JAVA Mag

Java™magazin

Java | Architektur | Software-Innovation

Mehr geht immer

Java EE einmal mit allem

www.javamagazin.de

Ausgabe 2.2017

Deutschland € 9,80
Österreich € 10,80
Schweiz sFr 19,50
Luxemburg € 11,15

4 194586 709801 02

Teil 2: Nachhaltige Analyse gesammelter Daten zur Projektunterstützung

Mit Blick auf die Zukunft

Gerade in verteilten Umgebungen sind Management und Monitoring wichtig, damit der Betrieb auch während weiterer Entwicklung mit Sicherheit gewährleistet ist. Zeitseriendatenbanken und geeignete Visualisierungen helfen dabei, Ordnung in das Daten- und Analysechaos zu bringen. Dabei ist immer ein kritischer Blick nötig, welche Daten überhaupt relevant sind.

von Joachim Arrasz und Jonathan Buch

Im letzten Teil dieser Artikelserie haben wir gezeigt, wie uns das Konfigurationsmanagement dabei helfen kann, automatisch neue Versionen einer implementierten Lösung zu installieren. Außerdem konnten wir aufzeigen, wie wichtig die Verwaltung maschinell zu verarbeitender Infrastrukturbeschreibungen selbst ist. Alle diese Punkte sind Voraussetzungen für gutes und effizientes Monitoring. Aber was erreichen wir eigentlich mit einem guten und effizienten Monitoring? Wir können damit einen sicheren und nachhaltigen Betrieb gewährleisten. Nur wenn man die Anwendung und Schwellwertparameter ständig überwacht, bekommt man ein Gefühl der Sicherheit im Betrieb der Anwendung. Für folgende konkrete Aufgaben suchen wir Lösungen. Dabei wollen wir aufzeigen, wie wir lernen, sie zu vergleichen und einzuschätzen, ob sie in unserem Szenario die sinnvollste Lösung sind:

- Monitoring
- Management
- Statistik
- Kritische Pfade
- Forecasts
- fachliche Auditierungen

Wie können wir uns den verschiedenen Aufgaben annähern? Dokumentiert werden sollte die Serverinfrastruktur, deren Monitoring, Management und Provisionierung. Dazu SLAs, securityrelevante Themen wie Zertifikate, VPN oder Security Tokens. Ebenso sollten hier schon Systemgrenzen klar definiert sein und beispielsweise auch Konsumenten der Schnittstellen angerissen werden. Außerdem kann man sich hier schon Gedanken über die Unterstützung des Supports machen, wozu neben dem Logging auch die Visualisierung von Statistiken und Metriken gehört. Visualisierungsarten wie Timelapses, Heatmaps oder kritische Pfade kann man ins Auge fassen. Um einen Anhaltspunkt zu bekommen, welche Informationen relevant sind, kann man beispielsweise das arc42-Template heranziehen [1]. Sobald wir diese Informationen gesammelt haben, pro System oder für den Gesamtkontext (Kasten), wollen wir versuchen, alles zu aggregieren.

Schließlich müssen die Informationen klassifiziert werden. Dadurch können wir Erfahrungswerte beisteuern, die für den stabilen Betrieb und somit für das Team relevant sind. Wie kritisch diese Einzelinformationen allerdings konkret für das Team beziehungsweise den Betrieb durch das Team sind, muss individuell bewertet und somit in der Visualisierung dargestellt werden (**Abb. 2**).

Alles hat Vor- und Nachteile

Graphite als Vertreter einer Zeitseriendatenbank ist wohl als Erstes zu erwähnen [2]. Aufbauend auf RRD-basierenden (Round-Robin-Database-)Werkzeugen wie RRDtool oder Munin wurde dieses 2008 unter der Apache-Lizenz 2.0 veröffentlicht. Es war zu jenem Zeitpunkt bereits zwei Jahre in Entwicklung und hat schon damals ein HTTP-API und Dashboard angeboten. Es ist somit nicht nur eine reine Datenhalde mit Anzeige, sondern ermöglicht auch angepasste Auswertungen. Nach einer Installation, die wohl nur bei Systemadministratoren wohlige Gefühle auslösen kann, ist die Bedienung sehr einfach. Neue Metriken werden zeilenweise in Form von Key/Value-Paaren an einen TCP-Port gesen-

Artikelserie

Teil 1: Stabile Produktion und Betrieb von Software in der Industrie
Teil 2: Nachhaltige Analyse gesammelter Daten zur Projektunterstützung
Teil 3: Optimierungsmaßnahmen gezielt umsetzen

det. Das Dashboard ist schlicht, bildet aber den Funktionsumfang von Graphite gut ab und ermöglicht es, die Datenbank durch eine Baumstruktur einfach zu durchsuchen. Seit 2012 entwickelt eine kleine Community das Projekt stetig weiter. Trotz der kleinen Versionsnummer und nach Metall riechenden Zusammenstellung von Python-Skripten erledigt Graphite seinen Dienst sehr stabil – solange bei höheren Lasten von Millionen an Metriken pro Minute genügend RAM und SSDs als Datenspeicher vorhanden sind.

Neuling der Reihe ist InfluxDB [3]. Es wurde 2013 entwickelt und gilt als Antwort der Firma InfluxData auf die Defizite von Graphite. Erst vor kurzer Zeit hat InfluxDB Version 1.0 erreicht und lässt sich seit Anfang 2016 mit dem Austausch der Storage Engine sehr gut einsetzen. Im Gegensatz zu Graphite lässt InfluxDBs Datenbankunterbau die RRD-ähnliche Speicherung von Zeitdaten hinter sich und orientiert sich, um dem hohen Schreibaufkommen gerecht zu werden, an neueren NoSQL-Entwicklungen. Die Daten werden in einer Art komprimierter Baumstruktur performant gespeichert. Auch hier sind das API und die Abfragesprache König. Dadurch, dass sich wesentlich mehr Daten als ein einfaches Zeit/Name/Wert-Tupel speichern lassen, ist auch eine komplexere Abfragesprache notwendig. Die Einstiegshürde ist für den Anfänger höher. Die ausgelieferte Administrationsweboberfläche ist zwar schön anzusehen, aber der Datenbankinhalt ist nicht einfach zu erfassen und benötigt noch zusätzliche Werkzeuge. Auch das Speichern von Metriken ist neben einer Graphite-kompatiblen Schnittstelle komplexer, sofern man die Strukturierungsmöglichkeiten von InfluxDB möglichst gut nutzen möchte. Neben numerischen Daten lassen sich hier mit so genannten Tags schemalos auch textuelle Anmerkungen speichern, die helfen, die Daten zu interpretieren und zu klassifizieren.

Wenn man noch weiter in die schemalose Speicherung von Daten geht, gelangt man unweigerlich zu Elasticsearch [4]. Dessen Spezialgebiet ist die Verarbeitung und Indizierung von Texten. Es bietet weniger Funktionalität, die auf die Verarbeitung von Zeitserien zugeschnitten ist. Elasticsearch, unter einer Open-Source-Lizenz von Elasticsearch BV zur Verfügung gestellt, ist zwar mit einem Erscheinungsdatum von 2003 der älteste im Bunde, aber trotzdem ein sehr modernes und viel verwendetes Produkt. Neben einem REST-API gibt es ein reichhaltiges Plug-in-Angebot, mit dem man Elasticsearch erforschen kann. Das API ist noch reichhaltiger als bei dem Vorgänger. Während das Anlegen von Einträgen noch recht einfach geht, ist der Zusammenbau von komplexen Aggregationen doch deutlich aufwändiger, und das ist ein wichtiger Punkt. Wie immer ist der Einsatz einer Lösung stark davon abhängig, was die Umgebung benötigt. Während es für den einen angenehm ist, mit Graphite Werte für Jahre mit abnehmender Granularität zu speichern und dafür einen überaus simplen und reichhaltigen Einblick in seine Infrastruktur zu bekommen, gehen die anderen Lösungen in anderen Bereichen darüber hinaus. Neben freien On-Premise-Lösungen gibt es natürlich auch kommerzielle Angebote in verschiedenen Varianten. Größere Cloud-Anbieter sind hier Librato und Datadog. Wobei auch hier gerade viel Bewegung im Markt ist. Neue Anbieter wie Honeycomb sind gute Ideenlieferanten, die zeigen, wohin sich die Welt derzeit bewegt [5]. Daneben bietet z. B. InfluxData, die Firma hinter InfluxDB, auch erweiterte Softwarelösungen, welche die freie Variante um Enterprise-Features wie Clustering und Support erweitert.

Alle diese freien Werkzeuge haben gemeinsam, dass man sie von Grafana aus abfragen kann. Grafana ist als freies Dashboard-System derzeit das Einzige mit gutem Communitysupport, das nicht nur datenquellenübergreifend leicht bedienbar und flexibel ist, sondern auch

Systeminfrastrukturdiagramme

Je nach Größe des Gesamtsystems empfiehlt es sich, unterschiedliche Systeminfrastrukturdiagramme zu erstellen. Je kleiner das System, vor allem bei der Anzahl der genutzten Schnittstellen, desto sinnvoller ist es, nach klassischer Architektur- und Infrastrukturdokumentation vorzugehen. Aktuelle Trends wie Microservices zeigen: je größer das Gesamtsystem, desto eher ist es sinnvoll, über eine navigierbare Per-System-View (**Abb. 1**) sowohl auf die Infrastruktur als auch die Architektur nachzudenken.

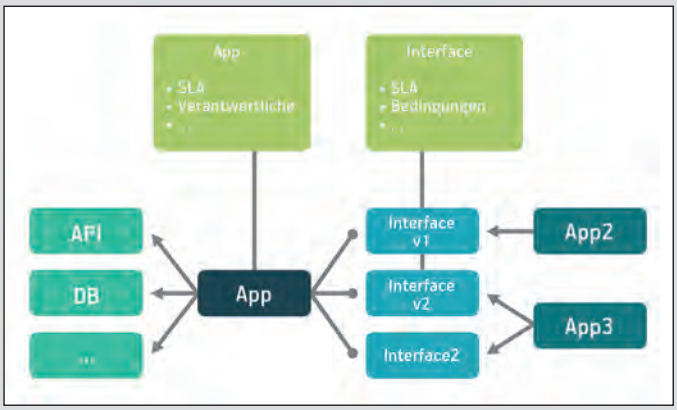

Abb. 1: Systemübersicht mit einer Per-System-View

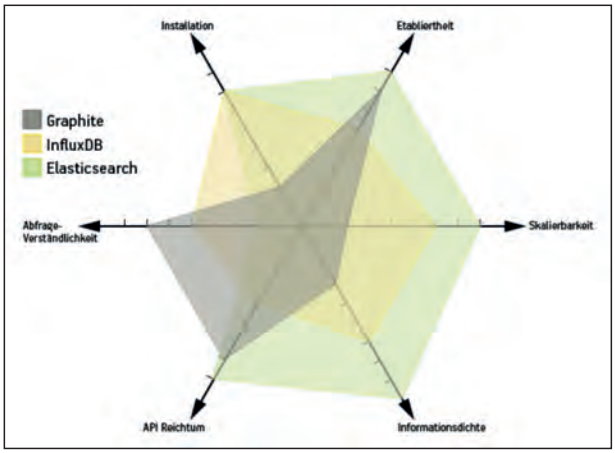

Abb. 2: Unterschiedliche Ausprägungen von Graphite, InfluxDB und Elasticsearch

noch eine schöne Benutzeroberfläche hat. Die schönste Oberfläche hilft jedoch nichts ohne einen Datenstand. Die Messpunkte sollten möglichst aktuell, fehlerfrei und regelmäßig gesammelt werden. Welche Daten man sammelt und wie oft, wird größtenteils von den vorhanden I/O-Ressourcen bzw. deren Kosten begrenzt. Es gibt den Spruch: „Was nicht gemessen wird, verschlechtert sich". Solange noch keine Erfahrungswerte für das konkrete System existieren, sollte möglichst umfassend gesammelt werden. Im Grund sind immer die gleichen voneinander abhängigen Metriken relevant: Latenz, Operationen/Zeit und Durchsatz/Zeit. Die genauen Ausprägungen dieser Metriken sind natürlich stark abhängig von der eingesetzten Software, der Infrastruktur und dem Nutzungsverhalten.

Collect me if you can

Daten werden am besten über den gesamten Stack hinweg gesammelt. Von Durchlaufzeiten einzelner problematischer Methoden innerhalb der Software über die Laufzeitumgebung und Container oder VM hin zur Virtualisierungshardware sowie der Netzwerkinfrastruktur. Wenn es Aggregationspunkte gibt, wie einen SDN- (Software-defined-Networking-)Controller, der Metriken über Netzwerkflüsse schon zuverlässig akkumuliert, sollten diese bevorzugt angebunden werden, um Redundanz zu vermeiden. Wenn eine Provisionierungssoftware, eine Server-Registry oder ein Lizenzmanagementsystem eingesetzt werden, sind das ebenfalls gute Datenquellen, um die Sicht auf das System zu verbessern. Und hier kommen wir wieder zurück auf die Verwendung des Konfigurationsmanagements, das es sehr leicht macht, automatisiert die notwendigen Vorbereitungen für ein geregeltes Abgreifen von Metriken zu treffen. Installiert werden meist lokale Kollektoren, die in definierter Granularität Metriken sammeln und sie an APIs der zentralen Datenbanken senden (**Abb. 3**). Mit den Entwicklern ist abzuklären, wie weit die Applikationen hier unterstützend mitwirken. Ob nun APIs bereitgestellt werden, nur Standardsystemmetriken oder die Metriken gar selbst an die Zeitseriendatenbanken gesendet werden, hängt von den Bedürfnissen ab. Es sollte jedoch ganz klar kommuniziert sein, wie Namensräume zentral definiert sind, um Daten nicht zu vermischen.

Wie an den vorgestellten Zeitseriendatenbanken gut sichtbar wird, können sie bereits einen Teil der Auswertung darstellen. Beispielsweise wird eine Aggregation vorgenommen, wenn die Messung zu lange zurückliegt oder eingehende Daten schon gemittelt wurden. Je mehr Daten zurückgehalten werden und je strukturierter sie sind, desto mehr unterstützen sie die Auswertung.

Es gibt einen lesenswerten Artikel über „The applicability of Graphs for Information Security combatants" von Henrik Johansen [6], der einen von Logs ausgehenden Ansatz beschreibt, sicherheitsrelevante Zusammenhänge in einem Graphenmodell zu speichern. Er beschreibt Sicherheit eher im Sinne von Security als Safety oder Reliability. Aber auch wenn diese Artikelreihe eher die anderen Bedeutungen aufgreift, sollte Security nicht außer Acht gelassen werden.

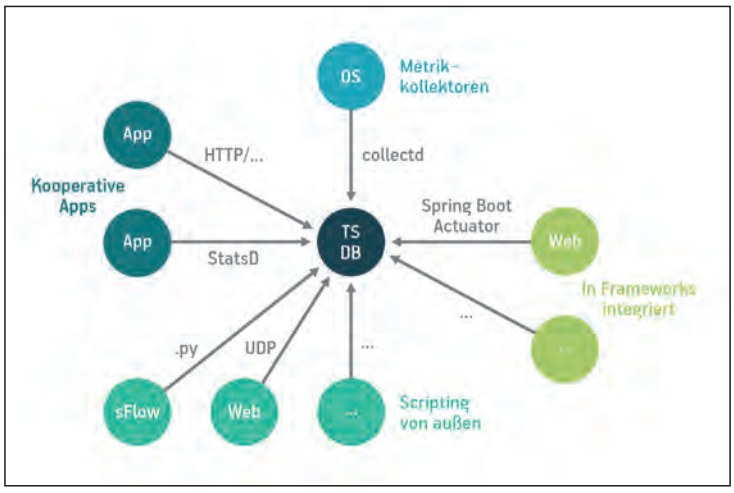

Abb. 3: Beispielhafte Sammlung an möglichen Datenquellen und Datenübertragungsarten

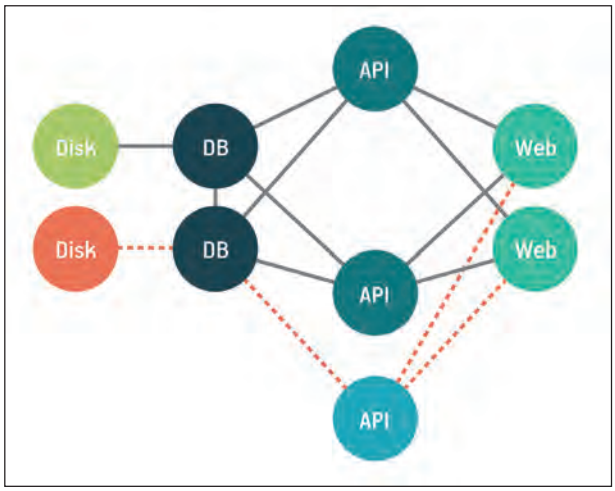

Abb. 4: Ein kritischer Pfad bei Ausfall einer Komponente

Ein Graphenmodell hat noch weitere Vorzüge: Beispielsweise lässt es sich, wie im Kasten zu Systeminfrastrukurdiagrammen erwähnt, zum Analysieren kritischer Pfade verwenden (**Abb. 4**). Die Welt der IT entwickelt sich mehr und mehr hin zu verteilten Services. Wenn eine akkurate Beschreibung der Vernetzung von Applikationen existiert, können Simulationen mit Ausfällen sowohl im Vorfeld als auch im Betrieb getestet und mit graphentheoretischen Ansätzen evaluiert werden.

Durch die Sammlung der Metriken an einer zentralen Stelle werden Korrelationen ermöglicht, die vorher so nicht möglich waren. So können Datenbank-Query-Latenzen einer Applikation mit der Netzwerkinfrastruktur und Applikations-Deployments direkt in Verbindung gebracht werden. Hierbei helfen Visualisierungsprogramme wie Grafana ungemein. Sie ermöglichen neben stabilen Dashboards auch die schnelle Konfiguration neuer Darstellungen, um Probleme oder Auffälligkeiten zu verfolgen.

Wenn genügend historische Daten verfügbar sind, macht das auch Vorhersagen möglich. Graphite bei-

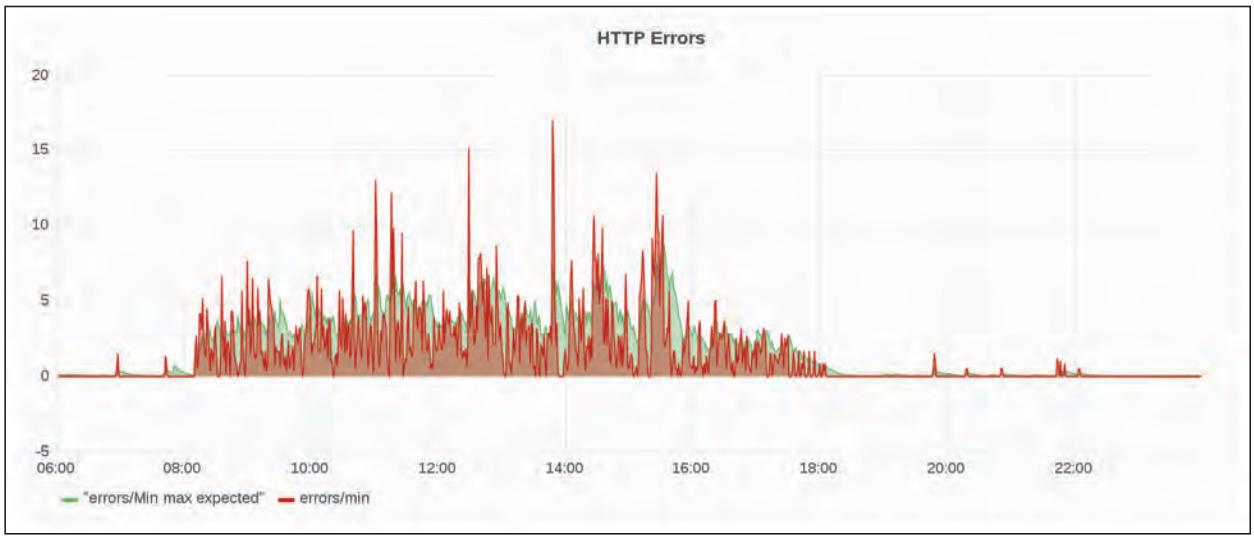

Abb. 5: Vorhersage von HTTP-Fehlermeldungen einer produktiven Applikation

spielsweise hat Methoden zur exponentiellen Glättung (Holt-Winters-Verfahren), mit denen sich Abweichungen der aktuellen Werte von vergangenen darstellen lassen.

```
§§§ Code Box "Graphite: Abweichungen von der Norm"
alias(maxSeries(holtWintersConfidenceBands(scaleToSeconds(
    nonNegativeDerivative(test.errorCount), 60))), "errors/Min max expected")
holtWintersAberration(...)
§§§ End: Code Box
```

Per Scripting würde sich ähnliche Funktionalität auch für InfluxDB bzw. Elasticsearch schaffen lassen. Diese sind aktuell jedoch nicht sofort verfügbar. Aufbauend auf gesammelten und aufbereiteten Statistiken und Vorhersagen lassen sich nun wieder die Überwachungslösungen und das Monitoring bzw. das Reporting erweitern. Das klassische Beispiel ist lineare Regression zur zeitlichen Berechnung wahrscheinlicher Füllstände von Festplatten. Aber auch ohne Vorhersagen hilft uns der Überblick, die Systemlandschaft nicht aus dem Auge zu verlieren. Ohne die Norm des Systems zu kennen, ist es schwierig, Irregularitäten zu erkennen. So gibt es z. B. eine Abhandlung über „Large-scale unusual time series detection“ [7] und ein daraus resultierendes R-Paket. Aber auch auf kleinerer Skala: Wer möchte nicht gerne wissen, ob nach einem Deployment plötzlich die HTTP-Fehlerrate seiner Webapplikation außerhalb der üblichen Rate liegt (**Abb. 5**) oder warum ein Endbenutzer sich beim Support über unzureichende Performance beklagt?

Achtung bei Automatismen

Neben manuellem Kapazitätsmanagement und Fehlererkennung/-analyse können die Auswertungen auch in Policies für das System umgewandelt werden, z. B. für automatische Provisionierung von Applikations-Nodes, um wiederkehrende Lastspitzen abzufangen. Aber Achtung: Je mehr Automatismen existieren, umso besser müssen diese getestet und überwacht werden!

Fazit

Mit diesem Artikel haben wir eine nachhaltige Analyse auf Basis der gesammelten Daten begonnen. Der Fokus der Analyse lag klar auf der Projektunterstützung für langlaufende Projekte, wie man sie häufig in der Industrie oder der Logistik vorfindet. Um in verteilten Umgebungen, wie sie gerade in diesen Branchen durch die Laufzeit von Aufträgen vorkommen, nachhaltig Informationen über den Betrieb analysieren zu können, haben wir geeignete Visualisierungsarten aufgezeigt.

Im kommenden, letzten Teil der Serie werden wir den Kreis schließen und aufzeigen, wie das gesammelte Wissen dabei hilft, in der Entwicklung weitere Schritte zu gehen, ohne dass wir den nun stabilen und sauber analysierten Betrieb gefährden. Seid gespannt, was da kommt!

Joachim Arrasz ist als Software- und Systemarchitekt in Karlsruhe bei der synyx GmbH & Co. KG als Leiter der CodeClinic tätig. Darüber hinaus twittert und bloggt er gerne.

@arrasz **http://blog.synyx.de/**

Jonathan Buch ist als Softwareentwickler in Karlsruhe bei der synyx GmbH & Co. KG tätig und beschäftigt sich dort mit Entwicklung, Systemadministration und CodeClinic-Aufgaben.

Links & Literatur

[1] Das arc42-Template: http://www.arc42.de/template/

[2] Graphite: https://graphiteapp.org/

[3] InfluxData: https://www.influxdata.com

[4] Elastic: https://www.elastic.co/products/elasticsearch

[5] Honeycomb: https://honeycomb.io/

[6] Johansen, Henrik: „The applicability of Graphs for Information Security combatants“: https://medium.com/@henrikjohansen/the-applicability-of-graphs-for-information-security-combatants-2570e879786e#.ah372jme6

[7] R package for detecting unusual time series: http://robjhyndman.com/hyndsight/anomalous/