016. URL: http://hadoop.apache.org/.

[Fou16a] Apache Software Foundation. *Lucene Feature*. Lucene offers powerful features through a simple API. Nov. 2016. URL: http://lucene.apache.org/core/features.html.

[Fou16b] OpenStack Foundation. *Introduction to Openstack Telemetry*. OpenStack Foundation. Nov. 2016. URL: https://wiki.openstack.org/wiki/Telemetry.

[Fou16c] OpenStack Foundation. *What is OpenStack?* OpenStack Foundation. Nov. 2016. URL: https://www.openstack.org/software/.

[Fre16] Inc. Free Software Foundation. *Coreutils - GNU core utilities*. An Introduction to Coreutils. Apr. 2016. URL: https://www.gnu.org/software/coreutils/coreutils.html.

[God16a] Sebastien Godard. *Introduction to sysstat software collection*. Dez. 2016. URL: http://sebastien.godard.pagesperso-orange.fr/download.html.

[God16b] Sebastien Godard. *Introduction to sysstat software collection*. Dez. 2016. URL: http://sebastien.godard.pagesperso-orange.fr/features.html#main_features.

[gro15] collectd working group. *The Virt Plugin*. Mai 2015. URL: https://collectd.org/wiki/index.php?title=Plugin:virt&oldid=4417.

[Gro16a] Pgpool-II Development Group. *What is Pgpool-II?* Nov. 2016. URL: http://www.pgpool.net/mediawiki/index.php/Main_Page#What_is_Pgpool-II.3F.

[Gro16b] PostgreSQL Global Development Group. *Postgres JDBC Driver*. Dez. 2016. URL: https://jdbc.postgresql.org/download.html.

[Gro16c] PostgreSQL Global Development Group. *PostgreSQL License*. Nov. 2016. URL: https://www.postgresql.org/about/licence/.

[Gro16d] PostgreSQL Global Development Group. *The PostgreSQL Wiki*. Nov. 2016. URL: https://wiki.postgresql.org/wiki/Main_Page.

[Gro16e] Postgres-XC Development Group. *Introduction to Postgres-XC*. Nov. 2016. URL: https://wiki.postgresql.org/wiki/Postgres-XC.

[gro16a] collectd working group. *List of collectd Plugins*. Jan. 2016. URL: https://collectd.org/wiki/index.php?title=Table_of_Plugins&oldid=4464.

[gro16b] procps working group. *procps*. Juli 2016. URL: https://gitlab.com/procps-ng/procps/blob/ecff2213bd480c87e209147ee8779ac6baec20ae/README.md.

[gro17] diamond working group. *Introduction to python-diamond*. Jan. 2017. URL: https://github.com/python-diamond/Diamond.

[Haa16] Florian Haas. *Fragile development*. Why Scrum sucks, and what you should be doing instead. Free und Open Source Software Conference (FrOSCon) e.V. Aug. 2016. URL: https://media.ccc.de/v/froscon2016-1722-fragile_development.

[Inc17] Puppet Inc. *Resource Type Reference*. Jan. 2017. URL: https://docs.puppet.com/puppet/latest/type.html.

[Kin13] Kyle Kingsbury. *Jepsen*. On the perils of network partitions. Mai 2013. URL: https://aphyr.com/posts/281-jepsen-on-the-perils-of-network-partitions.

[Kin15a] Kyle Kingsbury. *Jepsen on Elasticsearch*. Apr. 2015. URL: https://aphyr.com/tags/elasticsearch.

[Kin15b] Kyle Kingsbury. *Jepsen on Elasticsearch*. Apr. 2015. URL: https://aphyr.com/posts/323-jepsen-elasticsearch-1-5-0.

[Kin15c] Kyle Kingsbury. *Jepsen on Elasticsearch*. Mai 2015. URL: https://github.com/jepsen-io/jepsen/blob/ecab97547123a0c88bb39ddd3ba3db873dccf251/elasticsearch/src/elasticsearch/core.clj#L414-L431.

[Kin15d] Kyle Kingsbury. *Jepsen on Elasticsearch*. Mai 2015. URL: https://github.com/jepsen-io/jepsen/blob/ecab97547123a0c88bb39ddd3ba3db873dccf251/elasticsearch/src/elasticsearch/core.clj#L433-L451.

[Kin16] Kyle Kingsbury. *Jepsen Tests on MongoDB*. Nov. 2016. URL: https://aphyr.com/tags/MongoDB.

[Kin17] Kyle Kingsbury. *Introduction to the concepts of Riemann*. Jan. 2017. URL: http://riemann.io/concepts.html.

[KK08] U. Kaiser und C. Kecher. *C/C++: Von den Grundlagen zur professionellen Programmierung*. Galileo Computing. Galileo Press, 2008. ISBN: 9783898428392. URL: https://books.google.de/books?id=ge92PQAACAAJ.

[Kou16] Anmol Koul. *Trysts with Technology*. Dez. 2016. URL: https://anmolkoul.wordpress.com.

[Lan16] Gerlof Langeveld. *atop*. Mai 2016. URL: http://www.atoptool.nl/allnews.php.

[Lee15] William Leemans. *The Virt Plugin*. Feb. 2015. URL: https://share.zabbix.com/virtualization/kvm/kvm-monitoring.

[LLC16] Zabbix LLC. *The Virt Plugin*. Feb. 2016. URL: https://www.zabbix.com/documentation/3.2/manual/introduction/overview.

[Mar17] Marcellii. *grafana dashboard doesn't check if grafana is running*. März 2017. URL: https://github.com/voxpupuli/puppet-grafana/issues/35.

[Meu17a] Tim Meusel. *Archlinux Implementation in collectd*. März 2017. URL: https://github.com/voxpupuli/puppet-collectd/pull/658.

[Meu17b] Tim Meusel. *Archlinux Implementation in Logstash*. März 2017. URL: https://github.com/elastic/puppet-logstash/pull/334.

[Mon16a] Inc. MongoDB. *MongoDB Architecture*. Nov. 2016. URL: https://www.mongodb.com/mongodb-architecture.

[Mon16b] Inc. MongoDB. *What is MongoDB?* How does MongoDB work? Nov. 2016. URL: https://www.mongodb.com/what-is-mongodb.

[Öde16] Torkel Ödegaard. *Grafana Documentation Site*. Torkel Ödegaard & Raintank Inc. Dez. 2016. URL: http://docs.grafana.org.

[Pa16] Collection for Postgres developers und administrators. *The PostgreSQL Tutorial*. Nov. 2016. URL: http://www.postgresqltutorial.com/.

[Pre13] Tom Preston-Werner. *Semantic Versioning 2.0.0*. Juni 2013. URL: http://semver.org/spec/v2.0.0.html.

[SE07] S. Sumathi und S. Esakkirajan. *Fundamentals of Relational Database Management Systems*. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2007. ISBN: 9783540483977. URL: https://books.google.de/books?id=RjnNA0GW0wsC.

[Sha05] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180. http://www.rfc-editor.org/rfc/rfc4180.txt. RFC Editor, Okt. 2005. URL: http://www.rfc-editor.org/rfc/rfc4180.txt.

[SIA16] Zabbix SIA. *The Enterprise-class Monitoring Solution for Everyone*. Zabbix LLC. Dez. 2016. URL: https://zabbix.com.

[sof16] solidIT consulting & software development gmbh. *DB-Engines Ranking of Relational DBMS*. Nov. 2016. URL: http://db-engines.com/en/ranking/relational+dbms.

[Sou16] Karl Southern. *JDBC Plugin for Logstash*. Dez. 2016. URL: https://github.com/theangryangel/logstash-output-jdbc/tree/v5.1.0.

[Spr17] Spring. *Introduction to Spring Testing*. Apr. 2017. URL: http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/testing-introduction.html.

[SR86] Michael Stonebraker und Lawrence A. Rowe. *THE DESIGN OF POSTGRES*. 1986. URL: http://db.cs.berkeley.edu/papers/ERL-M85-95.pdf.

[Tea16a] JavaTPoint Development Team. *Differences between JDBC types*. Dez. 2016. URL: http://www.javatpoint.com/jdbc-driver.

[Tea16b] JRuby Development Team. *jruby*. Dez. 2016. URL: http://jruby.org/.

[Tsa16] Costa Tsaousis. *Grafana Documentation Site*. Netdata. Dez. 2016. URL: https://my-netdata.io.

13 Anhang

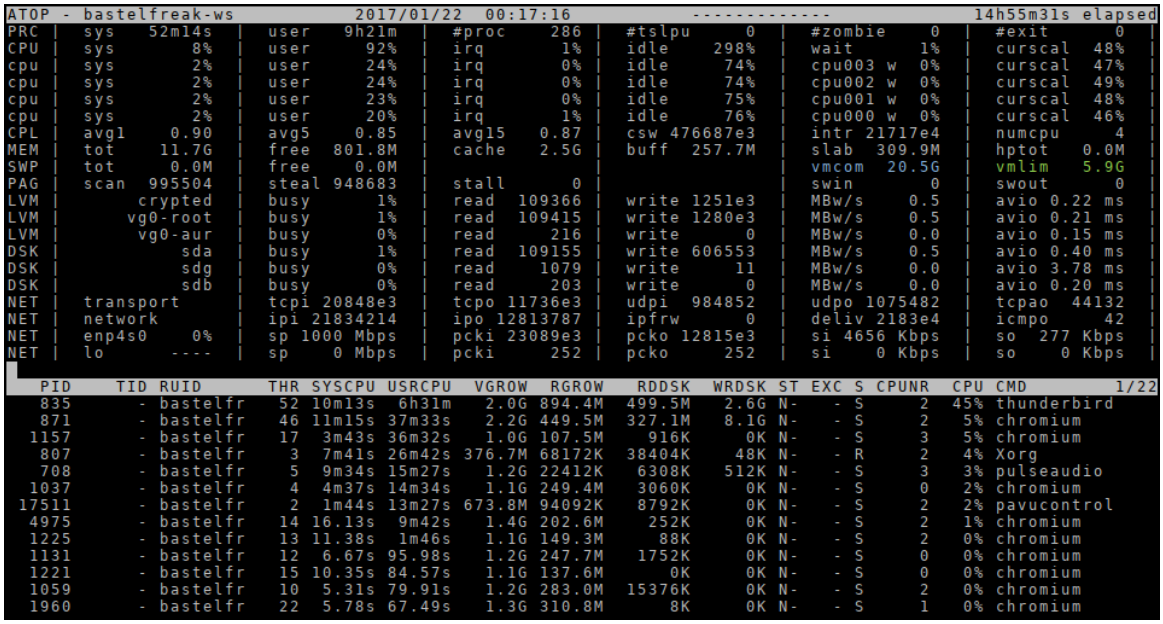

Abbildung 13.1: atop mit geringer Systemlast

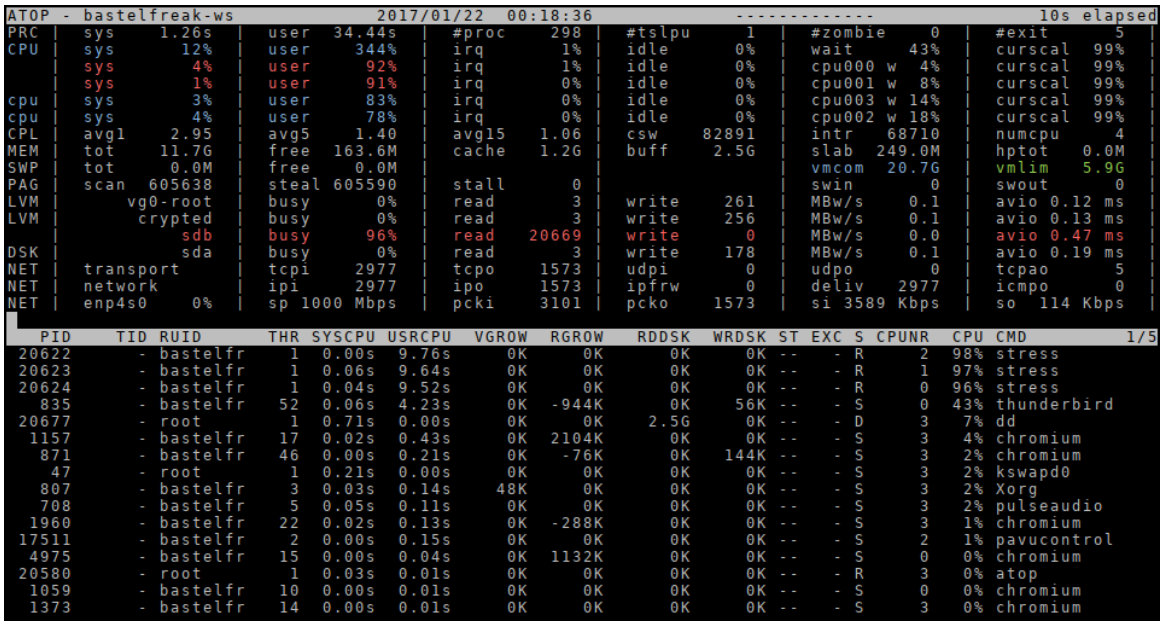

Abbildung 13.2: atop mit hoher CPU/Netzwerk Last

Abbildung 13.3: Offene PRs und Issues im Diamond Projekt am 22.01.2017

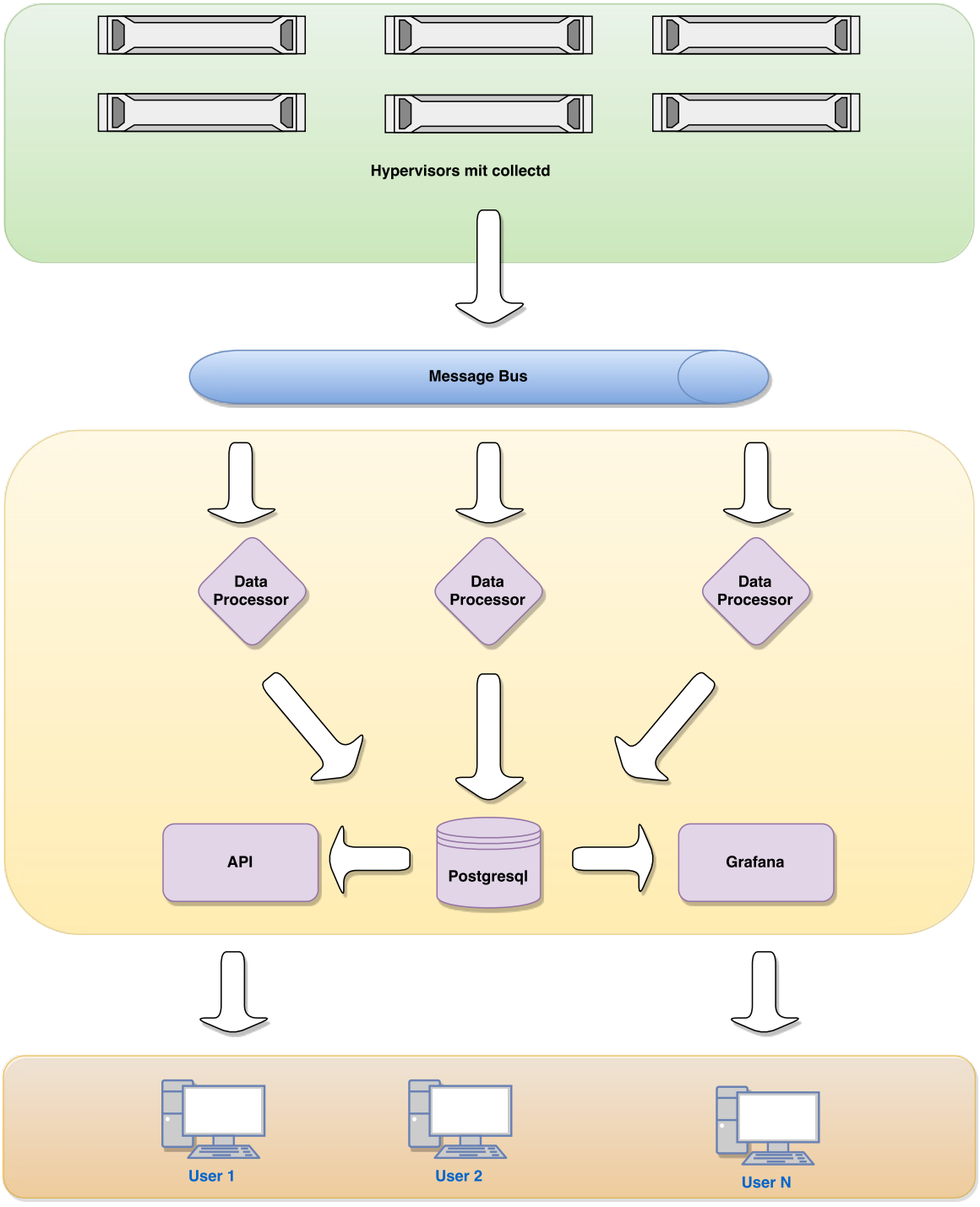

Abbildung 13.4: Architekturdraft Version 1

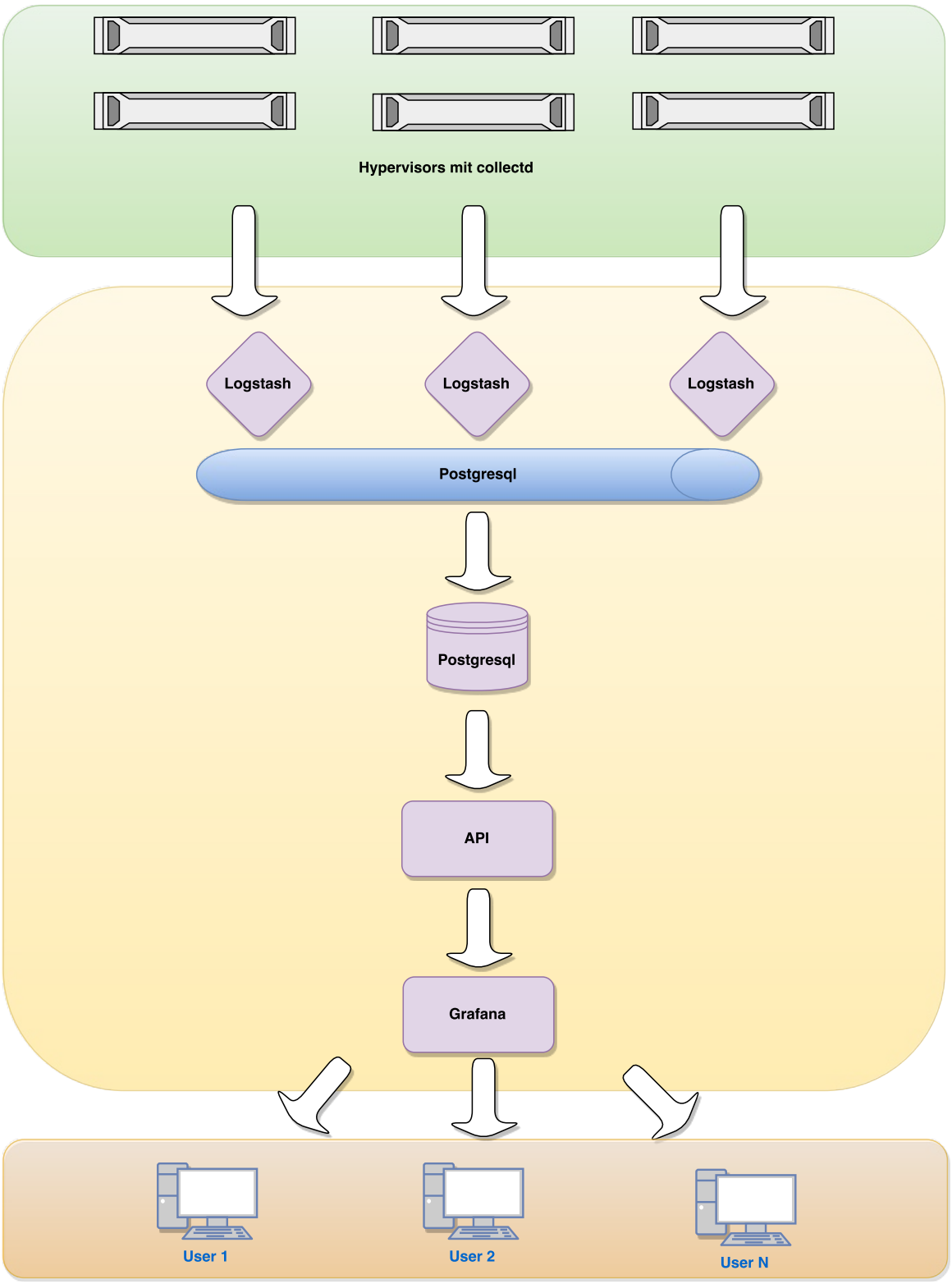

Abbildung 13.5: Architekturdraft Version 2

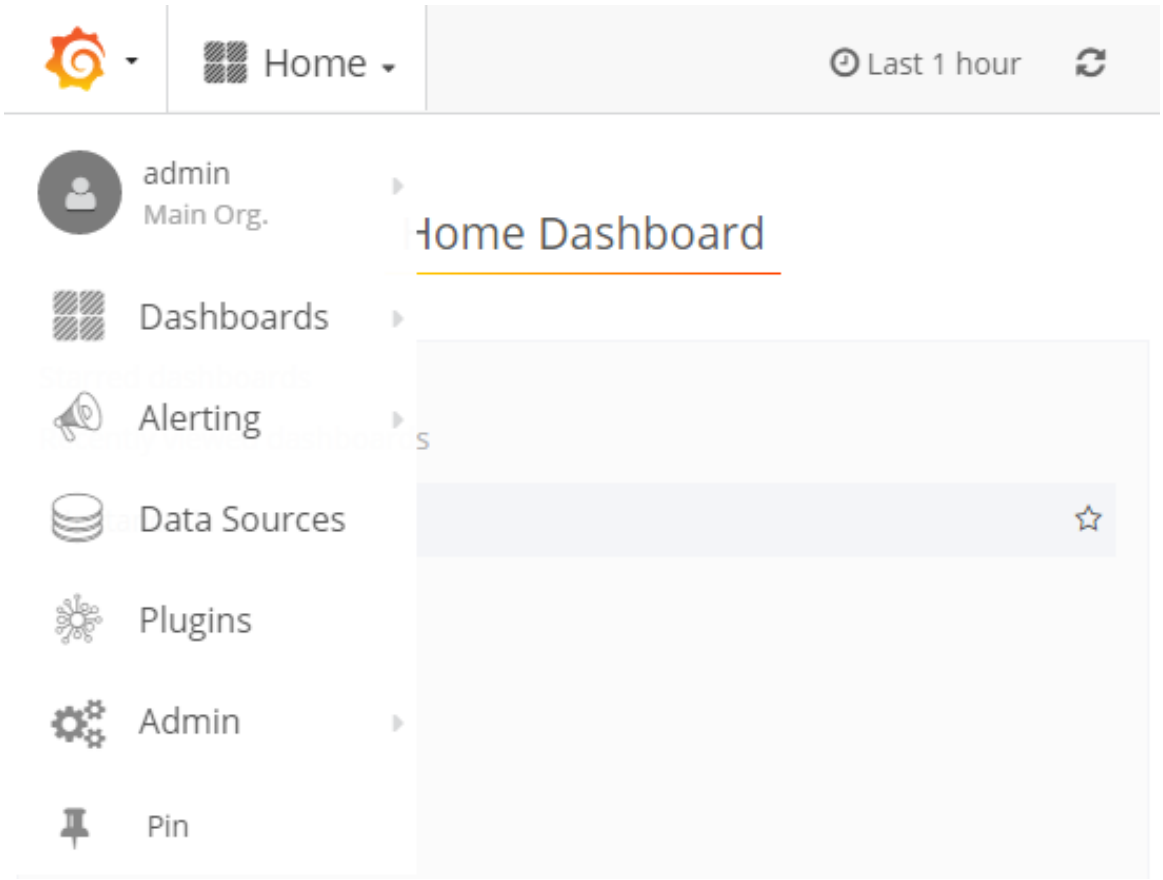

Abbildung 13.6: Grafana Dashboard

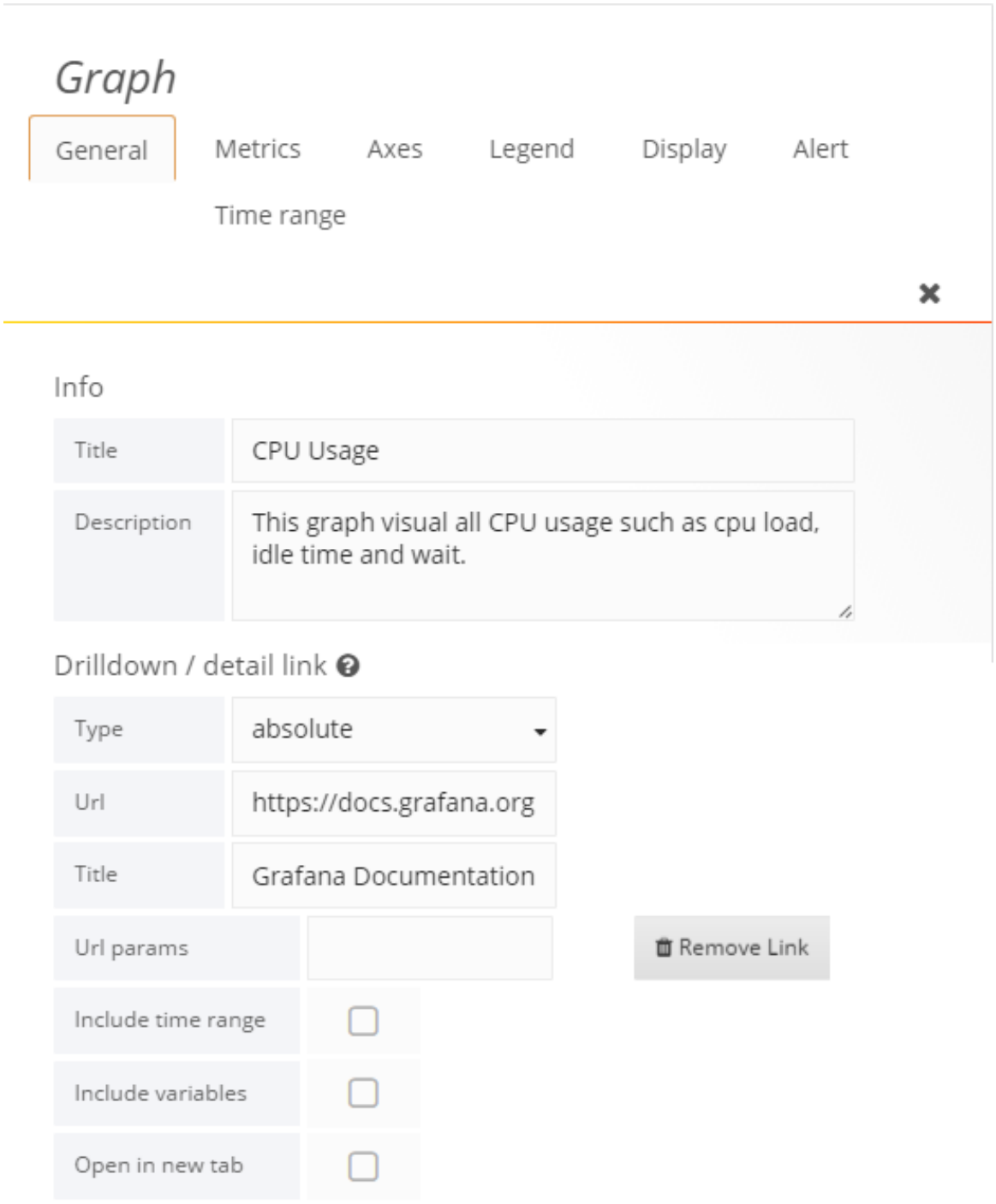

Abbildung 13.7: Grafana Graph Menüpunkt General

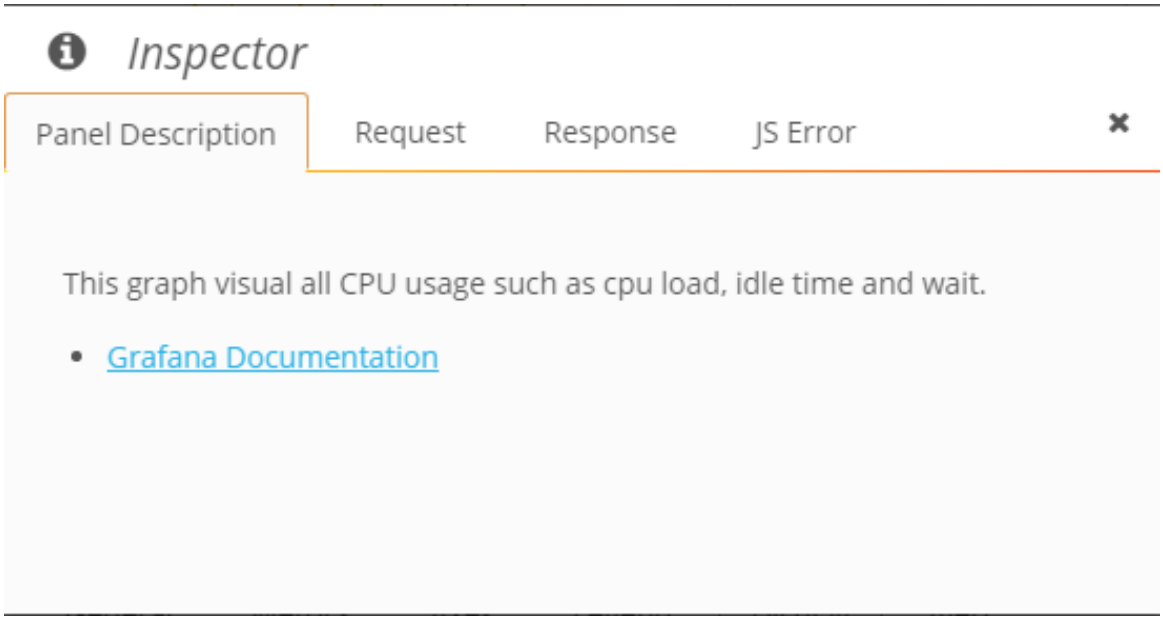

Abbildung 13.8: Grafana Graph Beschreibung

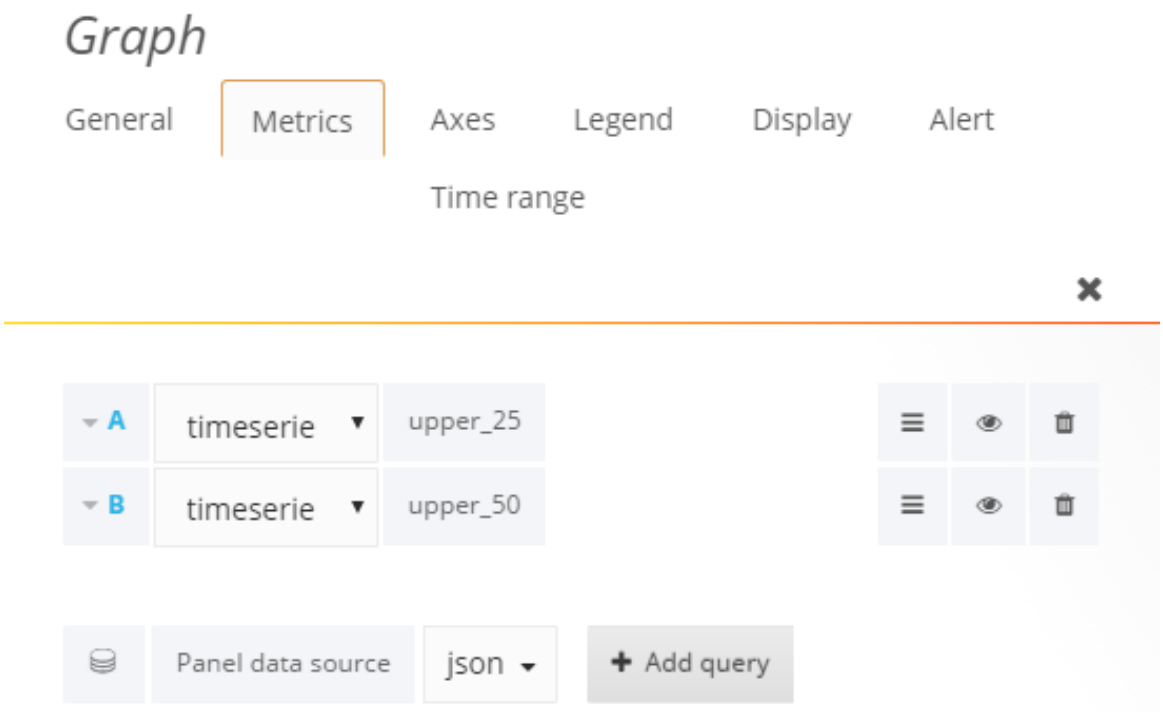

Abbildung 13.9: Grafana Graph Metriks

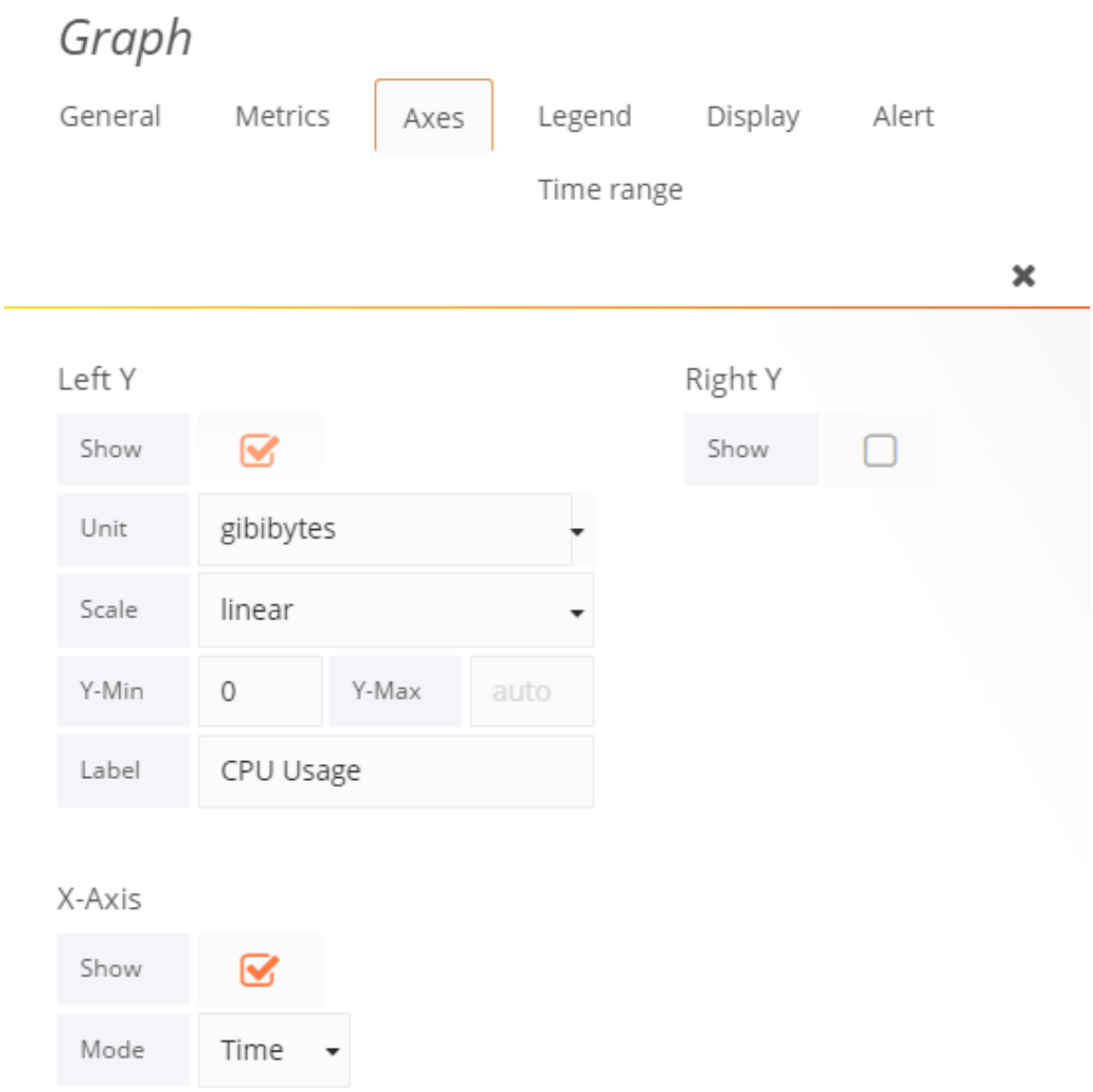

Abbildung 13.10: Grafana Graph Axen

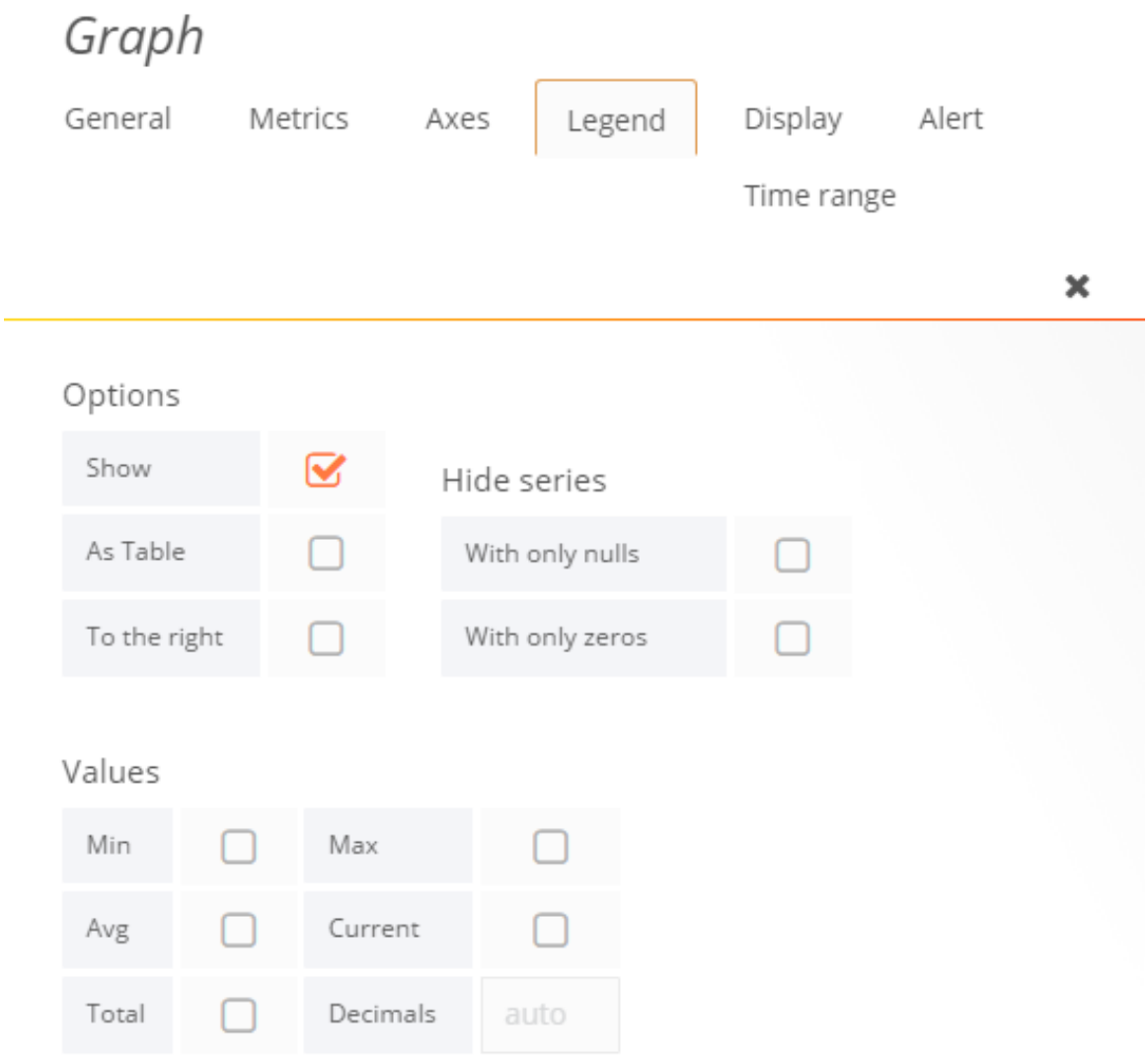

Abbildung 13.11: Grafana Graph Legende

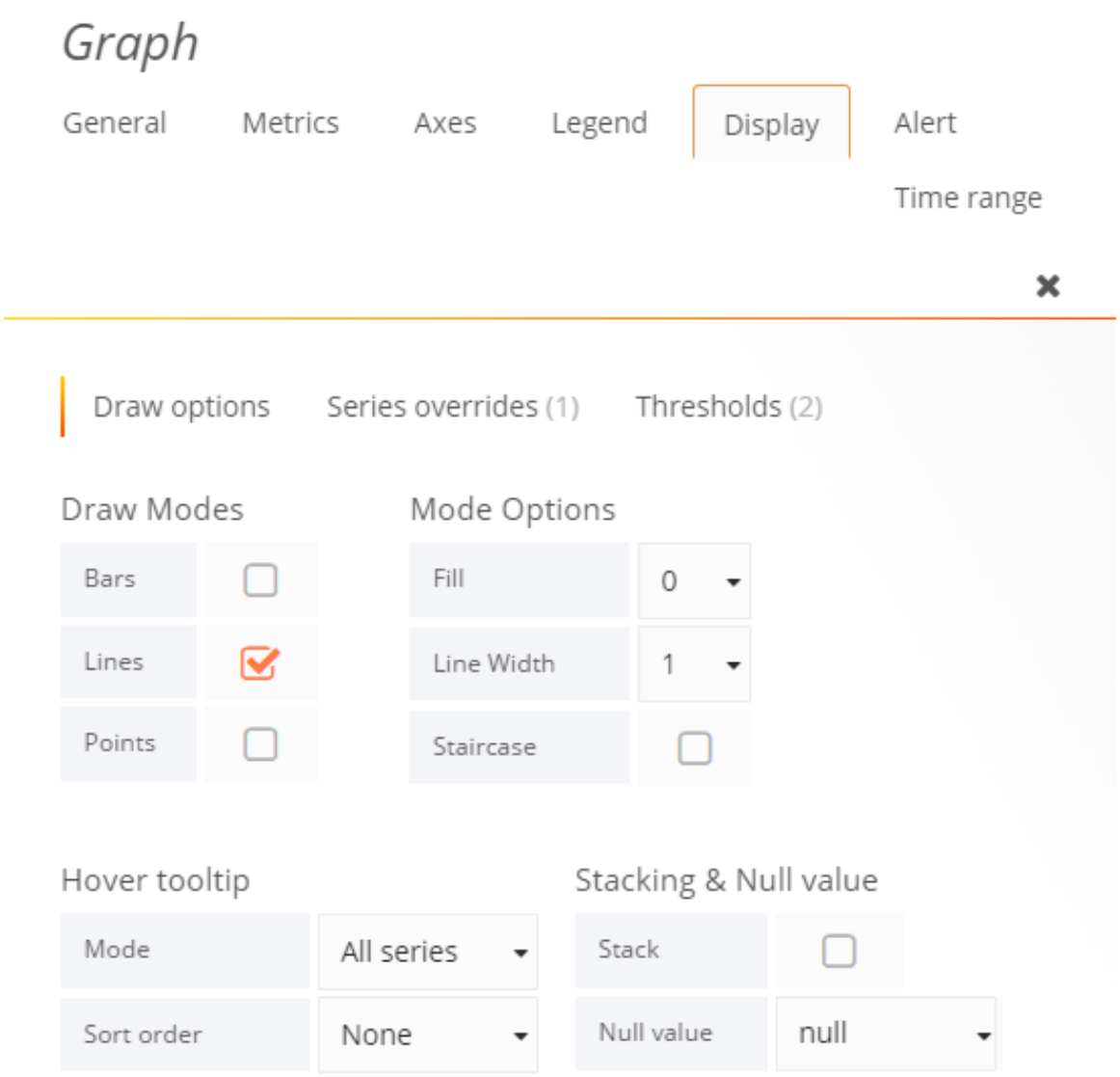

Abbildung 13.12: Grafana Graph Display

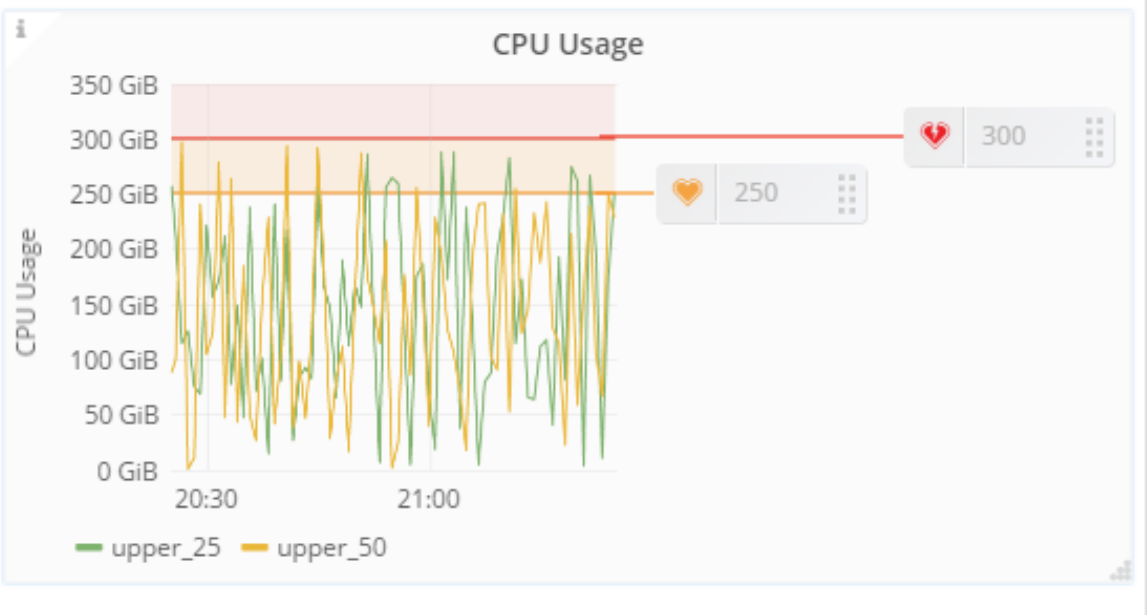

Abbildung 13.13: Grafana Graph Schwellenwert Beispiel

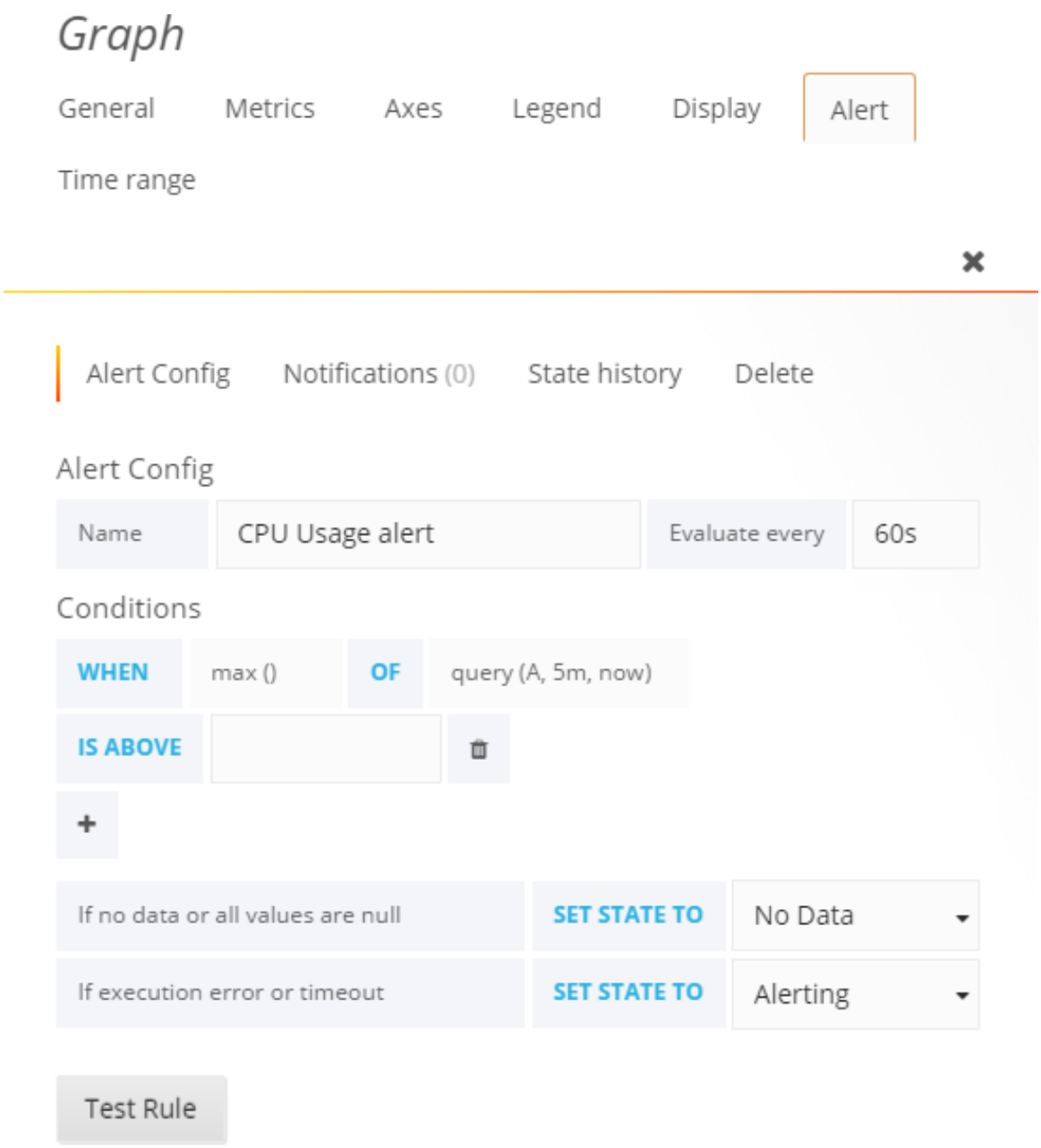

Abbildung 13.14: Grafana Graph Benachrichtigung

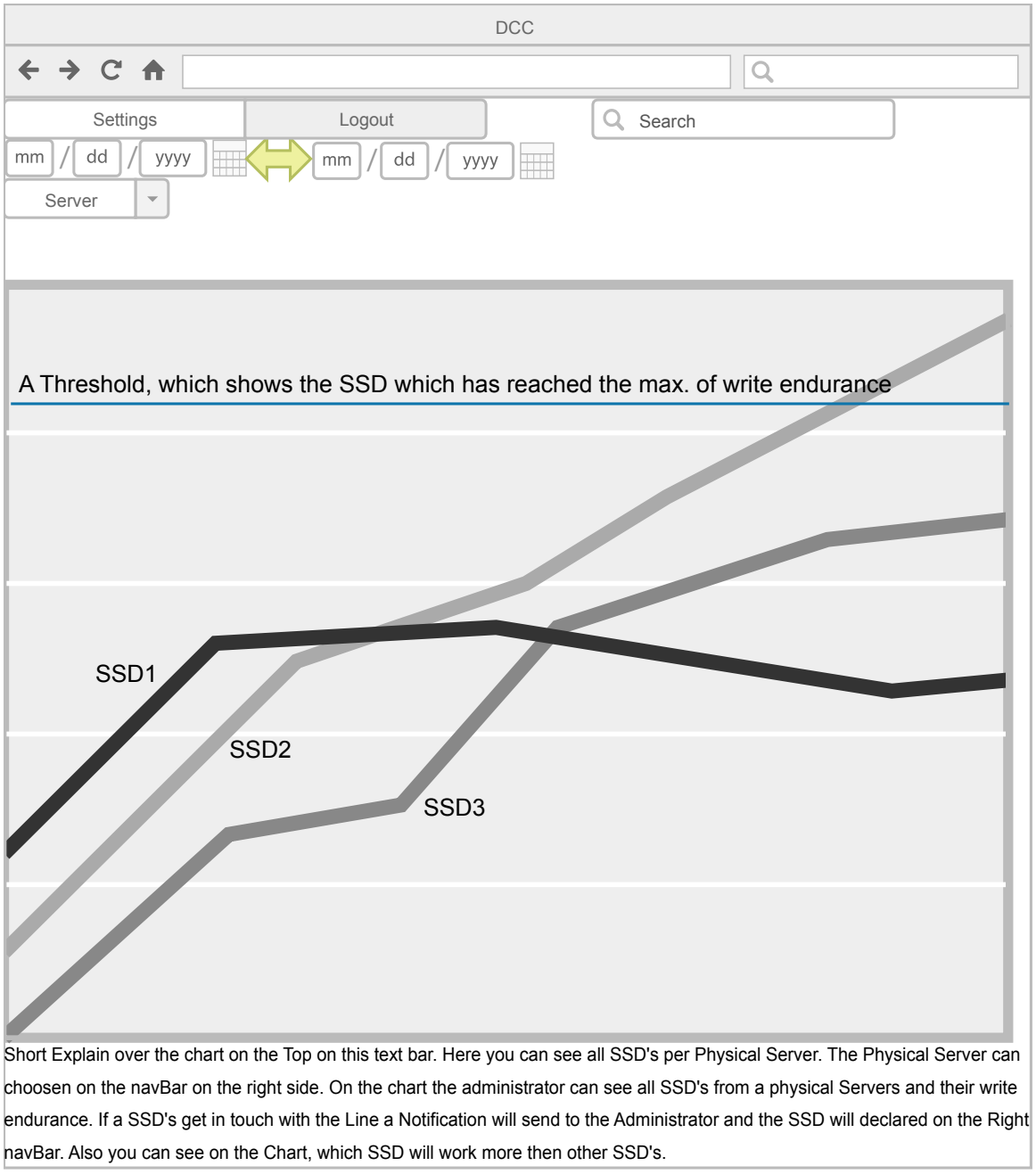

Abbildung 13.15: Wireframe für SSD Userstory

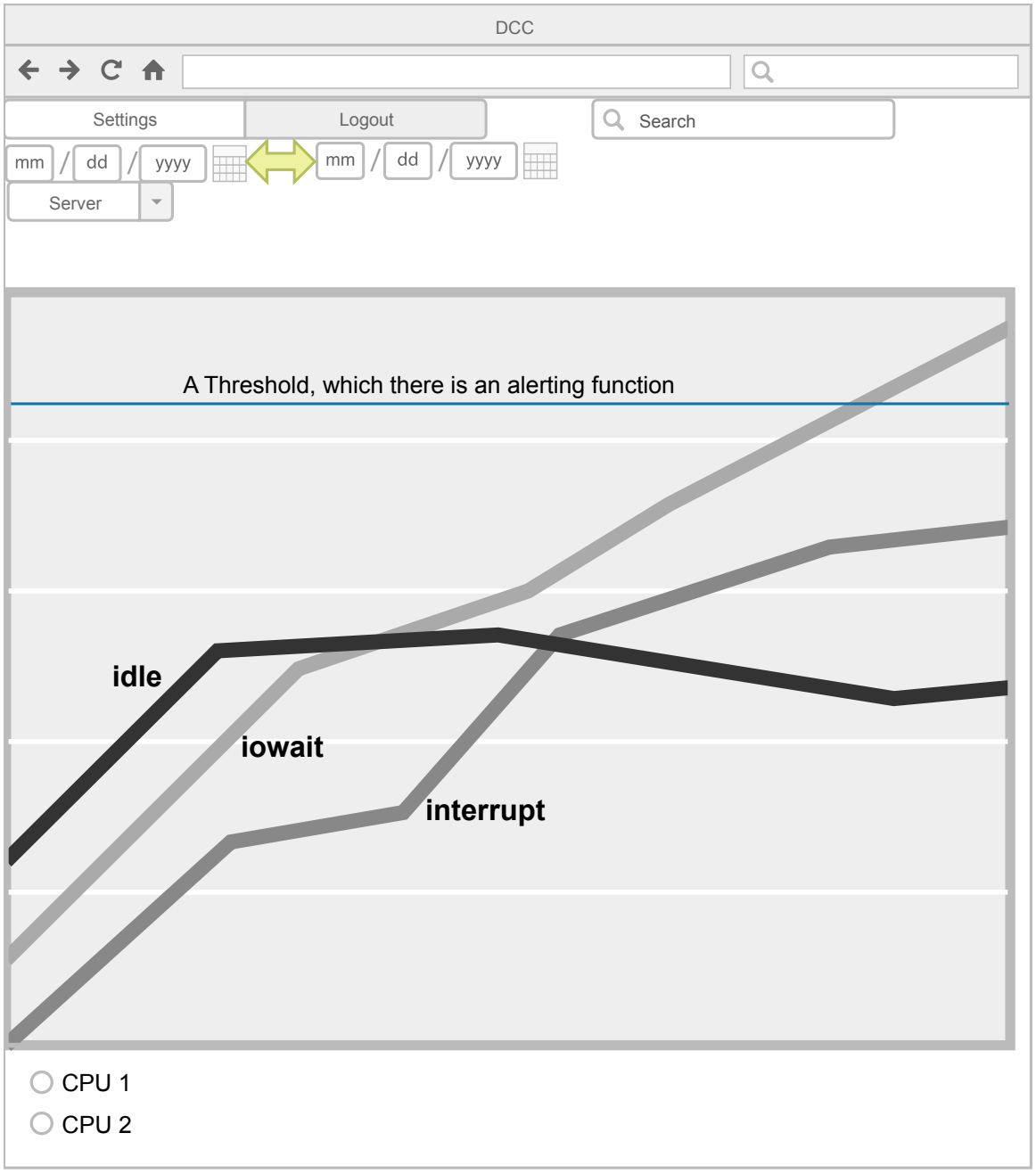

Abbildung 13.16: Wireframe für CPU Userstory

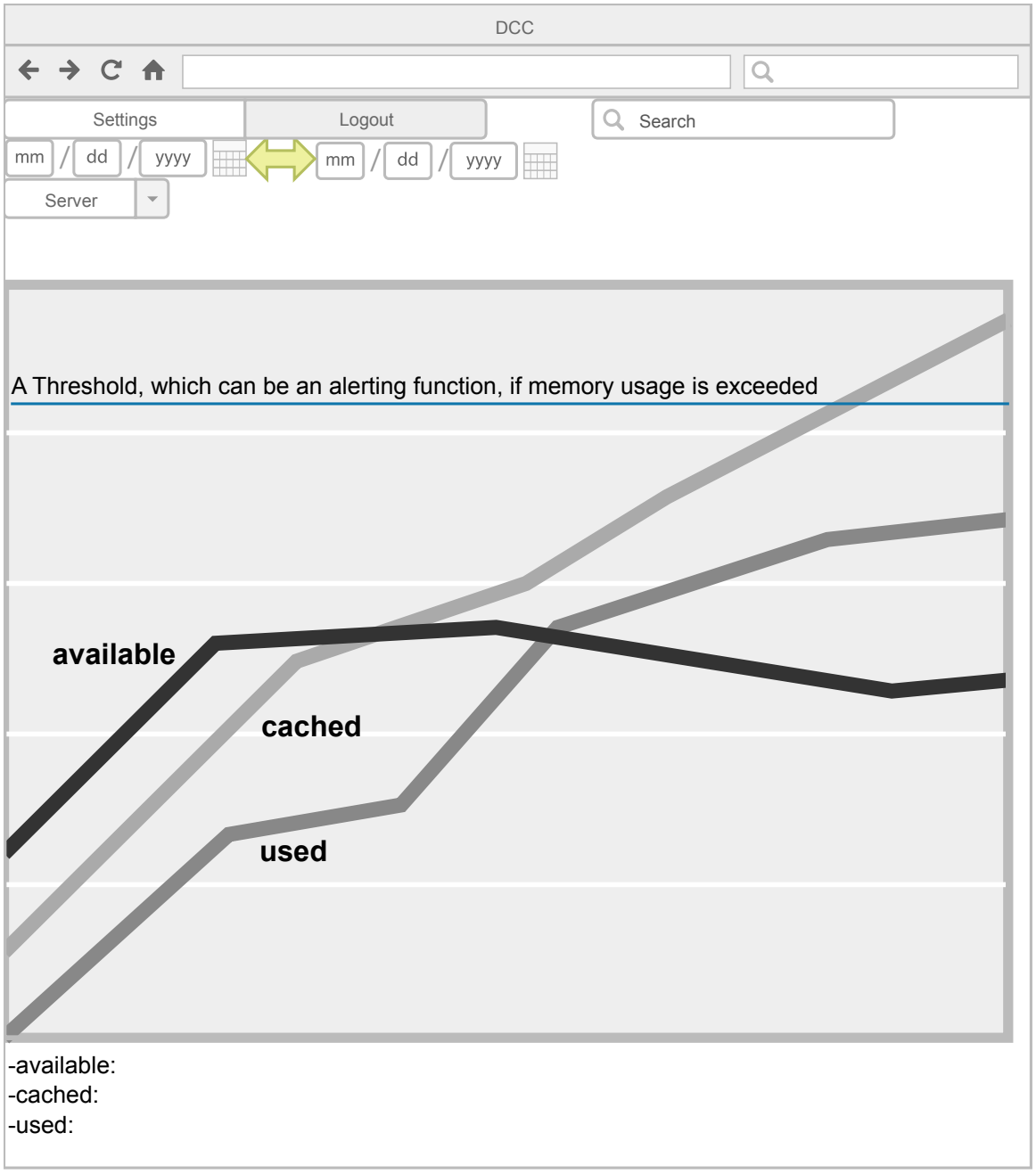

Abbildung 13.17: Wireframe für Memory Userstory

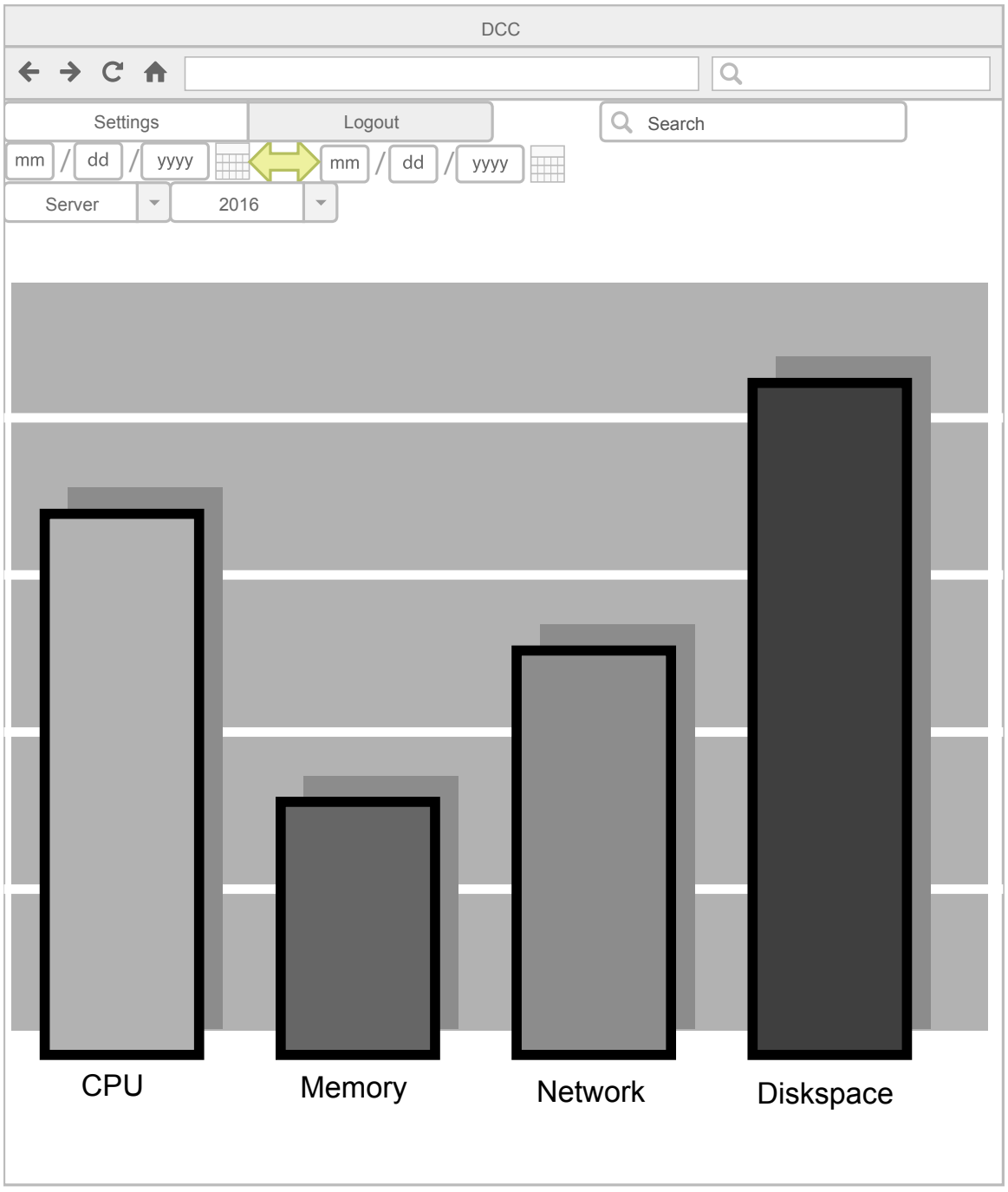

Abbildung 13.18: Wireframe für Zeitdefinierte Analyse Userstory

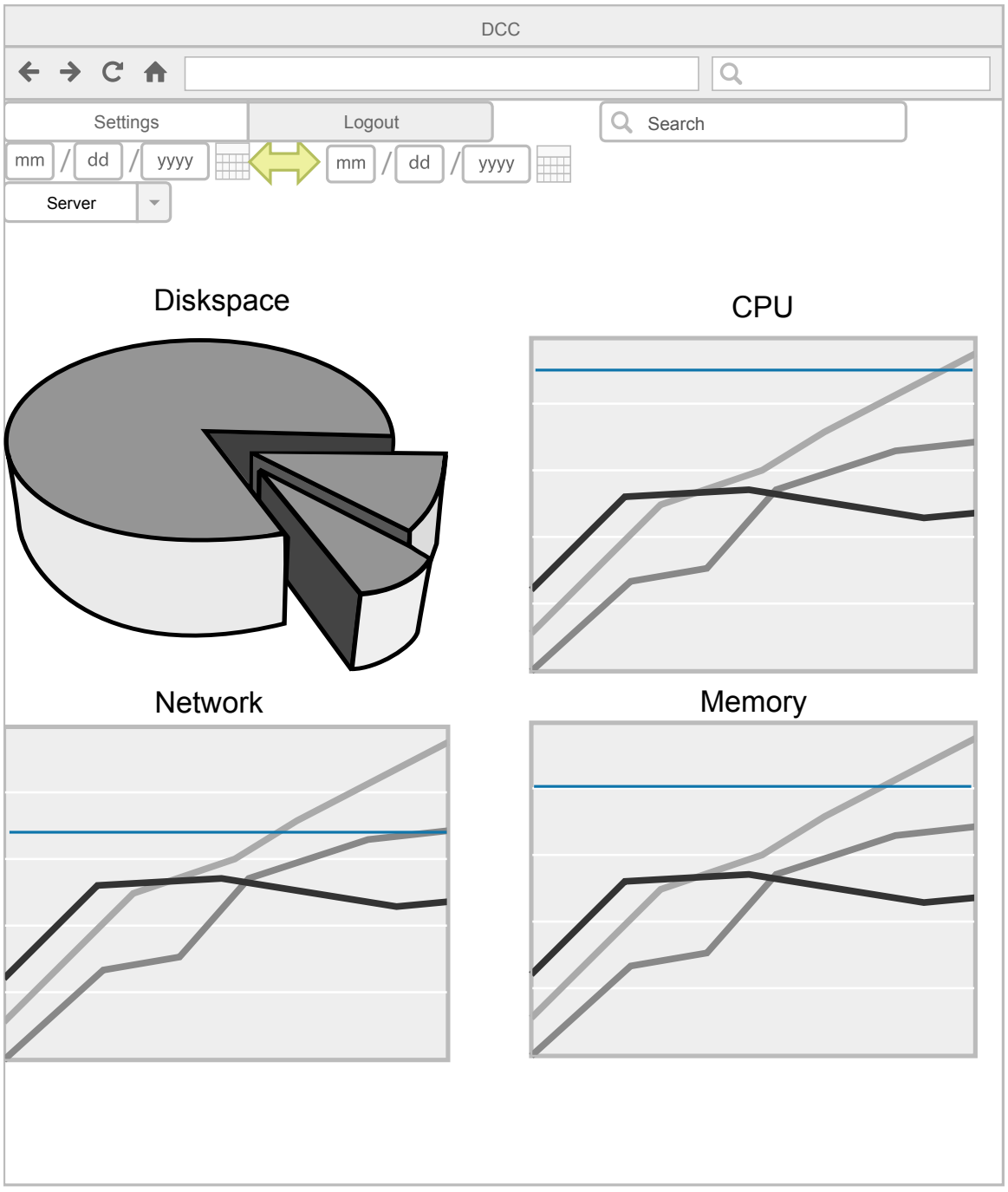

Abbildung 13.19: Wireframe für Weboberfläche Userstory

```
ATOP - basteles-bastelknecht  2017/01/22  00:00:02  ------  105d7h50m53s elapsed
PRC | sys  579h26m | user  52d03h | #proc    181 | #zombie    0 | #exit      0 |
CPU | sys      12% | user     65% | irq      12% | idle    308% | wait      2% |
cpu | sys       7% | user     42% | irq      12% | idle     38% | cpu002 w  0% |
cpu | sys       2% | user     10% | irq       0% | idle     87% | cpu000 w  1% |
cpu | sys       1% | user      7% | irq       0% | idle     90% | cpu001 w  0% |
cpu | sys       1% | user      6% | irq       0% | idle     93% | cpu003 w  0% |
CPL | avg1    1.00 | avg5    1.08 | avg15   1.11 | csw 147908e5 | intr 80117e5 |
MEM | tot     3.9G | free  215.1M | cache   2.2G | buff  356.2M | slab  367.3M |
SWP | tot     0.0M | free    0.0M |              | vmcom   1.0G | vmlim   1.9G |
PAG | scan  1317e6 | stall      0 |              | swin       0 | swout      0 |
DSK |          sdb | busy      2% | read 67097e3 | write 3415e3 | avio 2.21 ms |
DSK |          sda | busy      0% | read 36071e3 | write 1192e4 | avio 0.89 ms |
NET | transport    | tcpi 47975e6 | tcpo 64188e6 | udpi 27541e4 | udpo 27855e4 |
NET | network      | ipi 482914e5 | ipo 457259e5 | ipfrw  26326 | deliv 4829e7 |
NET | eth0    ---- | pcki 48304e6 | pcko 45725e6 | si   43 Mbps | so   58 Mbps |
NET | lo      ---- | pcki  927320 | pcko  927320 | si    0 Kbps | so    0 Kbps |
NET | veth9f9 ---- | pcki   23804 | pcko   30048 | si    0 Kbps | so    0 Kbps |
NET | docker0 ---- | pcki   23804 | pcko   30044 | si    0 Kbps | so    0 Kbps |
                  *** system and process activity since boot ***
  PID  SYSCPU  USRCPU  VGROW  RGROW  RDDSK  WRDSK ST EXC S  CPU CMD
11811 382h08m  50d06h 726.7M 384.0M  10.8G  17.9G N-   - R  63% tor
   13  38h55m   0.00s     0K     0K     0K     0K N-   - S   2% ksoftirqd/1
   23  36h50m   0.00s     0K     0K     0K     0K N-   - S   1% ksoftirqd/3
    3  33h30m   0.00s     0K     0K     0K     0K N-   - S   1% ksoftirqd/0
 1902 817m03s 971m32s 836.0M  5508K 113.0M    76K N-   - S   1% transmission-d
```

Quelltext 13.1: atop ASCII Logausgabe

```
input {
  udp {
    host => "127.0.0.1"
    port => 25826
    buffer_size => 1452
    codec => collectd {
      typesdb => [
        "/usr/share/collectd/types.db"
      ]
    }
    type => "collectd"
  }
}

output {
  stdout {}
  jdbc {
    username => 'metrics'
    password => '***'
    connection_string => 'jdbc:postgresql://127.0.0.1:5432/metrics'
    statement => [
    "INSERT INTO log (host, timestamp, type_instance, plugin_instance,
    ↪  plugin, collectd_type, type, value)
    VALUES(?, CAST (? AS timestamp), ?, ?, ?, ?, CAST(? AS FLOAT))",
    "host", "@timestamp", "type_instance", "plugin_instance", "plugin",
    ↪  "collectd_type", "type", "value"]
  }

}
```

Quelltext 13.2: Logstash Konfigurationsdatei

```
{
  "name": "bastelfreak-ws.fritz.box",
  "values": {
    "architecture": "x86_64",
    "augeas": {
      "version": "1.7.0"
    },
    "augeasversion": "1.7.0",
    "bios_release_date": "06/16/2011",
    "system_uptime": {
      "days": 0,
      "hours": 6,
      "seconds": 22910,
      "uptime": "6:21 hours"
    },
    "os": {
      "architecture": "x86_64",
      "distro": {
        "codename": "n/a",
        "description": "Arch Linux",
        "id": "Arch",
        "release": {
          "full": "rolling",
          "major": "rolling"
        },
        "specification": "1.4"
      }
    },
  "timestamp": "2017-03-05T16:40:50.851590189+01:00",
  "expiration": "2017-03-05T17:10:50.851879236+01:00"
}
```


Quelltext 13.3: Ausgabe von facter

```
class profiles::grafana {

  class{'::grafana':
  }

  grafana_dashboard{'example_dashboard':
    grafana_url      => 'http://localhost:3000',
    grafana_user     => 'admin',
    grafana_password => '***',
    content          => template('profiles/grafana_dashboard.json.erb'),
  }

  grafana_datasource{'influxdb':
    grafana_url      => 'http://localhost:3000',
    grafana_user     => 'admin',
    grafana_password => '***',
    type             => 'influxdb',
    url              => 'http://localhost:8086',
    user             => 'admin',
    password         => '***',
    database         => 'ressources',
    access_mode      => 'proxy',
    is_default       => true,
  }
}
```

Quelltext 13.4: Puppet Profil für Grafana Beispiel

```
-- Master
create table measurement_master (
  id bigint not null,
  metadata json not null,
  data json not null )

-- Children
CREATE TABLE measurement_xMINUS12Hour (
  CHECK (
    to_timestamp(data->>'timestamp', 'YYYY-MM-DD HH24:MI:SS.MS')::timestamp
    between (current_timestamp - INTERVAL '12 hour')::timestamp
    AND (current_timestamp + INTERVAL '1 minute')::timestamp)
) INHERITS (measurement_master);

CREATE TABLE measurement_xMINUS1Day (
  CHECK (
    to_timestamp(data->>'timestamp', 'YYYY-MM-DD HH24:MI:SS.MS')::timestamp
    between (current_timestamp - INTERVAL '1 day')::timestamp
    AND (current_timestamp - INTERVAL '12 hour')::timestamp)
) INHERITS (measurement_master);

-- INDEX with JSON
CREATE INDEX measurement_xMINUS12Hour_timestamp ON
  measurement_xMINUS12Hour ((data->>'timestamp'));
CREATE INDEX measurement_xMINUS1Day_timestamp ON
  measurement_xMINUS1Day ((data->>'timestamp'));
```

Quelltext 13.5: Manuelle Partitionierung Postgres

```
marcellii@ci ~/puppet-profiles $ bundle exec rake test
profiles::grafana
  on archlinux-3-x86_64
    with all defaults
      should contain Class[profiles::grafana]
      should contain Grafana_dashboard[example_dashboard]
      should contain Grafana_datasource[influxdb]
      should compile into a catalogue without dependency cycles
  on ubuntu-16.04-x86_64
    with all defaults
      should contain Class[profiles::grafana]
      should contain Grafana_dashboard[example_dashboard]
      should contain Grafana_datasource[influxdb]
      should compile into a catalogue without dependency cycles
  on centos-7-x86_64
    with all defaults
      should contain Class[profiles::grafana]
      should contain Grafana_dashboard[example_dashboard]
      should contain Grafana_datasource[influxdb]
      should compile into a catalogue without dependency cycles
  on ubuntu-14.04-x86_64
    with all defaults
      should contain Class[profiles::grafana]
      should contain Grafana_dashboard[example_dashboard]
      should contain Grafana_datasource[influxdb]
      should compile into a catalogue without dependency cycles
  on debian-8-x86_64
    with all defaults
      should contain Class[profiles::grafana]
      should contain Grafana_dashboard[example_dashboard]
      should contain Grafana_datasource[influxdb]
      should compile into a catalogue without dependency cycles

Finished in 6.64 seconds (files took 3.54 seconds to load)
20 examples, 0 failures


20 examples, 0 failures
```

Quelltext 13.6: Test-Ausgabe der Unit-Tests

```
describe 'collectd class' do
  context 'default parameters' do
    # Using puppet_apply as a helper
    it 'works idempotently with no errors' do
      pp = <<-EOS
      class { 'collectd': }
      EOS
      # Run it twice and test for idempotency
      apply_manifest(pp, catch_failures: true)
      apply_manifest(pp, catch_changes: true)
    end
    describe package('collectd') do
      it { is_expected.to be_installed }
    end
    describe service('collectd') do
      it { is_expected.to be_running }
    end
  end
  context 'install plugins' do
    it 'works idemptontently' do
      pp = <<-EOS
      class { '::collectd': }
      class { '::collectd::plugin::memory': }
      class { '::collectd::plugin::rabbitmq': }
      EOS
      # Run it twice and test for idempotency
      apply_manifest(pp, catch_failures: true)
      apply_manifest(pp, catch_changes: true)
    end
    if fact(:osfamily) == 'Debian'
      describe file('/etc/collectd/conf.d/10-rabbitmq.conf') do
        it { is_expected.to be_file }
        it { is_expected.to contain 'TypesDB
        ↪  "/usr/local/share/collectd-rabbitmq/types.db.custom"' }
      end
    end
    describe service('collectd') do
      it { is_expected.to be_running }
    end
  end
end
```

Quelltext 13.7: Vollständiger Akzeptanztest für das Modul collectd

```
---
HOSTS:
  centos-7-x64:
    roles:
      - master
    platform: el-7-x86_64
    box: centos/7
    hypervisor: vagrant
```

Quelltext 13.8: Beispieldatei für eine Betriebssystem definition

```
class profiles::collectd (
 String $logstash_ip
){
  class{'::collectd':
    purge        => true,
    recurse      => true,
    purge_config => true,
    fqdnlookup   => false,
  }
  # connect to logstash
  collectd::plugin::network::server{$logstash_ip:
    port => 25826,
  }

  # collect cpu stats
  class{'::collectd::plugin::cpu':
    reportbystate    => true,
    reportbycpu      => true,
    valuespercentage => true,
    reportnumcpu     => true,
  }

  # collect disk stats
  class { 'collectd::plugin::df':
    fstypes          => ['nfs','tmpfs','autofs','gpfs','proc','devpts'],
    ignoreselected   => true,
    valuespercentage => true,
  }

  # collect cpu frquency stats
  contain '::collectd::plugin::cpufreq'

  # collect network stats
  include '::collectd::plugin::conntrack'
}
```

Quelltext 13.9: collectd Profil

```
class profiles::logstash (
  String $psql_ip,
  String $psql_user,
  String $psql_pw,
){

  # install and configure logstash
  contain ::logstash

  # get the plugin
  logstash::plugin{'logstash-output-jdbc':
    ensure =>  present,
  }

  # create the config
  logstash::configfile{'logstash.conf':
    content => epp("${module_name}/logstash.conf.epp"),
  }

  # get the jdbc driver
  archive{'/usr/share/logstash/vendor/jar/jdbc/postgresql-42.0.0.jar':
    ensure => 'present',
    source => 'https://jdbc.postgresql.org/download/postgresql-42.0.0.jar',
    user   => 'logstash',
    group  => 'logstash',
    notify => Service['logstash'],
  }
}
```

Quelltext 13.10: Logstash Profil

```
date; git clone git://github.com/collectd/collectd.git; \
cd collectd; git shortlog -s -n --no-merges | wc -l; \
git rev-list HEAD --count
So 22. Jan 13:55:44 CET 2017
Cloning into 'collectd'...
remote: Counting objects: 44666, done.
remote: Total 44666 (delta 0), reused 0 (delta 0), pack-reused 44666
Receiving objects: 100% (44666/44666), 15.66 MiB | 3.58 MiB/s, done.
Resolving deltas: 100% (33003/33003), done.
415
9059
```

Quelltext 13.11: collectd git clone

```
$ date; git clone git@github.com:python-diamond/Diamond.git; \
cd postgresql; git shortlog -s -n --no-merges | wc -l; \
git rev-list HEAD --count
So 22. Jan 19:57:47 CET 2017
Cloning into 'Diamond'...
remote: Counting objects: 21101, done.
remote: Compressing objects: 100% (92/92), done.
remote: Total 21101 (delta 37), reused 0 (delta 0), pack-reused 21009
Receiving objects: 100% (21101/21101), 3.32 MiB | 1.78 MiB/s, done.
Resolving deltas: 100% (10364/10364), done.
292
2922
```

Quelltext 13.12: Diamond git clone

```
$ date; git clone git@github.com:elastic/elasticsearch.git; \
cd elasticsearch; git shortlog -s -n --no-merges | wc -l; \
git rev-list HEAD --count
Su 27. Nov 15:17:45 CET 2016
Cloning into 'elasticsearch'...
remote: Counting objects: 737381, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 737381 (delta 0), reused 0 (delta 0), pack-reused 737378
Receiving objects: 100% (737381/737381), 297.56 MiB | 4.85 MiB/s, done.
Resolving deltas: 100% (408502/408502), done.
869
25989
```

Quelltext 13.13: Diamond git clone

```
$ date; git clone git://git.postgresql.org/git/postgresql.git; \
cd postgresql; git shortlog -s -n --no-merges | wc -l; \
git rev-list HEAD --count
Sa 26. Nov 13:23:17 CET 2016
Cloning into 'postgresql'...
remote: Counting objects: 604118, done.
remote: Compressing objects: 100% (105910/105910), done.
remote: Total 604118 (delta 511822), reused 586613 (delta 495859)
Receiving objects: 100% (604118/604118), 175.83 MiB | 5.48 MiB/s, done.
Resolving deltas: 100% (511822/511822), done.
41
41366
```

Quelltext 13.14: postgres git clone

```
#!/bin/bash
echo "device,receive,transmit"
while :; do
  awk '/enp1s0/ {print "enp1s0,"$2","$10}' /proc/net/dev
  sleep 60
done
```

Quelltext 13.15: /proc mit awk parsen

```
device,receive,transmit
enp1s0,28932776123,1786084537
enp1s0,28935220128,1786163375
enp1s0,28939447268,1786302393
enp1s0,28942260029,1786392663
enp1s0,28946985781,1786547807
enp1s0,28949168552,1786622616
enp1s0,28951976308,1786708990
```

Quelltext 13.16: traffic stats enp1s0

```
(streams
  (where (and (service "CPU temperature")
              (tags "core 0" )
              (metric > 75))
         (email "tim@bastelfreak.de")))
```

Quelltext 13.17: Einfache Riemann Konfiguration

```
$ bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns   Host ID                Rack
UN  127.0.0.1  692.18 MB  256     50.0%  95659096-693f19ad7679  rack2
UN  127.0.0.1  1.02 GB    256     50.0%  25a0c51a-6899f36e8f3c  rack1
```

Quelltext 13.18: cassandra status

```
package { 'htop':
  ensure => '2.0.2-1',
}
```

Quelltext 13.19: Puppet package resource für htop

```
require 'spec_helper'
describe 'profiles::grafana' do
  on_supported_os.each do |os, facts|
    context "on #{os} " do
      let :facts do
        facts
      end
      context 'with all defaults' do
        it { is_expected.to contain_class('profiles::grafana') }
        it { is_expected.to contain_grafana_dashboard('example_dashboard') }
        it { is_expected.to contain_grafana_datasource('influxdb') }
        it { is_expected.to compile.with_all_deps }
      end
    end
  end
end
```

Quelltext 13.20: Tests für das Grafana-Profil

Feldname	Beschreibung
host	Name des Servers auf dem das Event entstand
service	Name des Dienstes der das Event ausgelöst hat
state	Beliebiger Text unter 255Bytes. Zum Beispiel „Ok“, „Warning“
time	Uhrzeit an dem das Event erstellt wurde
description	Beliebiger Text
tags	Array mit Strings anhand dessen gefiltert werden kann
metric	Die eigentliche Information, zum Beispiel die CPU Temperatur
ttl	Anzahl an Sekunden die ein Event nach Erstellung gültig ist

Tabelle 13.1: Definition einer Event-Struktur in Riemann

Projekt	URL
Logstash	https://github.com/elastic/puppet-logstash/pull/334
Grafana	https://github.com/voxpupuli/puppet-grafana/pull/32

Tabelle 13.2: Beiträge zu Open Source Projekten

Projekt	URL
Puppet	https://tickets.puppetlabs.com/browse/PA-668
Puppet	https://tickets.puppetlabs.com/browse/PUP-7383
Mcollective	https://tickets.puppetlabs.com/browse/MCO-804
Grafana	https://github.com/voxpupuli/puppet-grafana/issues/35

Tabelle 13.3: Gemeldete Bugs in Open Source Projekten

14 Erklärung

Hiermit erklären wir, dass wir die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Diese Arbeit wurde keinem anderen Prüfungsausschuss in gleicher oder vergleichbarer Form vorgelegt.

.....	.....	.....
Tim Meusel	Marcel Reuter	Nikolai Luis