Best Practices für Last- und Performance-Tests von Enterprise-Applikationen auf Basis der Oracle Fusion Middleware

Christian Kunzmann, enpit consulting OHG

Die Performance von Anwendungen auf der Fusion Middleware kann auf verschiedenste Weisen kontrolliert werden: Entwickler instrumentieren ihren Code in Frameworks oft händisch. Weiter Fortgeschrittene setzen auf Tools für automatische Instrumentierung unterhalb der Entwicklung. Last- und Performance-Tests sollten spätestens als Quality-Gate vor der Produktivsetzung durchgeführt werden. Danach bringt eine kontinuierliche Überwachung von Produktion und Entwicklungsumgebungen weitere Vorteile. Der Artikel gibt einen Überblick über Möglichkeiten und Einschränkungen dieser Arten des Performance-Monitoring und geht dabei auf Best-Practices aus Projekt-Erfahrungen ein. Der stufenweise Aufbau des Artikels spiegelt dabei das empfohlene, evolutionäre Vorgehen wider.

Während der Entwicklung von komplexen Systemen in umfangreichen Frameworks können oft einfache Mittel zu großen Mehrwerten führen. Auch bei Kunden mit erfahrenen Entwicklerteams konnten wir wiederholt feststellen, dass solche einfachen Mittel mit großem Optimierungspotenzial übersehen wurden. Wenn die Entwickler an relevanten Codestellen wie Laufzeiten, Ergebnisgrößen und Funktionsparameter loggen, führt dies zu einem besseren Verständnis und Nachvollziehbarkeit der Anwendung. Besonders in mächtigen Frameworks werden oftmals Fehler eingebaut, die die Anwendungsperformance gefährden, wenn sie übersehen werden. Dazu gehören Wiederholungen in der Geschäftslogik wie Validierungen, unnötige Datenbank- oder Web-Service-Aufrufe sowie unnötig große abgerufene Datenmengen. Dabei gilt es, einen guten Kompromiss zu finden zwischen einem umfassenden Logging, der Lesbarkeit des Logs sowie dem Performance-Overhead durch das Logging, der gering, aber vorhanden ist. Das Zielbild ist eine mehrstufige Konfigurierbarkeit sinnvoller, redundanzfreier Log-Ausgaben.

Zu den eingesetzten Log-Leveln sollte es eine team- oder unternehmensweite Konvention geben. So können mit einfachsten Mitteln sehr früh Fehler in der Entwicklung und damit eine häufige Ursache von Performance-Problemen verhindert werden. Die Transparenz für die Entwickler bei geringer Einstiegshürde sind weitere Vorteile dieses Ansatzes. So bildet eine qualitative Software-Entwicklung das Fundament für performante Anwendungen.

Frameworks und Werkzeuge zur Unterstützung

Das Entwicklungsteam wird geschaffene Werkzeuge generalisieren und wiederverwenden, sei es für wiederkehrende Anforderungen oder für Folgeprojekte. So sollte eine Basis-Instrumentierung beispielsweise in Basisklassen oder Utils verankert sein. Derartig programmierte Instrumentierung bedeutet zwar initial einen Aufwand für die Entwicklung, bringt aber den Mitarbeitern

viele wertvolle Einblicke in ihr Produkt sowie eingesetzte Frameworks oder Services.

Daneben bietet die Instrumentierung durch externe Tools oder Bibliotheken erfahreneren Teams die Möglichkeit, ohne expliziten Code und damit ohne Entwicklungsaufwand durch einfaches Zuschalten Einblicke in die Anwendungsperformance zu gewinnen und Schwächen zu analysieren. Dabei können grafische Übersichten die Auswertung unterstützen und viele Daten aggregiert effizient darstellen.

Bei derartigen Werkzeugen ist neben dem bekannten Java Mission Control auch ein Third-Party-Produkt zu nennen, mit dem der Autor in Projekten gute Erfahrungen machen konnte: Dynatrace. Java Mission Control war in JDK 6 nur in JRockit enthalten, ab JDK 7 ist es auch Bestandteil von Oracles JDK. Entwicklungsumgebungen können lizenzfrei instrumentiert werden. Für den Produktiveinsatz wird Java SE Advanced Support benötigt, der bei vielen Oracle-Kunden bereits vorhanden sein sollte. Bei Kunden, die noch JDK 6 Hotspot im Einsatz haben, kann mit anderen Third-Party-Produkten Ähnliches erreicht werden.

Als besonders wertvoll haben sich dabei die Möglichkeiten zum Drilldown in Stacks von Methodenaufrufen und deren Laufzeitverhalten erwiesen, um nicht performante Anwendungsteile oder auch Service-Aufrufe zu identifizieren. Allerdings muss dabei bedacht werden, dass für derartige Werkzeuge auch entsprechendes toolspezifisches Know-how nötig ist. Solche Application-Performance-Monitoring-Werkzeuge (APM) sollten auch in Lasttests eingesetzt werden, um Einblicke in das Anwendungsverhalten unter Last zu bekommen. Bei Eigenentwicklungen können und sollten die Ansätze von programmierter und impliziter Instrumentierung ergänzend und nicht alternativ genutzt werden.

Last- und Performance-Test als Quality Gate vor der Produktion

Als Quality Gate vor einer Produktivsetzung sollten Last- und Performance-Tests erfolgen. Diese prüfen die Qualitätsmerkmale der Effizienz und Zuverlässigkeit. Dabei wird typischerweise das Verhalten der Antwortzeiten bei typischer Last und Dauerlast geprüft, um die Einhaltung vereinbarter Service Level Agreements (SLA) sicherzustellen. Das Verhalten im Dauerbetrieb wird geprüft, um Probleme wie Speicherlecks auszuschließen. Das Verhalten bei Ausfallen von Systemkomponenten prüft, ob implementierte Failover-Strategien greifen und somit eingesetzte Systemkomponenten korrekt konfiguriert sind.Als Grundlage für Last- und Performance-Tests wird das Mengengerüst geschätzt, etwa in Form von gleichzeitigen Nutzern und deren Klickverhalten oder über die Anzahl von Requests. Dabei stehen bei internen Anwendungen oder bei Ablösungen bestehender Systeme konkretere Informationen bereit als bei der Produktivsetzung eines neuen Angebots im Internet. So ist mit höherer Ungewissheit mehr Flexibilität und Vorbereitung nötig. Relevante Betriebseinheiten sollten gegebenenfalls möglichst frühzeitig angefragt und vorgewarnt werden, um bei Bedarf möglichst schnell reagieren zu können.

Zur Durchführung sollten neben effizienten, protokollbasierten Tests auch Oberflächentests durchgeführt werden, um das Verhalten der Anwendung belastbar abzubilden. Dazu sollte man aus der Menge bestehender Oberflächentests eine kleine Untermenge auswählen, die die Anwendung möglichst repräsentativ abdeckt. Um aussagefähige Ergebnisse zu produzieren, sollten die Tests vor allem auch die Anwendungsteile abdecken, die als Nutzungsschwerpunkte angenommen werden oder Performance-kritisch sein könnten.

Dazu können dann mit einem Testmanagement-Werkzeug Testclients angesteuert werden, die automatisiert beispielsweise im Browser die Web-Anwendung durchlaufen. Die dort erhobenen Client-Metriken wie Antwortzeit-Verhalten und Korrektheit der Antworten werden gesammelt und mit Server-Metriken wie Heap-Verlauf, Response-Zeiten, Anzahl der Requests pro Minute etc. aggregiert. So ergeben die Innensicht der Anwendung, deren Ausführungsumgebung und die Clientsicht ein umfassendes Bild des Systemverhaltens, das bei Bedarf noch um Metriken bestimmter Systemkomponenten angereichert werden kann. Es bietet sich an, hier auf APM-Werkzeuge zurückzugreifen, die auch in der Entwicklung im Einsatz sind, damit alle Beteiligten schnell ein gemeinsames Bild bekommen und eventuelle Befunde möglichst effizient analysiert werden können.

Die Tests sollten auf einer dedizierten Umgebung durchgeführt werden, die der Produktion soweit wie möglich ähnelt. Dabei werden in geclusterten Systemen natürlich weniger Knoten zum Einsatz kommen, aber die Performance eines einzelnen Knoten sollte vergleichbar sein, um belastbare Ergebnisse zu erzielen. Hier sind die Ausstattung in RAM beziehungsweise Heap Size, CPU, IO- und Netzwerk-Anbindung relevante Aspekte sowie auch der Ausschluss paralleler Anwendungen beziehungsweise Lasttreiber.

Eingesetzte Systemkomponenten aus dem Produktivsystem sollten möglichst auch in der Lasttest-Umgebung vorhanden sein. So wird nicht nur die Belastbarkeit der Ergebnisse erhöht, sondern auch die Konfiguration der Systemkomponenten mit getestet und qualitätsgesichert. Zu betrachten sind beispielsweise Load Balancing, Clustering, Web Caches oder auch Firewalls. Während viele Kunden mit Firewalls umgehen können, die auf Ebene des Netzwerkprotokolls agieren, hat der Autor mit Web-Application-Firewalls andere Erfahrungen gemacht. Diese filtern Requests, um etwa Denial-of-Service-Attacken abzufangen. Damit wird auch diese Systemkomponente zum potenziellen Performance-Flaschenhals.

Selbst in großen und erfahrenen Häusern kann es vorkommen, dass durch Fehlkonfigurationen im Load Balancing und Clustering die Last ungleich verteilt wird. Hier kann eine Überlast auf einem einzelnen Knoten diesen lahm legen und so sukzessive ein ansonsten gut konzipiertes System außer Funktion setzen.

Auch im Zusammenspiel zwischen Web Caches, etwa Squid für die performante Auslieferung von statischen Ressourcen wie Bildern und vorgelagerten Load Balancern, kann es zu Fehlern kommen, wenn der Load Balancer beispielsweise an jeden Request eine Session-ID anhängt. Im Clustering von Oracle WebLogic sollte typsicherweise die angebotene State Duplication konfiguriert und genutzt werden, um mögliche Failover vor dem Kunden zu verbergen und dessen User Experience zu unterstützen.

Die Praxis zeigt, dass in solchen Fällen eine enge Zusammenarbeit zwischen den Durchführenden der Last-Tests, den Betriebseinheiten der Umgebung sowie dem Entwicklungsteam mit dem Wissen über Anwendungsinterna nötig ist. Je nach Projekt-/Kundenumfeld stellt dies eine nicht zu unterschätzende Herausforderung dar. Durch frühzeitige, zielgerichtete Kommunikation sollte hier am besten bereits vor den Tests eine gute Arbeitsbasis geschaffen werden. Ferner ist an dieser Stelle auf die Vorteile von DevOps zu verweisen. Durch personelle Nähe oder gar Zusammenlegung oben genannter Aufgaben wird deren effiziente Umsetzung nachhaltig gestützt.

Sofern die durchgeführten Last- und Performance-Tests Problew me aufgezeigt haben, muss man diese analysieren und be-

heben. Spätestens danach sollte es möglich sein, mit den Ergebnissen anhand der geschätzten Nutzerlast die nötige Skalierung der Produktivumgebung durchzuführen. Weiter kann dann anhand der Ergebnisse die Einhaltung vereinbarter Service Level Agreements geprüft werden und somit optimalerweise eine Freigabe erteilt werden.

Kontinuierliche Überwachung der Produktion

Mit der erhaltenen Freigabe sollten Last- und Performance-Tests nicht enden. Gerade im Zuge sich weiter verbreitender Continuous Integration und Continuous Delivery, aber auch klassischer Produkt-Updates, sollte sich die Durchführung des Quality Gate wiederholen. Zusätzlich sollte eine kontinuierliche Überwachung der Produktion instanziiert sein. Diese überwacht durch technisches Monitoring die Verfügbarkeit und Reaktionszeiten der Anwendung und damit eine fortwährende Einhaltung der SLAs.

Zu guter Letzt muss sichergestellt sein, dass die Nutzung der Anwendung mithilfe fachlichen Monitorings qualitativ ausgewertet werden kann. Um etwa bei einer eigenentwickelten Oracle-ADF-Anwendung die durch ADF-Task-Flows abgebildeten, fachlichen Anwendungsteile zu identifizieren, bietet sich das Oracle-Produkt „Real User Experience Insight" an, um das Konzept der Task-Flows zu verstehen und interpretieren zu können. In der Praxis hat der Autor jedoch erleben müssen, wie in der eingesetzten Version (korrespondierend zu ADF 11.1.1.6) dieses Werkzeug durch einen bekannten Bug in Exceptions lief und damit die Anwendung aushebelte. Somit bleibt zumindest für Eigenentwicklungen zu berichten, dass fachliches Monitoring oftmals nicht trivial ist, da vielen Standardwerkzeugen der Einblick in AJAX-Requests oder ADF Task Flows fehlt und somit meist eigenes fachliches Logging benötigt wird.

Fazit

Performance Monitoring sollte evolutionär betrachtet werden, um die Mitarbeiter mitzunehmen. Eine Bereitstellung umfangreicher Werkzeuge für ein wenig erfahrenes Team trägt nicht weit. Durch frühe und einfache erste Betrachtungen des Themas „Performance" wird Transparenz geschaffen, Know-how aufgebaut sowie für das Thema sensibilisiert.

Last- und Performance-Tests sind ein wertvolles Quality Gate. Annahmen zum Lastverhalten und somit die Skalierung von Produktions-Umgebungen sind abzusichern. Auch das komplexe Zusammenspiel diverser Systemkomponenten und deren passende Konfiguration sollte sich unter Last beweisen, da hier oftmals Nachbesserungsbedarf besteht.

Dabei sollte die Vergleichbarkeit zur Produktionsumgebung maximiert werden und damit die Belastbarkeit der Ergebnisse. Diese Tests sollten in einem iterativen Prozess verankert sein, um die gelieferte, hohe Qualität zu sichern und beizubehalten. Neue, Performance-relevante Anwendungsteile sollten dementsprechend immer Last- und Performance-Tests durchlaufen. Die Einhaltung der SLAs sollte auf Produktions-Umgebungen kontinuierlich überwacht werden.

Christian Kunzmann
ck@enpit.de

DevOps und Microservices beschleunigen Applikationsbereitstellung

Markus Eisele, Red Hat GmbH

DevOps verzahnt IT-Entwicklung und Betrieb miteinander und ermöglicht so die schnelle und schrittweise Bereitstellung von Software. Das Konzept bildet auch einen wichtigen Erfolgsfaktor für die Entwicklung und den Betrieb flexibler Microservices-Architekturen. Umgekehrt vereinfachen Microservices die Implementierung von DevOps.

Trends wie Big Data, Mobile, Cloud Computing und das Internet der Dinge stellen Unternehmen aller Branchen vor große Herausforderungen. Sie müssen ihre Geschäftsprozesse schnell und flexibel an diese Veränderungen und neue Anforderungen anpassen. Hier ist speziell die IT-Abteilung gefordert. Fachabteilungen erwarten, dass sie Anwendungen in hoher Qualität sehr schnell entwickelt und neue Funktionen sowie Updates zügig ausliefert. Kurzum: IT-Entwicklung und -Betrieb müssen agil sein. Um dies zu erreichen, gibt es zwei wichtige Optionen: Zum einen den Aufbau einer flexiblen Anwendungs-Architektur (unter anderem mit Microservices), zum anderen innerhalb der IT-Abteilung eine engere Zusammenarbeit von Entwicklung und Betrieb (DevOps), um Reibungsverluste zu minimieren und die Bereitstellung von Anwendungen zu beschleunigen. Beide Ansätze ergänzen sich gegenseitig.

Microservices als Ansatz für agile Anwendungen

Monolithisch strukturierte Anwendungen stoßen in puncto Agilität durchaus auch an ihre Grenzen. Ändern Entwickler nur einen kleinen Teil der Anwendung, muss die gesamte Applikation zumeist unter großem Aufwand neu getestet werden. Ziel einer Microservices-Architektur ist, dies deutlich flexibler zu gestalten.

Im Gegensatz zu monolithischen Architekturen bestehen Microservices aus lose