Architektur im Kontext der Cloud: Patterns und Best Practices ▸ 62

S&S

JAVA Mag

Deutschland €9,80 Österreich €10,80 Schweiz sFr 19,50 Luxemburg €11,15 11.2014

Java magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

Apache DeltaSpike
Portabilität und Community ▸16

Der Rest von REST
HATEOAS unter der Lupe ▸76

Logging

4 194586 709801 11

Robolectric: Do Androids Dream of Electric Sheep? ▸ 107

Boot your own Infrastructure: Spring Boot erweitern ▸ 91

Mehr Sicherheit in Webanwendungen Response-Header ▸ 84

Auswirkungen moderner Architekturansätze auf den Betrieb von Anwendungen

NEUE SERIE

Production-ready?

Reactive Programming, Microservices, Message-oriented Middleware und andere moderne Architekturansätze fordern von Entwicklern heutzutage bei der Auswahl ihrer Tools mehr Weitblick, mehr (Ein-)Blick in die Produktion und deren Anforderungen, als noch vor einigen Jahren. Alle diese Ansätze bringen üblicherweise die Möglichkeit einer besseren Verteilung der gesamten Anwendung mit sich. Dies hat aber beträchtliche Auswirkungen auf den Betrieb von Anwendungen dieser Art.

von Jonathan Buch und Joachim Arrasz

Innerhalb dieser Miniserie wollen wir die hierdurch entstehenden Anforderungen an Entwickler, als auch an die eingesetzten Tools selbst genauer unter die Lupe nehmen. Zum Ende eines jeden Artikels versuchen wir uns an einer möglichst objektiven Bewertung der Werkzeuge im jeweiligen Einsatzszenario.

Da aber viele miteinander interagierende Tools vorgestellt werden, müssen wir uns auf eine Beschreibung und eine Bewertung beschränken und wollen mit den Artikeln das Zusammenspiel und die Funktionsweise beschreiben (**Abb. 1**). Für konkretere Einblicke in die Tools wollen wir gern auf speziellere Artikel verweisen, vielleicht können wir ja die Neugier wecken. Einige Leser werden nun denken „Warte, dafür haben wir doch DevOps, was interessiert mich das", aber warten wir den Verlauf der Serie ab. Und wer hier den nächsten Vergleich schwergewichtiger Application Server und deren Leistungsvermögen erwartet, den müssen wir leider enttäuschen und auf ältere Artikel verweisen.

Weiterhin wollen wir unsere Leser noch darauf aufmerksam machen, dass wir die komplette Serie auf www.JAXenter.de begleiten werden und auch für Diskussionen rund um das Thema zur Verfügung stehen.

Beginnen wir nun im ersten Teil mit den Werkzeugen zum Logging aus Anwendungen, der Konfiguration von Logging sowie der Kumulierung von Logs über verschiedene Dienste, (Micro-)Services oder auch Server hinweg. Warum beginnen wir ausgerechnet mit Logging?

Logging ist wohl derjenige Teil der Implementierung, bei dem am offensichtlichsten eine Verbindung zwischen Entwicklern und Betreibern besteht. Üblicherweise versuchen Teamplaner dies heute mit dem Buzzword „DevOps" zu erschlagen. DevOps in Cross-functional Teams sollen die Verbindung zwischen Entwicklung und Betrieb stabilisieren. Sie sollen den Gedanken des Betriebs und der betriebsvorbereitenden Maßnahmen, auf die wir später detailliert eingehen werden, in die Entwicklerteams tragen, dort mitarbeiten und Betriebsgrundlagen manifestieren.

Leider ist es aber heutzutage immer noch so, dass ausgerechnet die uralte Disziplin des Loggings den Betrieb oft am wenigsten unterstützt. Folgende Probleme haben wir in den letzten Jahren immer und immer wieder aus verschiedensten Softwareteams gehört:

- Keine aussagekräftigen Inhalte in den Logs
- Viel zu groß, man findet die Information im Grundrauschen nicht
- Nicht sortierbar
- Nicht effizient indizierbar und somit
- Nicht effizient durchsuchbar
- Nicht oder kaum statistisch verwertbar

Aus unserer Erfahrung heraus ist es daher besonders wichtig, dass bereits in der Phase des Ramp-ups einer Anwendung abgeklärt ist, welche Anforderungen der Betrieb an Anwendungslogfiles stellt. Schließlich müssen gerade die Kollegen vom Betrieb wissen, was die Anwendung so anstellt. Wichtiger ist aber, dass Logfiles schnell und effizient auswertbar sind. Welche Voraussetzungen Entwickler hierfür einhalten müssen, werden wir später im Artikel noch detaillierter besprechen.

Aber was ist eigentlich an klassischem Logging so schwer? Das machen wir Entwickler doch schon von Beginn des Softwarezeitalters an? Nun ja, fragt man dies einen althergebrachten Systemadministrator, wird er wohl so oder so ähnlich antworten: „*Gute Logfiles konnten Entwickler noch nie. **WIR** sorgen immer im Nachgang dafür, dass die Logfiles wenigstens ein klein wenig Aussage und so etwas wie einen vollständigen Timestamp haben. Oft bekommen wir Anwendungsarchive über den Zaun geworfen, in denen das Logging noch auf Debug-Level steht. 3 Megabyte Logeinträge pro Sekunde sind keine Seltenheit.*"

Was also macht ein gutes, produktionsunterstützendes Anwendung-Logging aus?

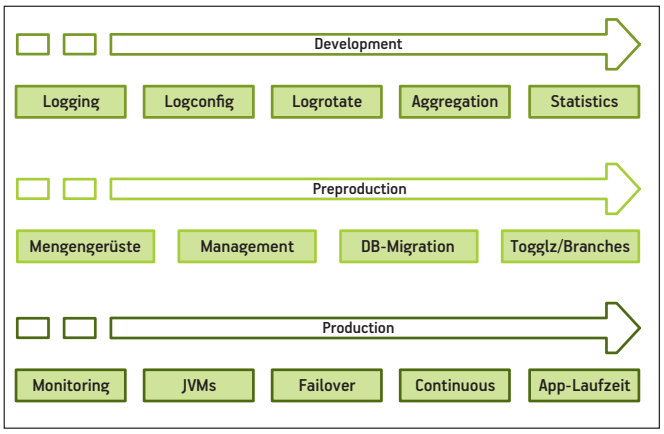

Abb. 1: Reise durch die Entstehung (das Development), über die Stabilisierungsphase der Preproduktion, bis hin zur Produktion

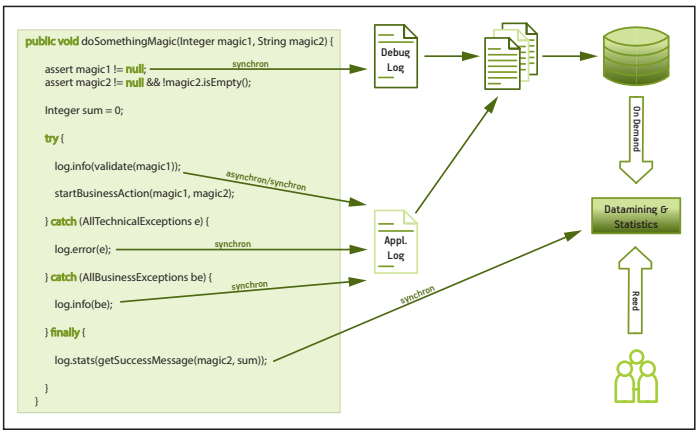

Abb. 2: Beispielhaftes Listing

- Loglevel im Betrieb anpassbar (darauf gehen wir in einem anderen Artikel der Serie detaillierter ein).
- Trennung technischer Logeinträge (z. B. gefangene Exceptions) von fachlichen Informationen (z. B. fehlende Möglichkeit, ein Dokument zu drucken, da noch Informationen im Dokument fehlen). Weiterhin sollte man Informationen, die einen rein statistischen oder auditierenden Inhalt haben, ebenso separat loggen. Der Anwendungsentwickler muss sich dessen bewusst sein.
- Um den Kollegen des Betriebs ein schnell und effizient auswertbares Log bereitzustellen, sollte der Level eines Logeintrags sehr sorgfältig gewählt werden. Eine gefangene und behandelte fachliche Exception ist sicherlich für den Betrieb kein Error-Level.
- Damit das Logging keinen Einfluss auf die Anwendung selbst hat, sei es Performance oder Stabilität betreffend, sollte man auch hier Appender mit Bedacht auswählen, die dementsprechend arbeiten. Logeinträge, die Statistiken füttern, müssen nicht in allen Fällen zwingend ankommen und verarbeitet werden, hier reicht eventuell ein „Fire and Forget“-Ansatz. Ein Error-Log jedoch sollte unter allen Umständen sauber geschrieben werden.
- Um sicherzustellen, dass der Betrieb eine Reihenfolge von technischen Problemen erkennen kann, ist es unabdingbar, dass Logeinträge mit einer einheitlichen, idealerweise übergreifend bereitgestellten Syntax ausgestattet sind, die Timestamps und Location beinhalten sollten. Wenn möglich, sind auch die Logmeldungen einheitlich in einer Sprache und idealerweise zur Verständigung in Englisch.
- Zu guter Letzt ist es sehr praktisch, wenn man in der Lage ist, Loglevels, Appender und alle weiteren relevanten Spieler zur Laufzeit anzupassen. Hier sollte man beachten, dass eine Anwendung eventuell in einem Clusterverbund läuft und man nur einen einzelnen Knoten im Cluster in einen Debug-Zustand versetzen können möchte. Hierbei muss beachtet werden, dass die Appender dieses auch unterstützen.

Listing 1

```
Schlecht:  05:23:14 INFO update on Entity user
besser:    2014-01-04T14:00:39,664+0200 INFO [d.f.t.UserService:22]
                                    [7f529ac-...] Updating user 66579
```

Hält sich ein Team an die genannten Punkte und diskutiert die genannten Probleme, so sollte von Beginn an der Informationsgehalt in den Logs stimmen, und sie sollten ordentlich lesbar, verfolgbar und auswertbar sein.

Auf die verschiedenen Logframeworks gehen wir in diesem Artikel nicht näher ein. In diesem Heftschwerpunkt finden Sie aber weitere Beiträge, die sich mit einigen Frameworks genauer beschäftigen.

Abbildung 2 zeigt ein beispielhaftes Listing, das die verschiedenen Loganforderungen aufzeigt. Man erkennt den Unterschied zwischen technischem und fachlichem Logging sowie das spezielle Logging für die Statistik. Ebenso erkennbar sind die Anforderungen an die verschiedenen Logs, manche Einträge müssen immer im Logfile auftauchen (Error), manche nur zur Detailbetrachtung (Info), und manche können sogar zeitlich flexibel geschrieben werden (Validation).

Kommen wir nun also dazu, welche Themen man für eine saubere Logkonfiguration im Auge behalten muss. Logkonfiguration ist ein Thema, das oberflächlich betrachtet tausend Mal behandelt wurde. Warum greifen wir das also nun hier doch wieder auf?

Die aktuellen Veränderungen im Aufbau von Anwendungen machen es nötig, hier einen genaueren Blick darauf zu werfen. Zuerst einmal können unbedachte Logkonfigurationen sehr einfach das I/O-Subsystem einer Architektur lahmlegen, zum anderen kann man hier mit ein paar Kniffen sehr ordentlich lesbare und somit gute Logfiles vorbereiten.

Es gibt nichts Schlimmeres, als einen wütenden Administrator vor sich stehen zu haben, der sich während eines Systemausfalls zehn Minuten durch verschiedene Logfiles wühlen musste, bis er erkannt hat, dass lediglich ein Lock auf der Datenbank die Probleme verursachte.

Ein weiteres Problem, das ein Entwicklerteam wunderbar auf die Betriebsmannschaft oder den DevOp in ihrem Team abwälzen kann, ist die Verantwortung für ein sinnvoll lesbares und strukturiertes Log. Um Tools verwenden zu können, die der Betriebsmannschaft ein effizient und schnell durchsuchbares Log bereitstellen, muss zuerst einmal das Entwicklerteam darüber nachdenken, welche Identifier wo in einem Logeintrag stehen soll(t)en, damit die Betriebsmannschaft schnelle Suchen und somit schnelles Incident-Management zur Verfügung stellen kann.

Natürlich bietet jedes der Werkzeuge zur Kumulierung unterschiedliche Vor- und Nachteile. Je nach Einsatzszenario sollte man hier genau abwägen, ob man eine steile Lernkurve gegenüber einer sehr detaillierten Suchfunktion in Kauf nehmen möchte, oder nicht.

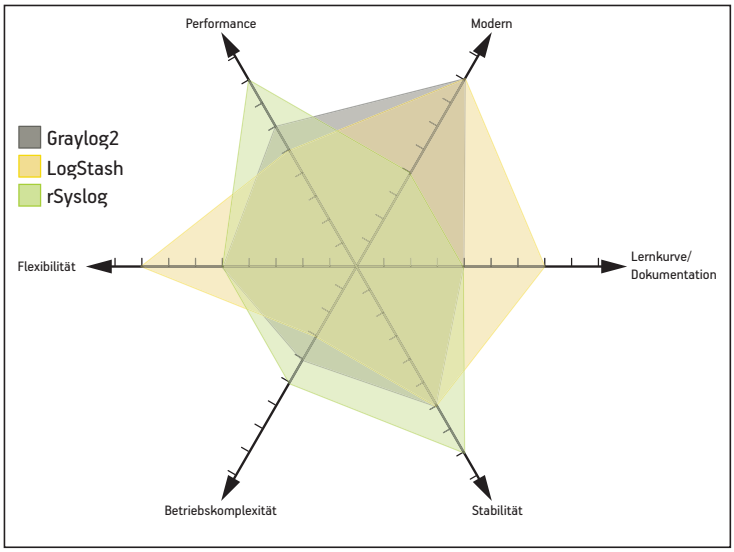

Abb. 3: Vor- und Nachteile der Log-Handler Graylog2, Logstash und rsyslog

Die Laufzeit eines Softwareprojekts spielt sicherlich auch in die Auswahl hinein. Eine steile Lernkurve und fein granulare Suchfunktion ist vielleicht nicht die beste Wahl für eine dynamische Webseitenentwicklung, die nur zur Fußballweltmeisterschaft in Brasilien gebraucht wird, ergibt aber für eine prozesssteuernde Software durchaus Sinn, um sie die nächsten fünfzehn Jahre weiter am Leben erhalten zu können.

Im Detail wollen wir nun auf die Vor- und Nachteile von drei freien Vertretern von Log-Handlern mit der Möglichkeit zentraler Datenhaltung eingehen: Graylog2, Logstash und rsyslog (**Abb. 3**). Diese Auswahl inklusive Bewertung ist durchaus subjektiv, soll aber

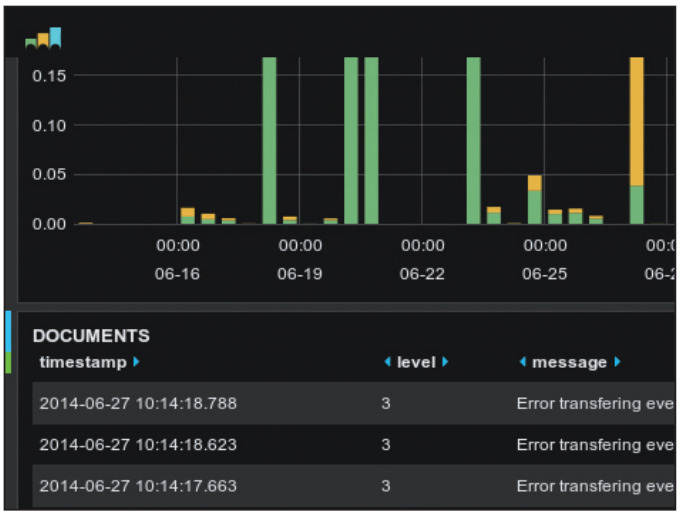

Abb. 5: Kibana

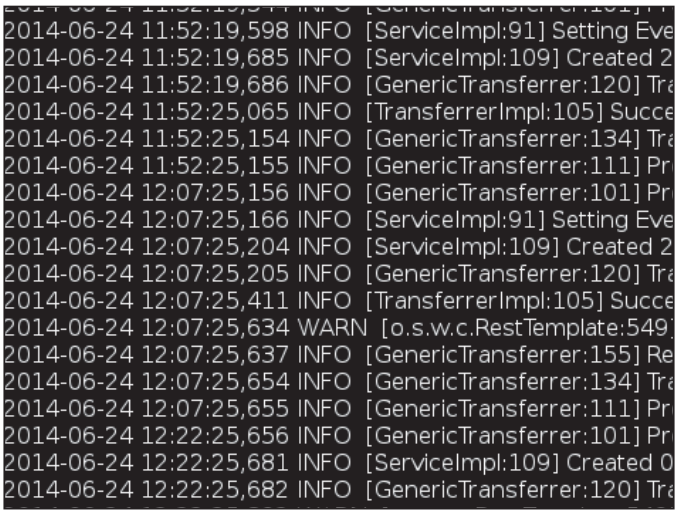

Abb. 6: rsyslog

die unterschiedlichen Logging-Ansätze aufzeigen und visualisieren.

Graylog2 wurde 2010 von Lennart Koopmann begonnen, und kommerzieller Support kann durch den Schirmherr TORCH GmbH erhalten werden (**Abb. 4**). Graylog2 wurde und wird sehr aktiv unter Einbeziehung der entstandenen Community weiterentwickelt. Es stellt eine umfassende Logging-Lösung bereit, die mit Elasticsearch zur Speicherung und einer Weboberfläche zur Analyse der gesammelten Daten punktet.

In einem ähnlichen Zeitrahmen ist auch Logstash entstanden – ab 2009 wurde dieses von Jordan Sissel und Pete Fritchman entwickelt, wobei auch hier die Entwicklung stetig fortschreitet. Durch die Nähe von Logstash zu Elasticsearch und Kibana geschieht dies seit 2013 im Rahmen von Elasticsearch (**Abb. 5**). Logstash ist im Gegensatz zu Graylog2 keine fertige Lösung, sondern eher ein flexibler Baukasten, um eine passende Lösung in Verbindung mit anderen Projekten herzustellen. Auf Kibana wird Tammo van Lessen ab Seite 48 genauer eingehen.

rsyslog ist eine syslog-Alternative von Rainer Gerhards, gesponsert von dessen Firma Adiscon GmbH (**Abb. 6**). Dieses Produkt ist mit Entstehungsjahr 2004 ein wenig älter als die vorherigen, wird jedoch ebenfalls noch aktiv weiterentwickelt. Durch seine Historie als syslog-Nachfolger ist rsyslog viel tiefer im System verankert und eher Systemadministratoren als Entwicklern geläufig. Durch ein Modulsystem ist es jedoch ähnlich wie Logstash eine flexible Anwendung, um Logs zu verarbeiten.

Ein Vergleich dieser drei leicht unterschiedlichen Lösungen ergibt auf jeden Fall Sinn – eine Bewertung ist jedoch sehr schwierig und stark abhängig von Faktoren des Projekts und der allgemeinen Umgebung.

In den Einleitungen der drei vorgestellten Programme wurde schon auf einen der größten Unterschiede

Cross-functional Teams

Unter einem Cross-functional Team versteht man ein Team, das aus Mitgliedern verschiedener Rollen besteht. Im Team sind Betreiber genauso wie Entwickler als auch UX-Experten integriert. Das Team löst im Konsens die anstehenden Aufgaben. Die Spezialisten im Team beraten anhand der kommenden Aufgaben, entschieden wird jedoch im Konsens.
Ein Cross-functional Team besteht oft aus:

- Juniorentwicklern
- Seniorentwicklern
- Architekten oder architekturerfahrenen Entwicklern
- UX Developer (wenn benötigt)
- DevOps/SysOps

Um dieses arbeitende Team herum braucht es, je nach Prozess, in dem gearbeitet wird, natürlich auch noch Mitarbeiter, welche die Kommunikation in die Fachabteilungen steuern und organisieren. Dies ignorieren wir an dieser Stelle eiskalt, da es den Rahmen des Artikels ansonsten sprengen würde.

Tipp

Fachprozesse mit einer eindeutigen Prozess-ID ausstatten und im Logeintrag immer exakt an der gleichen Stelle oder immer sehr eindeutig durch einen regulären Ausdruck auffindbar bereitstellen. Somit sind Werkzeuge wie Elasticsearch oder dergleichen sehr effizient in der Lage, die Einträge durchsuchbar zu machen und strukturiert bereitzustellen.
Voraussetzung hierfür ist natürlich, dass die Prozess-ID immer eindeutig ist, auch über verteilte Applikationskontexte hinweg (siehe beispielsweise das Stichwort MDC (=Mapped Diagnostic Contexts) im SLF4j-Umfeld).

Die Speicherung von Logs in Plaintext-Dateien wird wohl auch in Jahrzehnten noch ohne Spezialwerkzeug möglich sein – Graylog2 ist für Elasticsearch als Speicher-Backend geschaffen worden.

eingegangen: Wie „fertig“ bzw. flexibel muss die Lösung sein, wie kann sie sich in existierende Infrastruktur integrieren, wie viel Komfort ist out of the box vorhanden? Während zum Beispiel bei Graylog2 schon eine Rechteverwaltung für Benutzer und Zuordnung zu Hosts integriert ist, muss dieses bei den anderen Varianten im Nachhinein eventuell noch nachgebaut werden. Im Gegenzug sind jedoch die Ein- bzw. Ausgabeformate bei den beiden anderen Produkten viel flexibler und erlauben somit eine Anpassung an Infrastruktur, die man möglicherweise nicht komplett unter Kontrolle hat.

Struktur bringt hier besonders Graylog2 mit dem Graylog Extended Log Format (GELF). Um Nachteile eines syslog-Formats zu umgehen, das nur einzeilig, limitiert von der Größe und unstrukturiert ist, wurde für rsyslog ein strukturiertes Textformat auf Basis von JSON spezifiziert. Auf der anderen Seite hält sich rsyslog prinzipiell durch seine Vergangenheit an syslog, während Logstash mit „grok“ einen kompletten Log-Parser bereitstellt, der beliebig zwischen Formaten konvertiert.

Eine Anforderung kann der Ressourcenverbrauch während der Logsammlung auf dem Quellhost sein. Wenn die zu überwachende Applikation selbst für die Versendung der Logs verantwortlich ist, hält sich der Verbrauch in Grenzen – ein zusätzlich gestarteter Logstash Daemon zur Logsammlung aus einer Datei zieht jedoch eine JVM-Laufzeitumgebung mit JRuby nach sich und ist hierdurch schwergewichtiger als ein

rsyslog Daemon, der sowieso gestartet ist. Graylog2 beschränkt sich ganz auf die Serverseite.

Wie schon angesprochen, hat auch die Projektlaufzeit bzw. die konkreten Kompetenzen der Teammitglieder Einfluss auf die Auswahl. Komplexität des Stacks kann hier gegenübergestellt werden. Die Speicherung von Logs in Plaintext-Dateien wird wohl auch in Jahrzehnten noch ohne Spezialwerkzeug möglich sein – Graylog2 ist für Elasticsearch als Speicher-Backend geschaffen worden.

Weitere benötigte „bewegliche Teile“ bei Graylog2 sind die grafische Oberfläche (derzeit eine Play-Webapplikation), eine MongoDB zur Konfiguration, die Backend-Applikation zum Logempfang bzw. zur -verarbeitung und Elasticsearch. Logstash ist hier weniger vielfältig und besteht aus einer einzigen Applikation, die unterschiedlich konfiguriert wird – die Komplexität ist hier also wählbar.

Ein Beispiel wäre eine Umgebung mit Elasticsearch und Kibana – eine JavaScript-Webanwendung zur Visualisierung von Daten in Elasticsearch – und eine/mehrere Logstash-Instanzen zur Speicherung der Logs. Auch rsyslog lässt sich für Elasticsearch konfigurieren, üblicher ist aber in diesem Umfeld die simple Speicherung von syslogs in rollenden Logs im Dateisystem.

Fazit

In diesem Teil der Serie wollten wir aufzeigen, wie wichtig eine enge Kooperation zwischen Entwicklern und Betreibern, besser ein Cross-functional Team (siehe gleichnamiger Kasten), für den Start einer Softwareentwicklung ist. Es gibt viele kleine Fallstricke, aber auch Chancen, um das gesamte Projekt effizienter und besser zu machen, die man von vorne herein im Auge behalten sollte.

Ein Entwicklerteam, das von Beginn an mit solchen Anforderungen konfrontiert ist, macht sich komplett andere und deutlich mehr Gedanken über ihre Art, Logging zu entwickeln. Wir hoffen, wir konnten die richtigen Denkanstöße und Ansätze aufzeigen.

Im kommenden Teil der Serie gehen wir detaillierter auf die konkreten Anforderungen an die Phase der Preproduktion ein. Auch hier ist eine enge Zusammenarbeit zwischen Entwicklern und Betreibern eminent wichtig, Verständnis für das Zusammenspiel des Stacks muss gegeben sein, ansonsten werden die Entwickler auch hier in der Lage sein, viele unnötige Fallstricke für den Betrieb aufzubauen.

Ausblick

In der nächsten Ausgabe werden wir uns intensiver dem Thema Produktionsvorbereitung und -stabilisierung widmen. Wie können wir unser Set-up gegen Ausfälle, Grundlast beziehungsweise Lastspitzen stabilisieren und stählen?

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.

@bujo84

Links & Literatur

[1] http://pragprog.com/magazines/2011-12/justintime-logging

[2] http://logstash.net/

[3] http://graylog2.org/

[4] http://www.rsyslog.com/

[5] „Release It!: Design and Deploy Production-Ready Software“ by Michael T. Nygard

JAVA³

Jetzt abonnieren!
www.javamagazin.de

Jetzt 3 TOP-VORTEILE sichern!

1

- Alle Printausgaben frei Haus erhalten
- Intellibook-ID kostenlos anfordern (www.intellibook.de)

2 Abo-Nr. (aus Rechnung oder Auftragsbestätigung) eingeben

3 Zugriff auf das komplette PDF-Archiv mit der Intellibook-ID

S&S Media Group

www.javamagazin.de