Sonderdruck aus OBJEKTspektrum

Echtzeitmonitoring-Architektur mit Cassandra und Solr

Praxisbeispiel Bundesagentur für Arbeit

Monitoringdaten von geschäftskritischen Informationssystemen sind, je nach Beschaffenheit und Verwendung, selbst wertvolle und kritische Daten. Damit sind aber auch ähnliche Anforderungen wie an Nutzdaten zu stellen. Eine wesentliche Anforderung an diese Daten ist die Durabilität, sprich eine dauerhafte Speicherung. Zudem sind Monitoringdaten von den eigentlichen Nutzdaten zu trennen, damit Monitoringabfragen und Analysen durchgeführt werden können, ohne den Produktivbetrieb zu gefährden. Das Monitoring muss ebenfalls wie das operative Bestandssystem hochverfügbar sein. Ein konkretes Beispiel hierfür ist das IT-Systemhaus der Bundesagentur für Arbeit (BA).

Dort werden teils hochkritische Systeme eingesetzt, deren Weiterentwicklung und Betrieb auf konsistente und langfristig gespeicherte Monitoringdaten angewiesen sind. Aufgrund dieser Anforderungen wurde eine hochverfügbare Echtzeitmonitoring-Lösung entwickelt und evaluiert, die auf dem Architekturstack Cassandra – Solr – Grafana aufbaut.

Dabei wurden die Fähigkeiten von Grafana bei der Abbildung von umfangreichen Zeitreihen mit dem robusten und einfachen Betrieb von Cassandra kombiniert. Die bisherige Lücke zu Solr schließt ein eigens entwickeltes Grafana-Plug-in, sodass Abfragen flexibel aus Grafana heraus an Solr und Cassandra erstellt werden können.

Der Architekturstack, dessen Einsatz und die sich daraus ergebenden Möglichkeiten in der Private Cloud, werden nachfolgend anhand des Beispiels BA vorgestellt.

Ausgangssituation: Aufbau eines NoSQL-Umfeldes inklusive Monitoring

Mit Blick auf die langfristige DevOps- und Agilitätsstrategie des IT-Systemhauses der BA in der Softwareentwicklung wurde ein Team geschaffen, das mit den Bereichen Private Cloud und konventionellen Datenbanken vertraut ist. Ziel dabei war es, zusätzliches Wissen im NoSQL-Umfeld proaktiv aufzubauen und entsprechende Lösungen anzubieten. Eine dieser Lösungen beinhaltet den Architekturstack Cassandra, Solr, Spark und Grafenfunktionen.

Dieser Architekturstack wurde ermöglicht, indem eine lastfähige und hochskalierbare Infrastruktur – ein sogenanntes „Bare Metal“ – aufgebaut und dem Stack zugrunde gelegt wurde. Eine Herausforderung bestand darin, dass so zwar eine leistungsstarke Infrastruktur vorhanden war, die als ausgereifte Plattform auch Big Data verarbeiten kann. Was jedoch fehlte, war eine Out-of-the-Box(OOTB)-Monitoring-Lösung oder gar eine OOTB-Lösung für echtzeitnahes Monitoring, um die eingangs erwähnten Anforderungen wie Durabilität, Hochverfügbarkeit und analytische Leistungsfähigkeit zu erfüllen.

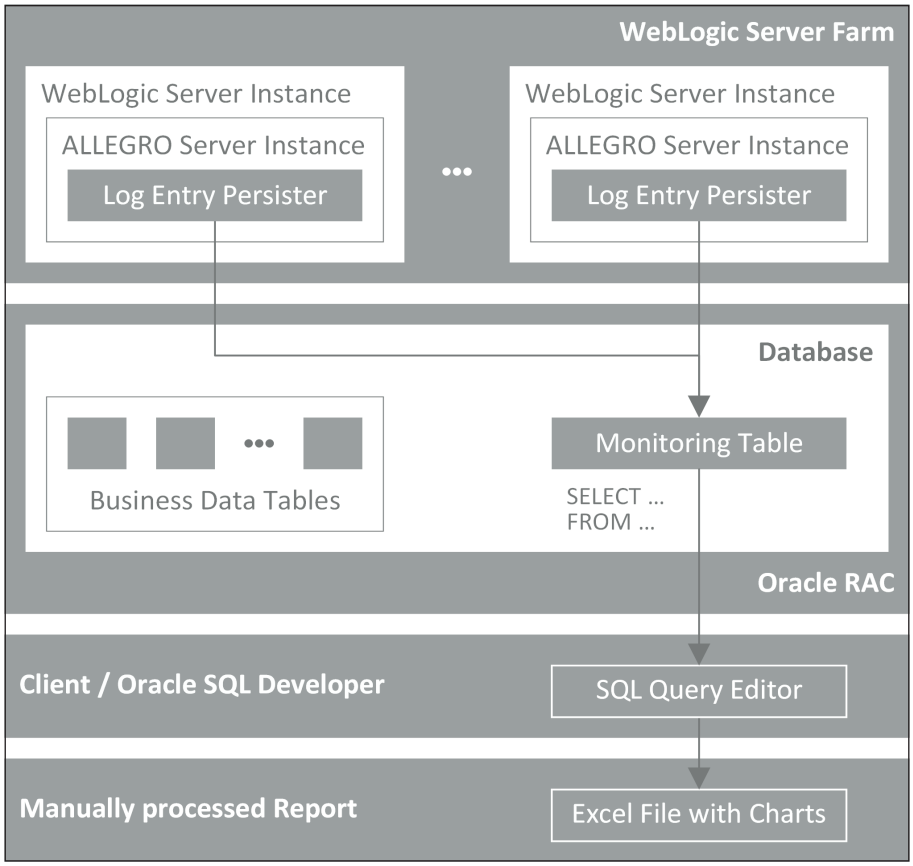

Abb. 1: IST-Architektur für das Monitoring des Bestandssystems

Diese musste somit erst auf dem Stack aufbauend entwickelt und evaluiert werden. **Abbildung 1** zeigt die IST-Architektur für das Monitoring des Bestandssystems.

Architektur: Abwägung zwischen Neubau und Out-of-the-Box

In Hinblick auf die Architektur musste damit folgende Frage beantwortet wer-

den: Baut man eine Lösung basierend auf der vorhandenen Infrastruktur auf, oder greift man auf eine OOTB-Lösung wie den ELK-Stack zurück und errichtet hierfür eine Umgebung mit entsprechender Infrastruktur?
Diese Frage lässt sich nicht pauschal beantworten. Vielmehr sollte zuerst erörtert werden, was eigentlich erreicht werden soll, um dann den Fokus entsprechend zu legen. Im konkreten Beispiel steht das Monitoring eines großen und kritischen Informationssystems im Mittelpunkt. Das Ziel war dabei die Verlagerung der Monitoringfunktionalität aus dem operativen System in einen unabhängig zu betreibenden Monitoringservice. Zudem sollte das operative Bestandssystem durch die neue Lösung nicht beeinträchtigt werden.
Weiterhin sollten Nutzer des Monitorings nicht mehr, etwa für zeitaufwendige Abfragen an die Operativdatenbank, auf die Ressourcen des Bestandssystems zugreifen müssen, sondern risikolos auf einem replizierten Datenbestand arbeiten können.

Technische Entscheidungsaspekte

Diese Anforderung ist besonders wichtig, da das zugrunde liegende Bestandssystem die meisten fachlichen Monitoringdaten bereits parallel zu den Fachdaten in einer relationalen Datenbank speichert. Mit der Zeit werden die entsprechenden Tabellen immer größer und flexible Abfragen dieser Daten ohne Beeinträchtigung der operativen Fachprozesse im System zunehmend schwieriger.
Die Beschaffenheit der Daten hingegen ist einfach: Jeder Eintrag besitzt einen Zeitstempel und diverse Metriken wie Anzahl oder Laufzeiten von Geschäftsvorgängen, Schnittstellenaufrufen usw. Diese Art der Datenverwaltung entspricht also der Speicherung von Zeitreihen, bei denen ein konstanter Strom von Sensorwerten (fachliche und technische Metriken) über die Zeit verteilt wird. Grafana ist auf die Visualisierung dieser Art von Daten spezialisiert und stellt gemäß der Anforderungen an Abfrage und grafische Auswertung die bessere Alternative im Vergleich zu Kibana dar.
Ein Blick auf die Statistiken von GitHub zeigt, dass Grafana (https://github.com/grafana/grafana) sowohl bei GitHub Stars als auch bei Forks (**siehe Tabelle 1**) vor Kibana (https://github.com/elastic/kibana) liegt. Beide Plattformen besitzen sehr aktive Communities mit täglich durchgeführten Aktualisierungen am Codebestand, wobei Kibana hinsichtlich der Aktivitäten auf Stack Overflow führend ist, während auf Reddit Grafana deutlich mehr Erwähnung findet.

	Grafana	Kibana
GitHub Stars	23,6 Tsd.	10 Tsd.
GitHub Forks	4,23 Tsd.	3,89 Tsd.
GitHub Commits Master Branch Aug. 2018	227 von 28 Autoren	232 von 60 Autoren
Reddit Points	476	191
Stack Overflow Questions	1,17 Tsd.	3,06 Tsd.

Tabelle 1: GitHub-Statistiken von Grafana und Kibana im Vergleich

Letztlich kam für die Nutzung der Cassandra-Infrastruktur nur Grafana infrage, da sich nur damit beliebige Datenquellen per Plug-in einbinden lassen. Kibana ist hingegen auf Elasticsearch angewiesen.

Organisatorische Entscheidungsaspekte

Hinzu kommt, dass die Cassandra-Infrastruktur einschließlich Server, Plattformen und Lizenzen im IT-Systemhaus der BA bereits vorhanden war. Das Beispiel verdeutlicht, dass neben technischen Bedingungen auch organisatorische Umstände bei Architekturentscheidungen berücksichtigt werden sollten. Im vorliegenden Fall spricht die Mehrzahl technischer Argumente für den DataStax-Architekturstack (Cassandra mit Solr), es sind jedoch auch Szenarien denkbar, wo die organisatorischen Gegebenheiten ausschlaggebend für die Architekturwahl sein können. Im hier beschriebenen Fall lag für die Anbindung an Cassandra noch keine fertige Lösung vor, sodass zu Beginn ein Plug-in entwickelt werden musste.
Für das Auslagern und (redundante) Speichern der Daten war, wie beschrieben, eine auf Cassandra aufbauende Lösung in

der Organisation vorhanden, die wie das operative System hochverfügbar ist. Der verwendete Storage ist, bedingt durch die verteilte Architektur von Cassandra, flexibel erweiterbar. Die in Cassandra verwendete vereinfachte Darstellung der Daten als eingebettete Key-/Value-Paare und die schemalose Struktur sind für die dargestellten „Sensorwerte" geeignet. Die hohe Insert-Rate von Cassandra stellt ebenfalls keinen Engpass für den unstrukturierten Datenstrom an Monitoringwerten dar und ermöglicht es, die operative Datenbank zu entlasten. Insgesamt lag also eine hochverfügbare Datenbank mit vergleichsweise günstigen Storage-Optionen vor, die zur Beschaffenheit der Daten passte.

Die Lösung: DataStax-Stack mit Cassandra-Datenbank

Grundsätzlich lassen sich zwei Ausprägungen von Monitoringsystemen unterscheiden: Zum einen die fortlaufende Auswertung von langzeitgespeicherten Daten mit echtzeitnaher Visualisierung und Verzögerungen im Sekundenbereich. Zum anderen ein Echtzeitmonitoring von ephemeren Durchlaufdaten mit Visualisierung in Echtzeit ohne Verzögerung, deren Monitoringdaten nicht für die Langzeitspeicherung vorgesehen sind. Vor dem Hintergrund der Anforderung, Monitoringdaten langfristig vorzuhalten, wurde hier der erstgenannte Ansatz gewählt. Eingesetzt wurde dazu der DataStax-Architekturstack, der als „Unterbau" die Cassandra-Datenbank nutzt.

Dadurch wird ein sowohl technisch als auch fachlich flexibleres Monitoring ermöglicht, einschließlich komplexerer Abfragen mit NoSQL. Die Visualisierung findet in einem zentralen Dashboard statt, Exporte und Nachbearbeitungen in Excel sind nicht mehr notwendig. Auswertungen können auch vom Fachbereich direkt im Dashboard eingesehen werden, das Risiko der Produktionsbeeinträchtigung durch Abfragen entfällt. Das Auswerten und Korrelieren von Fehlermeldungen (Exceptions) ist direkt möglich – ohne den nachgelagerten Sammelprozess von Logs über 256 Server. Perspektivisch ergibt sich eine Kostenersparnis dank günstigerem Speicher anstelle des Oracle RACs.

Die neuen Auswertungsmöglichkeiten beinhalten unter anderem die Identifikation von Engpässen und langer Latenz in bestimmten Dienststellen. Analysen sind über den vollen Datenzeitraum, beispielsweise bei Langzeitanalysen der Performanz, möglich. Da Abfragen im Vergleich zu relationalen Datenbanken deutlich schneller sind, können diese auch häufiger als einmal pro Woche durchgeführt werden. Zudem lassen sich mit dem neuen Architekturstack neue Monitoringanforderungen schneller umsetzen. Während Tabellen in der relationalen Datenbank sehr groß und daher nicht einfach um neue Spalten erweiterbar sind, sind Schemaoperationen in Cassandra sogar im Nachgang und unabhängig von der Größe deutlich effizienter.

Aufgrund der Langzeitspeicherung der Monitoringdaten können Metriken zudem noch im Nachhinein ermittelt werden. Ansonsten wären beispielsweise für Daten, die älter als ein Monat sind und für die nur der Durchschnitt beibehalten wurde, nachträglich keine Perzentile mehr zu ermitteln.

Die Cassandra-Datenbank hingegen ist für eine effiziente Big-Data-Verarbeitung ausgelegt und ermöglicht Incident-bezogene Abfragen – zum Beispiel *Finde alle Fehler, die im letzten Monat in einer bestimmten Java-Klasse aufgetreten sind.* Die Nachverfolgung von Incidents wird durch den neuen Architekturstack vereinfacht. Tritt ein Fehler in einer bestimmten Java-Klasse auf, kann dank der neuen Monitoringarchitektur ermittelt werden, wann und wo das Problem überall aufgetreten ist.

Weiterhin bietet Cassandra eine sehr hohe Kompression und kommt im Vergleich zur Oracle DB mit 25 Prozent des Speicherplatzes aus – ein Wert, der nur von wenigen Datenbanken wie Dalmatiner DB oder InfluxDB übertroffen wird. Diese sind allerdings kein Standard der Bundesagentur.

Weitere Vorteile resultieren daraus, dass Produktionskopien des operativen Bestandssystems ohne Monitoringdaten einfacher und schneller anzufertigen sind. Die Monitoringarchitektur liefert zusätzliche Erkenntnisgewinne hinsichtlich der Performanz technischer Komponenten und ist aufgrund von Synergieeffekten für weitere IT-Verfahren interessant. Hinzu kommt, dass der neue Architekturstack

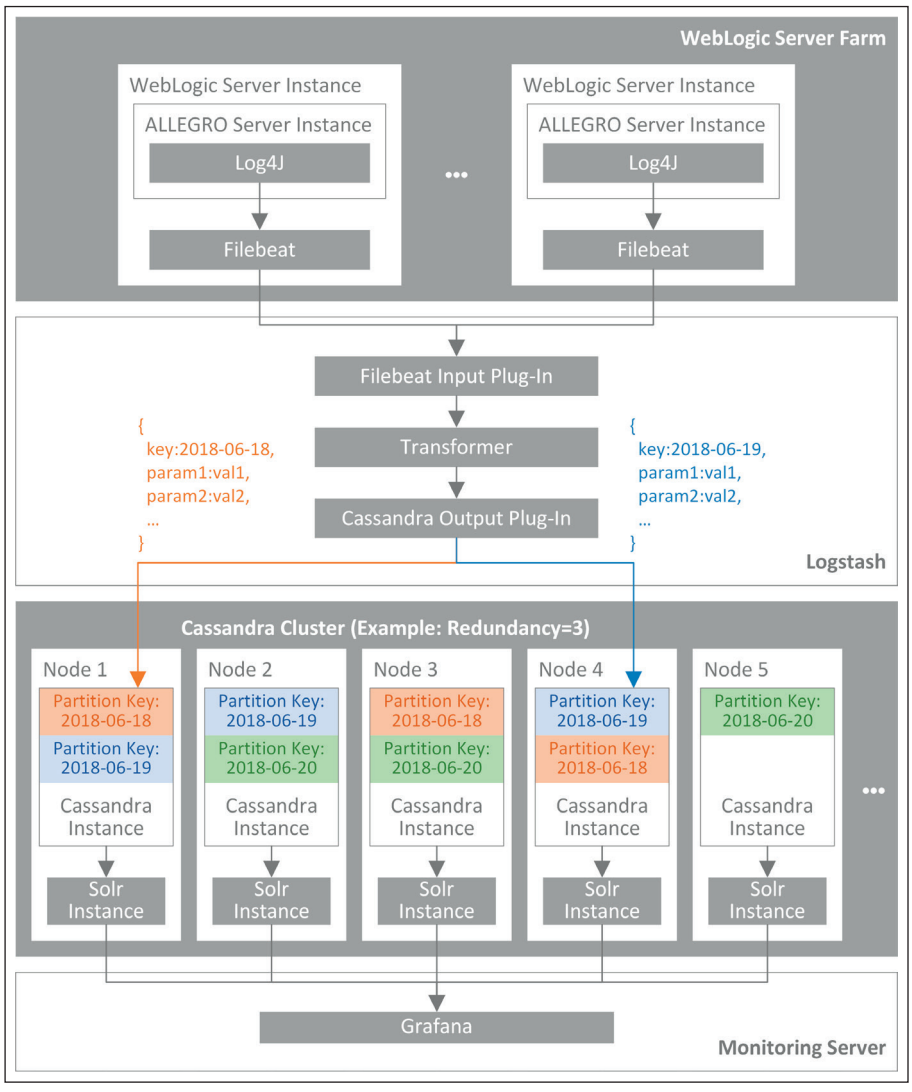

Abb. 2: Gesamtarchitektur des NoSQL-basierten Monitorings

etabliert ist: Bekannte quelloffene Lösungen bauen auf demselben Stack auf – nur ein Beispiel dafür ist die Monitoringplattform Graphite in Kombination mit Cyanite, einer auf Cassandra aufbauenden Zeitreihendatenbank [Dixo17]. Während Influx nur clusterfähig in der kommerziellen, nicht-quelloffenen Variante InfluxEnterprise ist, ist Solr generell cluster- und cloudfähig.

Beschaffenheit des Architekturstacks

Oberhalb der Datenbankschicht befindet sich ein Solr-Index, der komplexere, lange und verschachtelte Abfragen ermöglicht. Solr verfügt über Statistikfeatures wie die Abfrage von Perzentilen, Durchschnittswerten, Standardabweichungen oder der Approximation der Kardinalität von Attributwerten mittels HyperLogLog-Algorithmus. Die Abfragen selbst können mittels JSON oder XML erfolgen, wobei hier die Wahl auf JSON fiel, da das auf Grafana basierende Frontend in JavaScript geschrieben ist und sich JSON für REST-Schnittstellen besser eignet.

Solr unterstützt zudem mit der Facettierung einen Mechanismus, der das Suchergebnis anhand verschiedener Filterkriterien organisiert, verfeinert und formatiert: „*Faceting is the mechanism provided by Solr to categorize results in a meaningful arrangement on indexed fields.*“[NMV18] Ein simples Beispiel ist die Facette der Displaygröße *mindestens 6 Zoll* und Geräteklasse *Highend* bei einer Smartphone-Suche in einem Onlineshop. Im Vordergrund steht letztlich die analytische Fähigkeit einer Suche. So können Facetten beliebig verschachtelt werden. Solr bietet auch eine Pivot-Facette. Sie gibt Aggregatberechnungen in mehrere Ebenen von hierarchischen Ergebnis-Facetten zurück, die, ähnlich zu Pivot-Tabellen in Excel, Werte aus unterschiedlichen Feldern pivotiert.

Die Antwort einer einzelnen Solr-Abfrage enthält somit die Berechnung einer Mehrzahl von Aggregataufteilungen. Ein treffendes Beispiel ist die Abfrage als 3-Level-Pivot-Facette der Anzahl von 4- bis 5-Sterne-Restaurants in den Top-3-Städten der Top-3-Bundesländer [GrPo14]. Das Ergebnis dieser Abfrage enthält die Daten in dreifacher Aufschlüsselung – nach den Feldern *Bundesland*, *Stadt* und dem *Sterne-Rating*.

Setzt man Solr als Einzellösung ein, dann profitiert man von den Replikationsmechanismen, die Solr generell mitbringt. Hierzu zählen ein cloud-orientierter Mechanismus, sprich den in Hadoop enthaltenen Koordinationsdienst ZooKeeper, sowie eine Konfiguration der Solr-Knoten für Master-/Slave-Indexreplikation. Genutzt wird hierbei allerdings ein gesamter Architekturstack, der über Solr hinausgeht und die verteilte Suche auf Cassandra-Ebene vollzieht. Dadurch wird das verteilte, hochverfügbare Suchen nach Peer-to-Peer- anstelle des Master-/Slave-Ansatzes ermöglicht, und zwar ohne externe Prozesse wie etwa ZooKeeper.

Abbildung 2 zeigt die gesamte Architektur des beschriebenen NoSQL-basierten Monitorings. Daraus geht hervor, dass neben Cassandra und Solr noch drei weitere Schichten existieren: die WebLogic Server Farm, in der die zu loggenden Ereignisse auftreten, Logstash, eine Vermittlerschicht zur Übertragung der Ereignisse/Lognachrichten an Cassandra, sowie die Präsentationsschicht Grafana, einschließlich Monitoring-Dashboard.

Wie eingangs erwähnt, ist die Konzipierung und das Umsetzen eines dediziert auf hohe Anforderungen ausgerichteten Architekturstacks gegenüber dem bloßen Einsatz einer OOTB-Lösung aufwendiger. Die Entwicklung des Cassandra-Output-Plug-ins für Logstash zählt dazu. Es ist in Ruby geschrieben, sodass Logstash in einer Ruby-Umgebung läuft. Das Plug-in basiert auf dem *Vitalyros Logstash Cassandra Output*-Plug-in [Vita16].

Eine weitere Herausforderung bestand in der Entwicklung eines Grafana-Plug-ins für die Anbindung von Solr als Datenquelle. Im Gegensatz zu verschiedenen Datenquellen wie Kairos, Prometheus, Graphite, CloudWatch, InfluxDB usw. gibt es für Solr keine Plug-ins.

Da Grafana auf Go und Angular basiert und Solr eine RESTful-Programmierschnittstelle besitzt, lässt sich eine Integration nach Best-Practice-Ansätzen durchführen. Zu erwähnen ist, dass Grafana Angular 1 verwendet und es nicht geplant ist, auf eine höhere Version zu migrieren. Das liegt daran, dass die Entwickler von Grafana die komplette Codebasis von Angular zu dem leichtgewichtigen Framework React migrieren wollen, was sie selbst als mehrjähriges Vorhaben bezeichnen [Graf15].

Die aktuelle, auf AngularJS und TypeScript aufbauende Plug-in-Architektur wird genutzt, um das eigens entwickelte Datasource-Plug-in für die Anbindung an Solr umzusetzen. Die Aufbauarchitektur zur Interaktion von Grafana und Solr

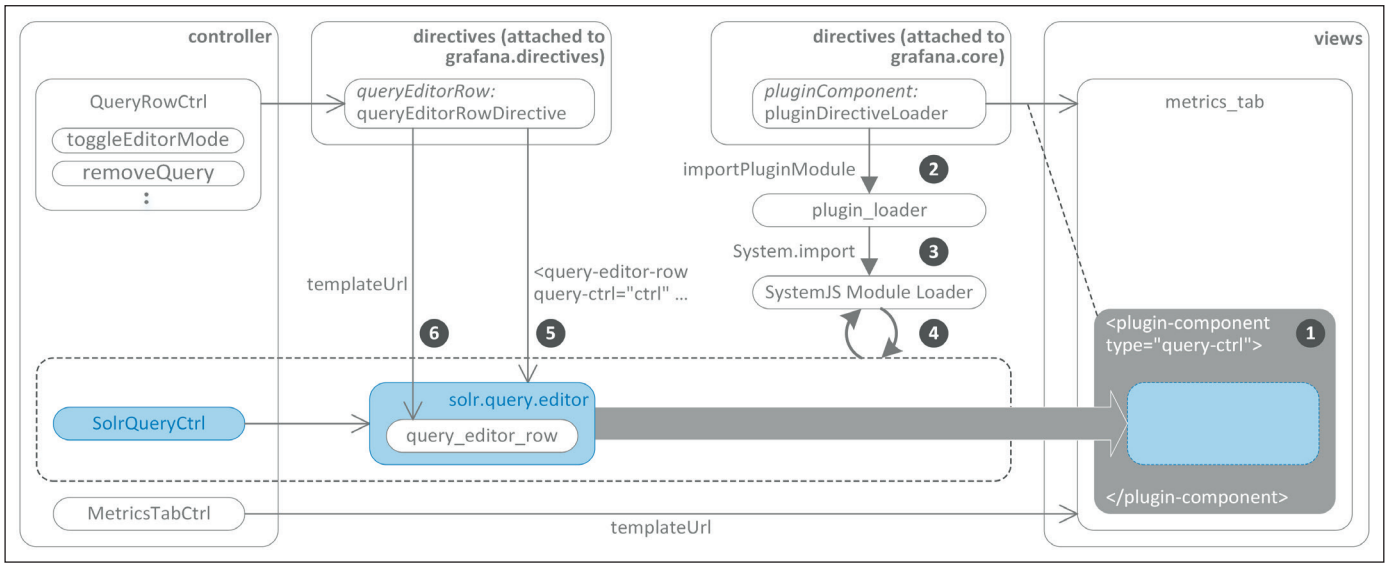

Abb. 3: Interaktion des Grafana-Backends mit unserem Plug-in für die Anbindung von Solr als Datenquelle

über das eigens implementierte Solr-Plug-in ist in **Abbildung 3** dargestellt.

Die Architektur ist gegliedert in Model (Daten), Controller (Funktionen) und Views (graphische Oberfläche), was letztlich auf AngularJS zurückzuführen ist. Des Weiteren gibt es Direktiven, welche HTML um individuelle Funktionen erweitern, und Services, welche in Form von Singletons diverse Funktionen mit hohem Wiederverwendungswert für Controller oder Direktiven bereitstellen. Ein weiteres architekturelles Gliederungsinstrument ist die Trennung von Direktiven, Controller und Services in Module. So werden grundlegende Querschnittsfunktionalitäten von Grafana dem Kernmodul (Grafana-Core) zugeordnet, eher fachliche Funktionalitäten den spezifischen Modulen.

Wie funktioniert aber nun die Einbindung unseres Solr-Plug-ins sowie dessen Interaktion mit dem Grafana-Backend? Plug-in-Module werden dynamisch mittels SystemJS geladen (**siehe Abbildung 3**). Die Plug-in-Struktur orientiert sich an dem durch AngularJS vorgegebenen MVC-Prinzip. Es gibt eine Klasse, die sich um die Kommunikation mit der jeweiligen Datenquelle kümmert, beziehungsweise um den Datenbezug *(Data Retrieval)* aus der Quelle.

Hier kommt der Controller *SolrQueryCtrl* ins Spiel, der verantwortlich für die Konfiguration der Metriken ist. **Abbildung 4** zeigt das in Grafana integrierte Plug-in. Im *Metrics*-Tab wird die an Solr zu stellende Abfrage konfiguriert sowie ein JavaScript, welches die von Solr gegebene Antwort in das von Grafana vorgegebene

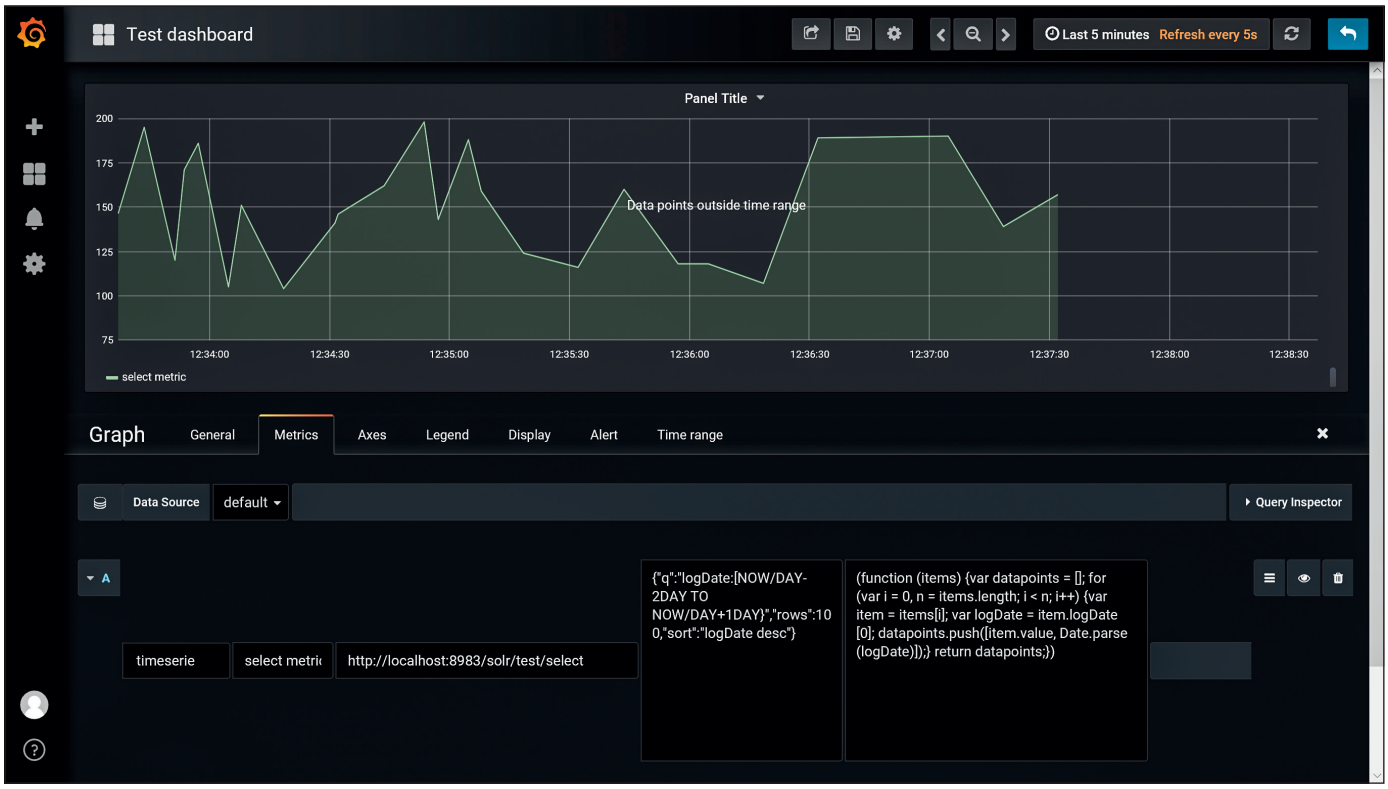

Abb. 4: Screenshot des Grafana-Plug-ins für die Anbindung der Solr-Datenquelle

Literatur & Links

[Dixo17] J. Dixon, Monitoring with Graphite: Tracking Dynamic Host and Application Metrics at Scale, O'Reilly, 2017

[Graf15] Grafana Frontend: Code structure, architecture and plans for the future, siehe: **https://github.com/grafana/grafana/wiki/Frontend:-Code-structure,-architecture-and-plans-for-the-future**

[Graf18] Grafana: Developing Datasource Plugins, siehe: **http://docs.grafana.org/plugins/developing/datasources/**

[GrPo14] T. Grainger, T. Potter, Solr in Action, Manning, 2014

[NMV18] S. Nair, C. Mehta, D. Vasoya, Mastering Apache Solr 7.x: An expert guide to advancing, optimizing, and scaling your enterprise search, Packt Publishing, 2018

[Sul17] E. Sultanow u. a., Praxisbeispiel des IT-Systemhauses der Bundesagentur für Arbeit, in: OBJEKTspektrum, 06/17

[Vita16] Vitalyros Logstash Cassandra Output Plugin, siehe: **https://github.com/vitalyros/logstash-output-cassandra/**

Format transformiert. Wie die JSON-Antwort von Solr beschaffen ist, hängt von der Abfrage ab. So können Daten in unterschiedlichen Facetten abgefragt und von Solr anforderungsgemäß per Facettierung unterschiedlich strukturiert und hierarchisiert zurückgeben werden.

Wie in **Abbildung 3** dargestellt, ist dem Controller *SolrQueryCtrl* die View *solr.query.editor* zugeordnet. Dieser View ist genau der Teil der grafischen Oberfläche, in dem die Solr-URL, -Abfrage und das dazugehörende JavaScript festgelegt werden. Die Werte (URL, Abfrage und JavaScript) der Eingabefelder übergibt Grafana beim Aufruf der in SolrDataSource implementierten Funktion *query*, innerhalb der die Kommunikation mit Solr stattfindet.

Mit dieser Offenheit in puncto Konfiguration sind zahlreiche Abfrageszenarien denkbar: Es können beliebig komplexe Abfragen an Solr im Grafana-Dashboard eingestellt sowie eine eigene JavaScript-Funktion definiert werden, welche die von Solr zurückgelieferten Ergebnisse in das Grafana-Zeitreihenformat überführt.

Resümee

In der Architektenwelt gilt nicht immer das Out-of-the-Box-Prinzip. Es gibt Situationen, in denen eine vorhandene Infrastruktur nicht mit einer fertigen OOTB-Lösung zusammenpasst. In einem solchen Fall besteht die Herausforderung darin, diejenige Architekturlösung zu erarbeiten, die unter den gegebenen Umständen am besten und für künftige Anforderungen langfristig und nachhaltig geeignet ist. Diesem Ansatz entspricht das hier vorgestellte Vorgehen, einen Architekturstack für komplexe Monitoringanforderungen aufzubauen.

Die Grundlage hierfür bildet eine hochskalierbare, auf dem DataStax-Architekturstack basierende Infrastruktur, die dank modularer Plug-in-Struktur flexibel erweiterbar ist. Im Gegenzug wurde dazu ein eigenes Grafana-Plug-in entwickelt, das Solr als Datenquelle verwendet, sowie ein Cassandra-Output-Plug-in für Logstash, welches die Log-Daten in diese Infrastruktur schreibt.

Zusammenfassend bleibt festzuhalten, dass sich der getätigte Aufwand gelohnt hat. Der Proof-of-Concept hat nachgewiesen, dass der Architekturstack die Anforderungen an den Umgang mit geschäftskritischen Monitoringdaten erfüllt und Betrieb und Weiterentwicklung des operativen Systems entlastet. ||

Die Autoren

Dr. Eldar Sultanow
(eldar.sultanow@capgemini.com)
ist Architekt bei Capgemini Deutschland. Davor war er fünf Jahre CIO eines mittelständischen Pharmagroßhändlers, nachdem er über vier Jahre als JEE-Entwickler/Architekt für ein E-Commerce-Unternehmen in Berlin tätig war, das ein führendes Verbraucherinformationsportal Europas entwickelt und betreibt.

Oliver Weiß
(oliver.weiss@capgemini.com)
hat mehr als zehn Jahre Erfahrung in Softwaregroßprojekten des Public Sektors. Er ist bei Capgemini aktiv und nimmt in Kundenprojekten eine Führungsrolle bei der Evolution komplexer Systemlandschaften ein. Aktuell ist Oliver Weiß Chefarchitekt in einem der größten öffentlichen IT-Verfahren, welches Hunderttausende Transaktionen pro Tag mit einem Jahresvolumen von über 25 Milliarden EUR vollzieht.

Matthias Seßler
(matthias.sessler@arbeitsagentur.de)
ist im IT-Systemhaus der Bundesagentur für Arbeit der Service-Betrieb-Infrastruktur (SBI)-verantwortliche Lead Architekt für Datenbanktechnologie. Davor war er 16 Jahre bei Microsoft im Bereich Data Insights verantwortlich.