DVD auf S.3

Jahresarchiv, Bonusvideos und Buchauszüge auf einer DVD!

S&S

Deutschland €9,80 Österreich €10,80 Schweiz sFr 19,50 Luxemburg €11,15 1.2015

JAVA Mag

Java™ magazin

Java • Architekturen • Web

1500 SEITEN EXPERTEN-WISSEN

Über 200 Artikel aus dem Java Magazin und Business Technology Magazin 2014

Noch mehr Bonusmaterial auf DVD VIDEOS und BUCHAUSZÜGE

4 194586 709801 01

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

Teil 3: Production!

Production-ready?

Nachdem wir im ersten Teil der Serie darauf abzielten, den Entwicklern aufzuzeigen, wie sie dem Betrieb das Leben einfacher machen können, und im zweiten Teil die Vorbereitungen des Betriebs besprochen wurden, wollen wir uns in diesem letzten Teil der Miniserie dem Thema Produktion widmen. Wie können die gesammelten Produktionsdaten der Entwicklung weiterhelfen?

von Sascha Rüssel und Joachim Arrasz

Unser beispielhaftes Projektteam hat es geschafft: Die Software ist in Betrieb und die ersten Hürden sind genommen. Das Team verfolgt die ersten Tage und Wochen der Produktion des neuen Betriebs (**Abb. 1, 2**). Statistische Auswertungen der gesammelten Logs sollten eventuelle Probleme rechtzeitig erkennen lassen, und die Betriebsmannschaft kann somit auf Basis dieser Auswertungen die ersten Verbesserungen vornehmen.

Doch wie geht das eigentlich? Wie kann die Betriebsmannschaft nun aus den Statistiklogs einen Mehrwert an Informationen für die Weiterentwicklung gewinnen? Um frühzeitig programmatisch oder infrastrukturell agieren zu können und somit den Betrieb der Anwendung auch in Zukunft zu gewährleisten, sollte man den aktuellen Zustand der Anwendung und sich abzeichnende Trends leicht erkennen können. Es ist an der Zeit, die Software und die benötigten Komponenten zu überwachen. Doch was will man eigentlich überwachen?

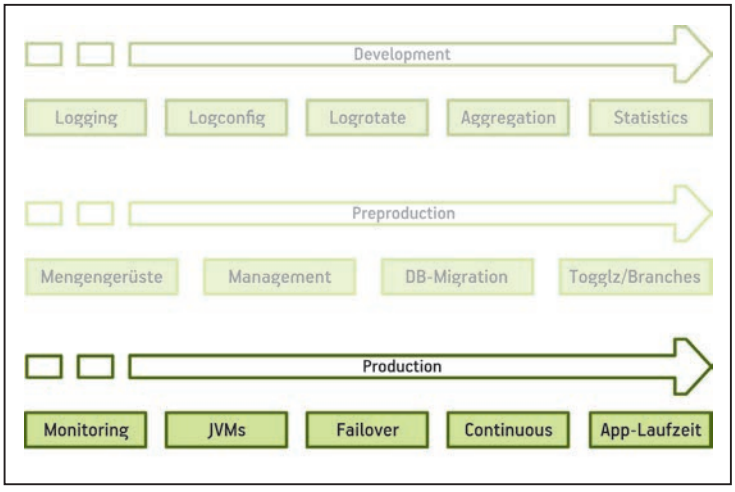

Abb. 1: Reise durch die Entstehung (das Development) über die Stabilisierungsphase der Präproduktion bis hin zur Produktion

1. Hardware und Netzinfrastruktur
2. Abhängigkeiten in der Infrastruktur
3. Die Laufzeitumgebung der Anwendung(en)
4. Die Funktionalität der Anwendung(en)

Zu allererst sind die rudimentärste Netz- und Hardwareinfrastruktur sowie die Grundfunktionalität des Betriebssystems zu überwachen. Dies ist für gewöhnlich Aufgabe des klassischen Betriebs und stark abhängig davon, in welcher Hostingumgebung die Anwendung läuft und welche Techniken zum Managen der Umgebung genutzt werden.

Die Laufzeitumgebung einer Anwendung ist genauso zu überwachen wie die darunterliegende Hardware. Nahezu jede Laufzeitumgebung bietet Möglichkeiten des Managements. Mit JMX stellt Java hier eine seit vielen Jahren etablierte, erfolgreich einsetzbare und standardisierte Option zur Verfügung. Innerhalb der Laufzeitumgebung ist es wichtig, alle von eben dieser Laufzeitumgebung bereitgestellten Ressourcen zu überwachen. Hierzu können gehören:

- Speicher
- Datenquellen
- Transaktionsmanagement
- E-Mail- und Messaging-Verbindungen
- Konnektoren an native Anwendungsteile, Legacy-Systeme oder Produktionsmaschinen
- Caches
- Verschiedene Parameter der Laufzeitumgebung selbst (-Xmx -Xms usw., je nach eingesetzter Java-Version)

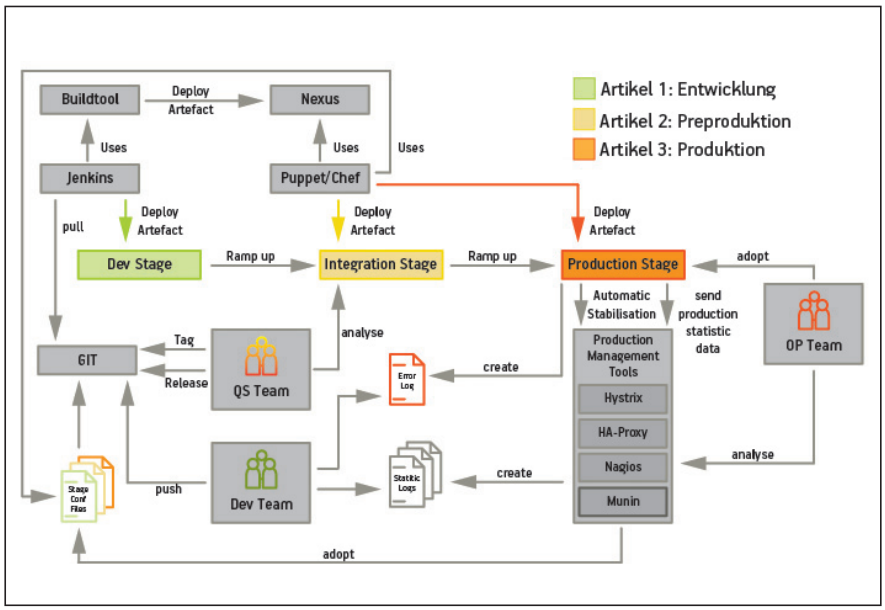

Abb. 2: Artikelübersicht

Durch unsere statistischen Auswertungen können hier über die Zeit immer exaktere Vorwarnstufen etabliert werden. Der Betrieb kann also deutlich früher auf Ungereimtheiten oder Lastspitzen, ausgelöst durch andere Effekte (Marketing ist hier ein Stichwort), reagieren. Man kann ein proaktives Management einführen, das bereits reagiert, sobald die ersten Böen durch die Stadt fegen, und nicht wartet, bis der Sturm mitten in der Stadt wütet.

Über Werkzeuge wie CRaSH, das allseits beliebte Jolokia oder aber jmx4perl sind wir nun in der Lage, die Daten und Methoden, die die Anwendung über JMX für uns bereitgestellt hat, zu interpretieren und darauf zu reagieren. Man kann beispielsweise neue Datenbankverbindungen bereitstellen, bevor die freien, nutzbaren Verbindungen ausgehen und die Anwendung in eine brenzlige Situation gerät. Eine andere Möglichkeit wäre es beispielsweise, Cache-Daten zu invalidieren, um den Cache neu befüllen zu lassen.

Wir wissen nun, *was* wir überwachen. Beleuchten wir als Nächstes das *Wie* näher. Monitoring besteht im Wesentlichen aus:

Tipp

Das Java-Management-Extensions-(JMX-) API stellt einen standardisierten Mechanismus zum Verwalten von Anwendungen zur Verfügung. Man kann durch Einsatz von ein paar Annotationen Managementoberflächen erzeugen und interaktiv machen. Nahezu alle aktuellen Container oder Frameworks unterstützen JMX auf die eine oder andere Art. Bei Spring Boot beispielsweise kann man mittels der Annotationen

@ManagedResource
@ManagedAttribute
@ManagedOperation

einfach Managementoberflächen für Klassen, die mit Obigem annotiert sind, bereitstellen. Diese sind dann entweder per HTTP oder aber Tools wie jConsole nutzbar. Natürlich sind auch die aktuellen Container wie ein JBoss in der Lage, JMX-Management out of the box zu unterstützen und können selbst darüber verwaltet werden. JBoss selbst bietet folgende Annotationen an.

@Service (objectName="demoJMX:service =demoManagement")
@Management(DemoManagement.class)

Nutzt man weder das eine noch das andere, ist man mit Werkzeugen wie Dropwizards „Metrics“ sehr gut beraten. Will man keine Annotationen nutzen, gibt es noch die programmatische Möglichkeit, über das Interface *MBeansServer* und eine Naming-Convention JMX zu implementieren.

Anzahl Prozesse	OK	PROCS OK: 80 processes
Ping	OK	PING OK - Packet loss = 0%, RTA = 0.43 m
SSH port	OK	SSH OK - OpenSSH_5.3p1 Debian-3ubunt
Zombies	OK	PROCS OK: 0 processes with STATE = Z
free disk space	OK	DISK OK - free space: / 58330 MB (94% inc
load	OK	OK - load average: 0.02, 0.08, 0.08
users logged in	OK	USERS OK - 0 users currently logged in

Abb. 3: Icinga

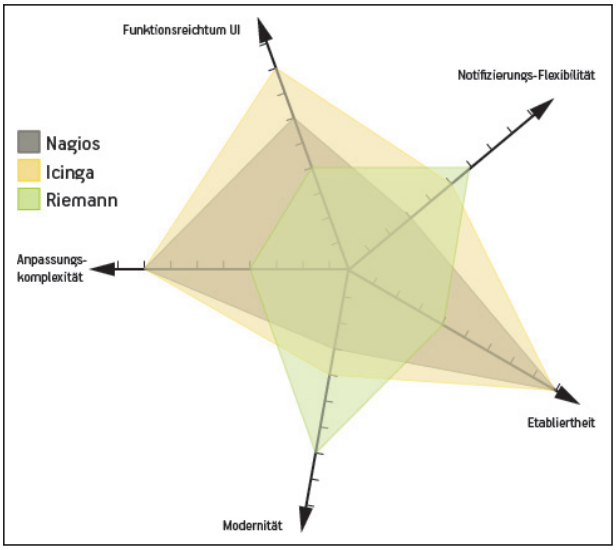

Abb. 4: Bewertung der Serviceprüfung

- Erfassung von Daten
- Speicherung der erfassten Daten
- Auswertung der gesammelten Daten
- Visualisierung der aufbereiteten Daten

Dies kann nur als sehr grobe Einteilung dienen, da nicht jeder Anwendungsfall jede dieser Funkionen unterstützen muss. Wenn der augenblickliche Zustand des Systems wichtiger ist als es wäre, die konkreten Werte zu historisieren, fallen die Speicherung und Auswertung weg. Ein Beispiel hierzu wäre eine geclusterte und verteilte Anwendung, die einen Einblick in das System – über ein Hystrix-Dashboard – bietet. Wenn es nicht wichtig ist, wie viele meiner Anwendungen laufen, sondern nur, dass es genügend sind, kann dies auch meine Überwachung widerspiegeln.

TIPP

Grundsätzlich gilt: Je mehr aktive Checks in einem Ausführungsintervall ausgeführt werden, desto leistungsfähiger muss die Hardware des Monitoring-Systems sein. Um Last auf dem Monitoring-System einzusparen, können mehr passive Checks eingesetzt werden.
Jedoch ist zu bedenken: Je mehr Daten anfallen, umso größer ist die I/O-Last, die durch geeignetes Caching oder Skalierung des Systems behandelt werden muss.

Das Sammeln von Daten ist grundsätzlich auf zwei verschiedene Arten möglich: **aktiv** und **passiv**. Aktiv bedeutet: Vom Monitoring-System aus wird ein Script (Bash, Python, Perl,…) auf der zu überwachenden Maschine ausgeführt oder man fragt einen Dienst nach dem Zustand eines Service.

Ein Service kann in diesem Zusammenhang einen realen, auf einem Host laufenden Service (wie DNS, SMTP, NTP etc.) oder aber auch eine andere Art von Messung, wie z. B. belegter Arbeitsspeicher, Bandbreitennutzung oder Anzahl laufender Prozesse, darstellen. Passiv bedeutet: Das Monitoring-System bekommt von einer externen Anwendung in einem definierten Format Ergebnisse geliefert.

Passive Checks haben den Vorteil, dass sie das Monitoring-System nicht zusätzlich mit der Ausführung von Tests belasten und dass auch lang laufende Tests asynchron getestet werden können. Für den letzteren Fall, das Abfragen langlaufender Tests, ist es auch möglich, einen aktiven Check auf gecachte Werte zugreifen zu lassen, wobei diese Werte von einem anderen Prozess angelegt bzw. ausgeliefert werden müssen.

Das birgt aber auch die Gefahr, dass die abgerufenen Ergebnisse nicht mehr der Realität entsprechen. Darum ist zu gewährleisten, dass die gecachten Werte mit einem Zeitstempel versehen sind und eine Bewertung des Monitoring-Systems, ob die Aktualität der gecachten Daten zufriedenstellend ist, verfügbar ist.

Das Auswerten der gesammelten Daten beginnt bereits mit einer Bewertung der Funktionalität des Service anhand von angegebenen Schwellwerten oder zu erwartenden Ergebnissen. Der Zustand des Service wird klassifiziert, um differenziert auf diese Prüfungen reagieren zu können.

In der Regel haben die verschiedenen Monitoring-Lösungen bereits ein Webinterface, das auf fehlgeschlagene Tests hinweist und einen groben Überblick liefert.

Aufgrund der Klassifizierung kann man nun verschiedene Reaktionen einleiten. Der Standard ist es, erst einmal die Verantwortlichen via E-Mail oder Pager zu benachrichtigen, um bei kritischen Events eine Handlung einzufordern. Darüber hinaus können auch weitere automatische Reaktionen via Action Handler veranlasst werden, etwa ein automatischer Restart des kritischen Service oder Aktionen mit JMX.

Es gibt Unmengen an Werkzeugen, die das Monitoring betreffen. Sie sind so unterschiedlich wie die Projekte, in denen sie eingesetzt werden. Nagios ist sicherlich einer der bekanntesten Vertreter. Aber auch Namen wie Icinga, Hystrix, Yammer, Riemann, Munin, Ganglia, Graphite, Jolokia und collectl finden sich hier wieder. Um den Überblick nicht zu verlieren, stellen wir hier kurz drei recht unterschiedliche Monitoring-Lösungen vor.

Der Platzhirsch Nagios und in jüngster Zeit auch dessen Fork Icinga beherrschen noch immer den Markt.

Um tatsächlich Trends, Spitzen und Anomalien zu erkennen, empfiehlt es sich, die Performancedaten aufzubereiten und Graphen zu generieren. Ein Mittel wären Nagios-Plug-ins.

Icinga bietet gegenüber Nagios den Vorteil, mehrere Prüfungen parallel ausführen zu können, und bringt eine zeitgemäßere Weboberfläche mit (**Abb. 3**).

Die meisten Checks finden via SSH oder über NRPE, einen Daemon, der auf dem zu überwachenden Host läuft und die Checks vorkonfiguriert vorliegen hat, statt. Andere Tests, wie die reine Erreichbarkeit des Hosts und netzwerkfähige Services, sind auch lokal ausführbar. Die Bewertung der Serviceprüfung geschieht durch an das Monitoring-System zurückgegebene Rückgabewerte, wobei diese dann direkt den Zustand des Service widerspiegeln (**Abb. 4**).

Darüber hinaus geben Servicechecks einen (möglichen) Grund für einen Status ungleich OK an und geben Performancedaten (gemessene Werte bzw. das erhaltene Ergebnis) zurück.

Um tatsächlich Trends, Spitzen und Anomalien zu erkennen, empfiehlt es sich, die Performancedaten aufzubereiten und Graphen zu generieren. Ein Mittel hierfür wären Nagios-Plug-ins wie Nagiosgraph oder Graphios.

Eine andere Monitoring-Lösung wäre Riemann, die eine Eventverarbeitung für vorrangig verteilte Systeme bietet. Die Events werden dabei vom System in Form von Protocol-Buffern empfangen. Ein Event enthält in seiner Struktur den Host, von dem das Event stammt, die Servicebeschreibung, einen Status sowie weitere für die Klassifizierung sinnvolle Merkmale. Mit einer funktionalen Sprache wird der Eventfluss konfiguriert, um zum Beispiel wie Nagios auf kritische Status mit einer Notifizierung zu reagieren oder das empfangene Event an andere Systeme wie Graphite weiterzuleiten.

Sascha Rüssel ist als Systemadministrator bei der synyx GmbH & Co. KG in Karlsruhe tätig, regelmäßig auf Konferenzen wie der FrOSCon, FOSDEM und OSDC sowie auf Twitter unter @da_zivi anzutreffen.

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**

Links & Literatur

[1] http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle

[2] http://jaxenter.de/artikel/Be-pragmatic-not-dogmatic-Sitzen-wir-nicht-alle-im-selben-Software-Boot

[3] http://blog.synyx.de/2013/09/yammer-metrics-made-easy-part-i

[4] http://www.crashub.org/

[5] http://www.jolokia.org/client/javascript-cubism.html

[6] https://github.com/Netflix/Hystrix/wiki/Dashboard

[7] https://www.icinga.org

[8] http://riemann.io/

[9] https://dropwizard.github.io/metrics/3.1.0/manual

[10] http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#production-ready

JAVA³

Jetzt abonnieren!
www.javamagazin.de

Jetzt 3 TOP-VORTEILE sichern!

1

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

Auch im entwickler.kiosk erhältlich

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