„DIE SERVER SIND AUSGELASTET. BITTE VERSUCHEN SIE ES SPÄTER ERNEUT.“

Samstag
22:00 Uhr
Deployment des neuen Simultaneous-Releases mit mehreren Dutzend abhängigen Systemen und zahlreichen neuen Features in jedem System mitsamt einer neuen technologischen Basis (Java 8). Vier Monate Entwicklung und Test werden am gleichen Tag ausgerollt.

Montag
08:00 Uhr
Zum Arbeitsbeginn starten die Anwender den Webbrowser und greifen auf das neue Release zu.

09:30 Uhr
Beschwerden unter den Anwendern häufen sich: Die Anwendung reagiere wieder einmal langsam. Erste Tickets werden beim User-Help-Desk eröffnet.

10:15 Uhr
Das Betriebspersonal sieht erste Alarmmeldungen: Einzelne Systemanfragen (Threads) benötigen mehr als zehn Minuten für einzelne Anfragen.

10:30 Uhr
Immer mehr Threads hängen, die CPU-Last erreicht die kritische Grenze von 95 Prozent.

10:35 Uhr
Einzelne Server fallen aus und müssen neu gestartet werden.

10:45 Uhr
Ein komplettes Cluster bricht zusammen.

11:00 Uhr
Breakdown, sämtliche Cluster fahren herunter. Sie konnten die Last aus dem ersten Cluster nicht abfangen.

11:05 Uhr
Panische Anrufe. Eskalation von Tickets.

13:00 Uhr
Alle Anwender werden nach Hause geschickt, nichts funktioniert mehr.

14:00 Uhr
Performance-Analysten sichern den Tatort und starten die Ermittlungen.

14:15 Uhr
Spiegel Online, Heise und andere Medien berichten über den Komplettsystemausfall – wieder einmal …

Go-Live mit Performance-Breakdown – und wie man sich dagegen wappnen kann.

| von JOSEF LEHNER

WIE KONNTE ES SO WEIT KOMMEN?

Von solchen (oder ähnlichen) wie links beschriebenen (Horror-) Szenarien hat fast jeder schon einmal gehört. Und die meisten enthalten auch ein Körnchen (oder mehr) Wahrheit. Aber warum konnte es überhaupt so weit kommen? Warum wurden die Performanceverschlechterungen nicht früher entdeckt und behoben?

Entwickler schreiben gerne performanten Code. Und in der Regel sind sie sehr stolz darauf. Aber sie haben in der Entwicklung kaum eine Chance, festzustellen, welche Codestellen die Anwendung ausbremsen werden. Hauptursachen sind massive Unterschiede zwischen Entwicklungsumgebung und Produktion (CPU, RAM, Netzwerk, Datenbank, Betriebssystem, kein Cluster, kein Enterprise Service Bus etc.), kaum repräsentative Testdaten, Single-User-Tests ohne Parallelität und vieles mehr – siehe weiter unten im Text.

Das Testteam, das mit viel Engagement nahezu alle Fehler vor dem Release findet, wird während des Tests kaum Performanceprobleme spüren. Denn die explorativen Tests werden kaum zu Last führen, und die automatisierten funktionalen Tests spiegeln die Aufrufhäufigkeit der Anwendungsfälle keineswegs wider. Zudem weist die Testumgebung oft ähnlich massive Unterschiede zur Produktion auf wie bei der Entwicklungsumgebung. Darüber hinaus greifen während der Tests keine anderen Systeme auf das System zu (wie zum Beispiel Fremdsysteme, Batches, Datensicherung).

Der Betrieb verwaltet in der Produktion eine Umgebung, die sich bezüglich nichtfunktionaler Anforderungen und deren Umsetzung durch Systemkomponenten meist gravierend von der Entwicklungs- und Testumgebung unterscheidet. Daher können Entwicklungs- und Testteam keine zuverlässigen Prognosen abgeben. Meist wird gepokert, in der Hoffnung, dass die Produktion wesentlich mehr Leistung besitzt als die anderen Systeme. Und dass die Anwendung selbst dann performant laufen wird, wenn sie ungeschickt programmiert wurde.

Kein Teilteam kann alleine dafür sorgen, dass die Anwendung performant und auch unter Stress fehlerfrei funktionieren wird. Sowohl für Projekte, die noch in der Entwicklung sind, als auch für produktive Systeme können jedoch Maßnahmen definiert werden, die ausreichend Qualität bezüglich Lastverhalten und Performance sicherstellen, sodass das nächste Release ein stressfreies wird.

MANAGEMENT

- Ressourcen für eine Lasttestumgebung bereitstellen
- Lasttester einstellen
- Performanceanalyse-Tools bereitstellen
- Anonymisierung der Daten einplanen

Ressourcen für eine Lasttestumgebung bereitstellen

Eine möglichst produktionsnahe Lasttestumgebung ist das A und O für ein stressfreies Release. Aus Kostengründen kann sie in der Praxis meist kein 1:1-Abbild der Produktion sein, daher muss sie sorgfältig herunterskaliert werden. Die Umgebung sollte jedoch die gleiche Hardware und Konfiguration wie die Produktion aufweisen. Dies gilt für Applicationserver, Datenbankserver und Netzwerk und schließt sämtliche Systeme wie Load Balancer, Enterprise Service Bus, Firewalls, Switches etc. mit ein. Damit Verschlechterungen zwischen Meilensteinen schnellstmöglich auffallen, sollte die Umgebung möglichst oft für Last- und Performancetests zur Verfügung stehen.

Eine abgespeckte Lasttestumgebung, in der die abhängigen Systeme simuliert werden, sollte zusätzlich bereitgestellt werden, damit für kleinere Meilensteine die Performance nachgetestet werden kann. Wichtig ist, dass diese Umgebung ständig verfügbar ist und vom Projektteam selbst verwaltet werden kann.

Lasttester einstellen

Lasttests – und geeignete Testdaten – müssen separat zu anderen Tests erstellt und gepflegt werden. Dies erfordert extra Personal und kostet Zeit und Geld.

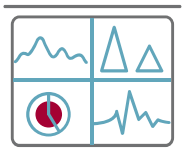
Performanceanalyse – Tools bereitstellen

In Abstimmung mit Betrieb und Entwicklung sollten geeignete Tools beschafft und genügend Lizenzen bereitgestellt werden, damit sie dem Team bereits zur Entwicklungszeit vertraut sind. Dies beschleunigt das Analysieren, Finden und Beheben von Fehlern über den ganzen Entwicklungsprozess.

Anonymisierung der Daten einplanen

Sowohl für die Entwicklung als auch für Tests sind anonymisierte Echtdaten von besonderer Bedeutung, da sie den Daten der Produktion entsprechen und somit reale Geschäftsvorfälle abbilden können.

FACHLICHKEIT



- Erwartete Aufrufhäufigkeit je Anwendungsfall
- Echtdatenbestandsanalyse durchführen

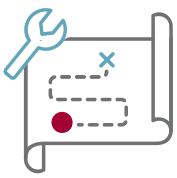
Erwartete Aufrufhäufigkeit je Anwendungsfall

Damit die Lasttester und Entwickler wissen, auf welche Anwendungsfälle sie sich konzentrieren müssen, ist eine umfassende Analyse notwendig. Welche Anwendungsfälle werden häufig aufgerufen, welche seltener? Wie wirkt sich die Performance auf den jeweiligen Anwendungsfall aus? Kann womöglich ein selten aufgerufener Anwendungsfall performancekritisch sein? Welche Service Level Agreements (SLA) wurden vereinbart?

Echtbestandanalyse durchführen

Wie groß sind die Datenstrukturen, die in den performancekritischen Anwendungsfällen behandelt werden? Wie viele Anschriften, Zahlungsverbindungen, Kommunikationsverbindungen, historisierte Daten etc. sind in einem Stammdatensystem für ein Unternehmen verknüpft? Performancerelevante Daten sollte der Fachbereich aufbereiten und für die Teams leicht zugänglich machen.

ENTWICKLUNG



- Profiling der Anwendung
- Analyse von Produktionslogs
- Trends aus dem Buildserver ableiten
- Erfahrung mit Frameworks aufbauen

Das Image des Unternehmens ist beschädigt. Wie konnte es dazu kommen?

Was sechs Wochen zuvor geschah

Die vom Testteam qualitätsgesicherte Version soll einem Last- und Performancetest[1] unterzogen werden, die Lasttestumgebung steht jedoch noch nicht bereit (neue Hardware, neue Softwareversionen, neue Versionen abhängiger Systeme, neues Deployment-Skript – es hakt an verschiedenen Stellen).

Andere Projekte beschweren sich ebenfalls. Jedes Projekt hat eine bestimmte Zeitscheibe reserviert, um alleine testen zu können. Eine Verzögerung in einem Projekt verschiebt alle Termine in allen anderen Projekte. Niemand kann und will auf die Lasttests verzichten – zu Recht!

Vier Wochen vorher …

Eskalation: Die Lasttestumgebung steht immer noch nicht bereit.

Zwei Wochen vorher …

Aus verschiedenen Gründen kann das Testen erst jetzt beginnen – endlich.

Erste Ergebnisse zeigen ein gravierend schlechteres Laufzeitverhalten als die Vorversion.
Liegt es an der neuen Java-Version? Oder an der neuen Netzwerk-Hardware, die kurz zuvor noch installiert wurde? Oder am neuen Applicationcode?

Analysen ergeben, dass manche Projekte die Terminverschiebung nicht mitbekommen und parallel getestet haben. Die Performancemessungen müssen wiederholt werden, sie haben dadurch zu viele ungültige beziehungsweise schlechte Werte produziert.

Der Wiederholungslauf ist deutlich besser, zeigt aber immer noch eine Verdopplung der Antwortzeiten gegenüber dem Vorrelease. Hat wieder ein anderes Projekt durch paralleles Testen zusätzliche Last erzeugt? Oder was ist diesmal der Grund? Umfassende und zeitaufwendige Analysen werden angestoßen. Das Vertrauen in die Testumgebung ist verloren.

Wenige Tage vor dem Breakdown …

Die Probleme sind nicht geklärt. Zahlreiche Performanceoptimierungen wurden mit heißer Nadel gestrickt. Wirklich geholfen hat nichts. Die Anwendung ist nun zwar in bestimmten Szenarien deutlich schneller als in der Vorversion, aber die durchschnittliche Performance ist nach wie vor doppelt so schlecht wie die der Vorversion.

Das Aussetzen des Releases ist nicht möglich, da die neuen Schnittstellen mit zahlreichen anderen Systemen abgesprochen wurden und alle am gleichen Tag live gehen müssen (Simultaneous Release Train). Also wird die Anwendung ausgerollt. Alle Beteiligten drücken die Daumen. Es wird schon gutgehen. Es ist bisher immer irgendwie gutgegangen.

Montag, 14:15

Mit bangen Blicken verfolgt das Team den Artikel auf Spiegel Online.

1 Performance = Werden (einzelne) zeitkritische Funktionen/Anwendungsfälle schnell genug ausgeführt (zum Beispiel komplexe Suchen über verschiedene Datenbestände)?
Last = Wie verhält es sich mit den Antwortzeiten bei der parallelen Anfrage/Funktionsausführung (auch von einfachen Funktionen) durch viele Benutzer?
Die beiden Begriffe Lasttests und Performancetests werden in diesem Artikel teilweise synonym oder als Sammelbegriff für beide Testarten verwendet.

Profiling der Anwendung

Damit die Entwickler im Falle eines Problems effizient zur Ursachenforschung eingesetzt werden können, sollten sie mit den Profiling-Tools in der Produktion beziehungsweise der Lasttestumgebung vertraut sein. Idealerweise sind dieselben Tools auch auf dem Entwicklungsrechner lauffähig. So können potenzielle Performanceprobleme frühzeitig erkannt und Optimierungen schon lokal auf dem Entwicklerrechner geprüft werden. Auch gut geschriebene Logfiles, die die Problemanalyse beschleunigen, indem sie wertvolle Informationen zum Anwendungsfall liefern, sind ausgesprochen hilfreich.

Analyse von Produktionslogs

Produktionslogs sollten auf Auffälligkeiten untersucht und Aufrufhäufigkeiten sowie Performanceausreißer dem Fachbereich und den Testern bekanntgemacht werden.

Trends aus dem Buildserver ableiten

In den meisten Projekten kommt bereits Continuous Integration zum Einsatz – eine Technik, die das Auffinden von Fehlern zeitnah ermöglicht. Diese Technik sollte um Tests, die Performanceveränderungen erkennen, erweitert werden. So kann frühzeitig und aktiv auf Veränderungen der Performance reagiert werden. Durchschnittliche Antwortzeiten werden wegen fehlender Repräsentativität wahrscheinlich kaum messbar sein, dennoch können Trends abgeleitet werden. Hierzu ist es wichtig, dass die Testumgebung sich die Ressourcen nicht mit anderen Systemen teilen muss (das heißt, keine Virtualisierung) beziehungsweise von der Performance externer Systeme nicht beeinflusst wird, sodass gemessene Werte vertrauenswürdig sind. Interessante Trends sind Anzahl Methodenaufrufe, durchschnittliche Antwortzeit aufgeteilt nach Perzentil (90 % < 100 ms, 95 % < 150 ms ...), Anzahl abgesetzter SQL-Statements, Speicherallokationen, Garbage-Collection-Häufigkeit und -Dauer sein.

Erfahrung mit Frameworks aufbauen

Der Sinn und Unsinn von Frameworks wurde bereits in vorherigen Artikeln besprochen.[2] Verwendet man ein Framework, verkapselt es viel Komplexität, mit der sich der Entwickler nicht mehr auseinandersetzen muss. Was dann jedoch auch „verkapselt“ ist, sind die Auswirkungen auf Last und Performance durch diese „schwergewichtigen“ Unterbauten. In Hinblick auf die Performance sollte man jedoch die Konfigurationen der Frameworks überprüfen und gegebenenfalls anpassen. Standardwerte für Framework-Konfigurationen können massive Auswirkung auf die Performance in der Produktion haben.

2 .public Ausgabe 01-2015 und 02-2015

TEST

- Sicherstellen einer produktionsnahen Lasttestumgebung
- Sicherstellen produktionsnaher Testdaten
- Reproduzieren historischer Messwerte
- Lasttestumgebung ausnutzen

Sicherstellen produktionsnaher Lasttestumgebungen

Bevor die Tests beginnen, sollte die Übereinstimmung der Konfiguration mit derjenigen der Produktionsumgebung sichergestellt werden. Nichts ist ärgerlicher, als nachträglich festzustellen, dass die Laufzeitumgebungen in Test und Produktion sich bezüglich der Konfiguration deutlich unterscheiden (zum Beispiel andere Software-Versionen, falsche Arbeitsspeicherzuweisung [Heap], andere Garbage-Collection-Strategie, viel zu kleiner Datenbank-Connection-Pool, zu detailliertes Logging).

Sicherstellen produktionsnaher Testdaten

Um möglichst reale Geschäftsvorfälle abzubilden, sollten die eingesetzten Testdaten mit der Produktion abgeglichen werden. Auf keinen Fall darf nach dem Go-Live eines Releases vergessen werden, die Lasttestdaten (Echtdaten, Anzahl und zeitliche Verteilung der Anfragen) mit denen aus der Produktion zu vergleichen und bei Differenzen die Lasttests zu überarbeiten.

Reproduzieren historischer Messwerte

Vor dem nächsten Release empfiehlt es sich, die Lasttestdaten mit einem Wiederholungslauf der alten Version nochmals zu verifizieren, um eine vertrauenswürdige Ausgangsbasis für die nachfolgenden Lasttests aufzubauen. Sofern eine Aktualisierung der Produktionskomponenten geplant ist, empfiehlt sich ein zweistufiger Prozess, bei dem zuerst die Hardware und dann das neue Release aktualisiert werden (oder umgekehrt). Es erleichtert die Fehleranalyse sehr, da sich immer nur ein Bestandteil im Aufbau verändert hat.

Lasttestumgebung ausnutzen

Über mehrere Testläufe sollten die Last- und Performancegrenzen der Anwendung ausgelotet werden. Die Simulation sollte die Lastverteilung mit Lastspitzen, gleichzeitige Zugriffe externer Systeme und parallel dazu ausgeführte Batches über einen längeren Zeitraum einschließen, um anhaltende Performanceverschlechterungen nach Stressphasen und bei Speicherlecks feststellen zu können.

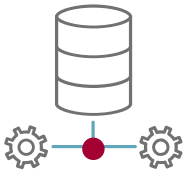
DATENBANKSPEZIALIST

- Einbeziehen eines Datenbankspezialisten

Einbeziehen eines Datenbankspezialisten

Die meisten Anwendungen basieren auf einer Datenbank, und diese bildet bei über der Hälfte der Anwendungen einen Performanceengpass.[3] Ein Datenbankspezialist kann bereits während der Entwicklung diesem Problem entgegenwirken, indem er performante Lösungen für Datenbankdesign, -abfragen, Indexstrukturen etc. erarbeitet und von der Entwicklung bis zur Produktion die Datenbankmetriken überwacht.

BETRIEB (OPS)



- Einbeziehen des Betriebes

Einbeziehen des Betriebes

Nach dem Vorbild des Dev-Ops-Ansatzes[4] sollte schon während der Entwicklung mit dem Betrieb zusammengearbeitet werden. Wünschenswert wäre es, wenn in der Entwicklung und im Betrieb dieselben Tools für Installation, das Monitoring und Profiling verwendet würden. Außerdem sollte der Betrieb in die Betreuung der Lasttestumgebung involviert sein, damit sich keine Unterschiede zur Produktion einschleichen. Auffälligkeiten sollten der Entwicklung und dem Test mitgeteilt und gemeinsam an Lösungen gearbeitet werden (Logdateien, die ständig rollieren, weil zu viel geloggt wird, Server, die neu gestartet werden müssen etc.).

Vier Monate später – Montag

08:00 – 09:00 Uhr

Zum Arbeitsbeginn starten die Anwender den Webbrowser und greifen auf das neue Release zu.

09:30 Uhr

Noch kein Benutzer hat sich bei der Störungshotline gemeldet; die Log-Files zeigen keine Auffälligkeiten.

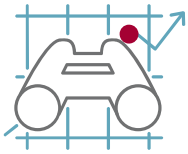
10:15 Uhr

Der Projektleiter ruft beim Betrieb an, um sich zu vergewissern, dass die neue Version live gegangen ist.

11:30 Uhr

Der Projektleiter bedankt sich für die reibungslose Produktivsetzung bei den Teilteams; er hat für 12:30 Uhr Pizza für alle bestellt …

ÜBERGREIFENDE THEMEN



- Architekturvorgaben
- Definition der Produktionsumgebung
- Support

Architekturvorgaben

Der Hard- und Software-Stack wird meist von Architekturgruppen und Experten bestimmt. Daher ist es in ihrer Verantwortung Best Practices für Architekturmuster bereitzustellen und bei Schwierigkeiten Unterstützung zu liefern.

Definition der Produktionsumgebung

Die Umgebung, in der die Anwendung betrieben wird, liegt meist außerhalb des Entscheidungsbereichs des Projektteams. Dies betrifft den Großteil des Hardware- und des betrieblichen Software-Stacks. Daher liegt die Verantwortung bezüglich Performance auch bei den Betreibern dieser Umgebung. Bei Upgrades kann nicht jedes Projekt Performancemessungen durchführen, sondern muss sich auf die Betreiber verlassen können.

Support

Funktionierender Support ist ein nicht zu unterschätzender Faktor bei der Entwicklung und vor allem im Betrieb. Unzureichende Problemlösung wird den Ruf des Supportanbieters, des Betriebes und letztendlich auch das Projekt an sich schädigen, auch wenn Letztgenannte nichts dafürkönnen.

FAZIT

Mit diesen beschriebenen Maßnahmen wird das nächste Release deutlich besser verlaufen. Klar ist: Ein stressfreies Release ohne Last- und Perfomancetests ist nicht möglich. Zu groß ist die Ungewissheit, ob die Anwendung die operative Last und die Performanceanforderungen aushalten wird. Last- und Performancetests bilden die Basis für Vertrauen in die Anwendung. Daher müssen sie mit Sorgfalt entwickelt und auf einer realen Datenbasis und Umgebung durchgeführt und reproduziert werden können. Alle Teilteams müssen Vertrauen in die Umgebung und Daten besitzen und gemeinsam Optimierungen durchführen. ●

ANSPRECHPARTNER – JOSEF LEHNER

IT-Consultant

Public Sector Solutions Consulting

- 49 176 31592222
- josef.lehner@msg-systems.com

3 http://www.appdynamics.com/solutions/database-monitoring/
4 https://de.wikipedia.org/wiki/DevOps