Ausgabe 05/2020

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

D: 4,90 EUR | CH: 9,80 CHF | A: 5,60 EUR | Benelux: 5,80 EUR

4 191978 304903 05

Java aktuell

iJUG Verbund
www.ijug.eu

Microservices
Performancetests, Service Meshes, MicroProfile GraphQL und mehr

Javas Geheimnisse
Weniger bekannte Features und Eigenheiten

Deep Learning
Einblick in das Trend-Thema

Klein aber oho: MICROSERVICES

JavaLand

16. - 18. März 2021
in Brühl bei Köln

Save the Date

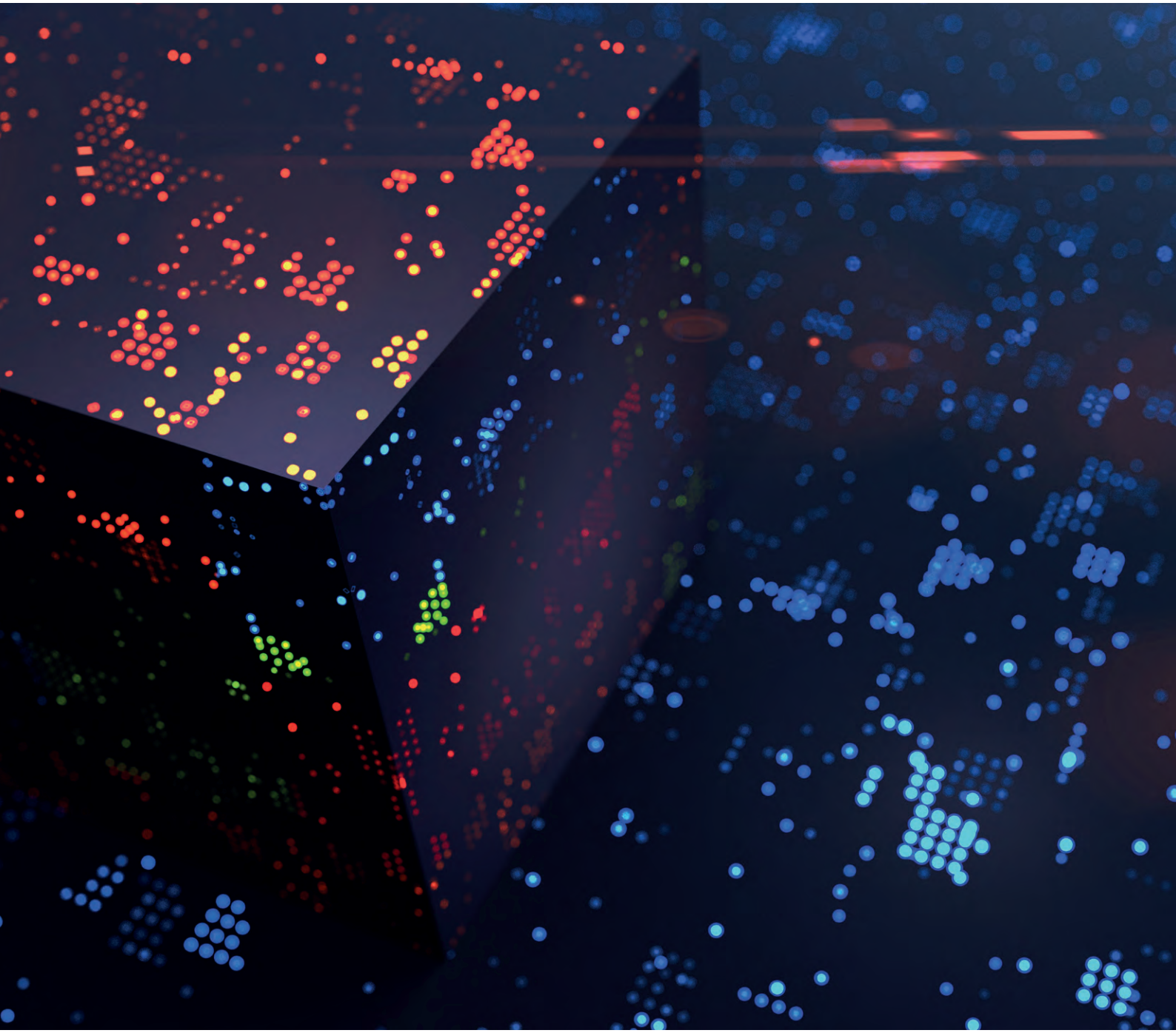
Hybride Veranstaltung

Was die JavaLand als Plattform für Wissentransfer und Networking ausmacht, kannst du im Phantasialand oder online erleben. Als Teilnehmer entscheidest du selbst, welche Variante du wählen möchtest.

Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu



Performancetests von Microservices

René Schwietzke, Xceptance GmbH

Eine Softwarearchitektur, die aus kleinen und unabhängigen Teilen zusammengesetzt ist, ist einfacher zu erstellen und zu warten. Microservices sind dadurch zum dominierenden Thema in der modernen Softwareentwicklung geworden. Während die Komplexität auf der Entwicklungsseite sinkt, steigt sie in Bezug auf die Themen Architektur, Performance und Zuverlässigkeit. Dieser Artikel diskutiert die Anforderungen an die Performance und die Planung von Performancetests von Microservices.

Microservices definieren sich laut Jaxcenter *[1]* wie folgt: „Die wesentliche Eigenschaft von Microservices ist das unabhängige Deployment." Laut Red Hat *[2]* hat eine Microservice-Architektur die folgenden Vorteile: schnellere Markteinführung, hochgradig skalierbar, Robustheit, einfache Implementierung, besserer Zugriff, mehr Offenheit." Die Punkte Skalierbarkeit und Robustheit lassen sich allerdings nur über umfangreiche Tests nachweisen. Diese Tests sind gleichzeitig auch Bedingung für eine schnelle und verlässliche Markteinführung. Skalierung und Robustheit sind hochkomplexe Themen, die im Design und in der Implementierung logisch und beherrschbar erscheinen, deren tatsächliches Verhalten jedoch meist von den Erwartungen abweicht.

In den folgenden Kapiteln wird meist „Services" als Synonym für Microservices verwendet, um die textuelle Darstellung zu verkürzen und zu verdeutlichen, dass Tests von größeren Services ähnlich zu Microservices erfolgen.

Testansätze

Services lassen sich auf vier Arten testen: isoliert, direkt, indirekt oder im Rahmen des Gesamtsystems.

Isoliert: Die Kommunikation zu anderen Services wird durch geeignete Mocks oder Simulatoren ersetzt und die Testlast erreicht den Service direkt über sein Interface.

Direkt: Beim direkten Test wird die Kommunikation zu weiteren Services nicht ersetzt (gemockt). Damit lassen sich Probleme mit der Service-Interkommunikation sowie durch Feedback hervorgerufenes Verhalten beobachten und beurteilen.

Indirekt: Wenn ein Service als konsumierter Service getestet wird, dann wird die Last nicht direkt auf den Service angewendet, sondern indirekt durch den Test anderer Anwendungsbereiche.

Gesamtsystem: Der Test im Gesamtzusammenhang ist am realistischsten. Er hilft falsche Annahmen auszuschließen, zeigt die Abhängigkeiten und verdeutlicht den Einfluss einzelner Komponenten auf das Gesamtsystem.

Es gibt kein allgemeines Rezept dafür, welcher Ansatz zum Test eines Service am besten geeignet ist. Diese Fragestellung muss individuell beantwortet werden. Empfohlen sind zu Beginn das isolierte Testvorgehen und ein Gesamtsystemtest auf der zukünftigen Hard- und Softwareplattform.

Trotz gegenteiliger Empfehlungen verwenden Services oft gemeinsame Komponenten. Das sind häufig Datenbanken und Eventbusse sowie generische Dienste, wie Storage, Security, Log-Aggregatoren und Monitoring. Bei der Planung von Tests ist der Einfluss dieser Drittsysteme zu beachten. Häufig lastet zum Beispiel ein Dienst die Datenbank aus und verringert dadurch die Performance anderer Dienste.

Anforderungen

Zuerst werden allgemeine, nicht funktionale Anforderungen wie Skalierbarkeit, Robustheit und das schnelle Deployment betrachtet.

Das Thema Performancetests sollte nicht in seiner oft klassischen Definition gesehen werden – als reine Messung der Performance unter einer bestimmten Arbeitslast –, sondern als ausführliches Vorgehen, um das gesamte Verhalten der Anwendung unter wechselnden Lasten und Zuständen zu beurteilen. „Performance testing… It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage *[3]*." Denn aus dem späteren Einsatzzweck der Services ergeben sich weitergehende Anforderungen, zum Beispiel eine vorgegebene Verfügbarkeit oder garantierte Antwortzeiten. Auch das Datenwachstum zustandsbehafteter Services kann die Performance beeinflussen.

Durchsatz

Typischerweise steht meist der gewünschte Durchsatz im Vordergrund und selten die erwarteten Antwortzeiten. Beide stehen allerdings in einem engen Zusammenhang und sollten auf keinen Fall losgelöst voneinander getestet werden, denn der Durchsatz definiert sich als die Anzahl von Anfragen, die ein Service in einem Zeitraum unter Einhaltung der Kriterien Antwortzeiten und Stabilität beantworten kann.

Man sollte auf keinen Fall den Durchsatz als alleinige Größe nutzen, denn damit sind weder Antwortzeiten noch Fehlerraten ein Teil des Gesamtbildes. Es würde also nur eine quantitative Aussage getroffen, keine qualitative.

Antwortzeiten/Latenz

Die Zielgrößen für die Antwortzeiten sollten mindestens ein Perzentil mit einer Maximalzeit für den Großteil der Antworten sowie die maximal tolerierbare Antwortzeit für einzelne Requests umfassen, zum Beispiel 100 ms im Perzentil P99, aber nie mehr als 500 ms.

Ein P99 von 100 ms bedeutet, dass 99 Prozent aller Anfragen in 100 ms oder weniger beantwortet werden müssen. Nur in einem Prozent der Fälle darf die Antwortzeit darüber liegen *[4]*. Ein P99 definiert kein oberes Limit.

Die Nutzung von Durchschnitten sollte vermieden werden, da diese Spitzen nicht abbilden und oft langsam auf Änderungen reagieren. Durchschnitte können jedoch zum Vergleich zweier Tests zusätzlich zu den PXX-Werten herangezogen werden.

Für Services mit hohen Anforderungen an den Durchsatz sollte das P99.9 genutzt werden, da ein P99 es einem Prozent der Antwortzeiten erlaubt, über dem Zielkriterium zu sein. Das sind bei 100 Requests pro Sekunde bereits 3.600 Anfragen pro Stunde, die nicht innerhalb der Zielerwartung liegen. Auch die Kombination von P95, P99, P99.9 und einer Maximalzeit kann hilfreich sein, um die Erwartungshaltung genau zu definieren und damit auch das spätere Produktionsverhalten mit einer Garantie ausstatten zu können.

Bei der Definition der Ziele sind die spätere Nutzung und natürlich die Aufgabe des Service einzubeziehen. Die Antwortzeiterwartung sollte bereits zu Beginn der Implementierung feststehen, da diese einen entscheidenden Einfluss auf die Umsetzung haben kann.

Bei der Betrachtung der Antwortzeiten kann man entweder von einem komplett identischen Verhalten zu jedem Zeitpunkt ausgehen oder die Erwartungen abstufen, zum Beispiel in Normalzustand

und Ausnahmezustand. Ausnahmezustände können Deployments, Komponentenausfall oder Überlastsituationen sein.

Stabilität

Im Idealzustand sollte ein Service natürlich fehlerfrei laufen. „Fehlerfrei" ist hier durch das erwartete Antwortverhalten definiert. Fehler können rein technischer Natur sein, wie Connection-Fehler oder Timeouts, aber auch funktionaler Natur, verursacht beispielsweise durch inkorrekte Implementierung von Logik, die unter Belastung zu unerwarteten Daten oder Betriebszuständen führt.

Zusätzlich kann die Stabilität wieder für den normalen Betriebszustand und für Ausnahmen definiert werden. Für normale Betriebszustände sollte man eine Null-Fehler-Policy definieren. Das sollte auch für die Skalierung gelten, da dies kein besonderer Betriebszustand ist. Ausnahmesituationen sollten auftretende technische Fehler selten sein, zum Beispiel ein Prozent der Anfragen betreffen, und das Verhalten sollte definiert sein, also sind beispielsweise 502/503 Proxyfehler-Meldungen in Ordnung, Connection Resets oder Timeouts jedoch nicht. Funktionale Fehler sollten in keinem Fall auftreten.

Skalierung

Der Begriff Skalierung ist im Zusammenhang mit Microservices oft leichtsinnig im Gebrauch: „Hochgradig skalierbar – wenn der Bedarf für bestimmte Services steigt, können diese über mehrere Server und Infrastrukturen hinweg flexibel implementiert werden *[2]*."

Ein Microservice oder Service skaliert nicht per se, sondern bringt die Grundlagen mit, im Vergleich zu einer traditionelleren Architektur einfacher skalierbar zu sein.

Systeme lassen sich auf zwei Arten skalieren *[5]*. „Scale Out" – mehr zusätzliche Komponenten der gleichen Art (logische Einheit), also zum Beispiel mehr Serviceinstanzen durch mehr Maschinen – und „Scale Up" – mehr Ressourcen pro Serviceinstanz, zum Beispiel mehr CPUs oder Speicher. In beiden Fällen ist die Erwartung, dass eine Verdoppelung der Ressourcen auch die Leistung verdoppelt, also mehr Durchsatz bei gleichen Antwortzeiten. In den wenigsten Fällen dürfte diese Erwartung eintreten, denn die Aufgaben des Service und sein Zustand müssten dafür linear skalierbar sein.

In fast allen Fällen muss Skalierung erprobt und untersucht werden. Liefern mehr Ressourcen mehr Leistung und wenn ja, bis zu welchem Punkt und mit welchem Faktor? Welche unerwarteten Skalierungshemmnisse treten dabei auf und lassen sich diese beseitigen? Diese Tests können auch zu einem Teil der betriebswirtschaftlichen Abschätzung der Kostenbasis eines Service dienen.

Ausfallsicherheit

Ausfallsicherheit wird in modernen Service-Landschaften häufig durch Chaos Testing *[6]* ausprobiert und herausgefordert. Nicht jede Organisation hat den Mut dazu und es scheint auch nicht ratsam, ohne Basistests mit dieser Methode zu beginnen, da zunächst Gewissheit bestehen sollte, dass der Service prinzipiell den Anforderungen an Ausfallsicherheit genügt und nicht sofort in undefinierte Zustände gerät.

Abseits von der Erprobung genereller Ausfallsicherheit und der Reaktionen darauf innerhalb des Systems ist vor allem das Verhalten der Anwendung bei einem Teilausfall interessant. Zu den Anforderungen an das Verhalten bei einem Ausfall zählen: Wie hoch ist der Verlust der Performance? Wie viele Fehler welcher Art treten auf und wie lange dauert der Übergang zu einem stabilen System?

Deployment

Ein Betriebszustand, der häufig bei Performancetests nicht betrachtet wird, sind die für Services essenziellen kontinuierlichen Deployments. Aus Sicht der Anforderungen stellen sich die folgenden Fragen:

- Sind Fehler während des Deployments erlaubt und wenn ja, welche Fehler und mit welcher Rate?
- Welchen Einfluss auf die Performance des Gesamtsystems darf ein Service-Deployment haben?
- Wie lange darf ein System beeinträchtigt sein, bevor es in den Normalzustand zurückkehrt?

Selbstverständlich muss man beachten, dass Services eigentlich unabhängig sind, deshalb kann es auch dazu kommen, dass mehrere Deployments gleichzeitig laufen. Hier kann es zu kaskadierenden Problemen kommen, die bis zum Ausfall führen können.

Daten

Daten treten als zwei Dimensionen auf. Daten, die als Bewegungsdaten gesehen werden können, die also empfangen, verarbeitet und zurückgesandt werden. Weiterhin bilden Daten auch einen Zustand, wenn der Service diese Daten persistiert.

Kundenservice

Ein Service, der Kundendaten speichert und es erlaubt, diese abzufragen, zu aktualisieren und gegebenenfalls zu löschen, hat zwei Grundanforderungen: die Anzahl der Kundendatensätze im System und die Anzahl der Anfragen. Für die Operationen „Anlegen", „Abfragen", „Aktualisieren" und „Löschen" lassen sich einzelne Anforderungen definieren, die es zu testen gilt. Wichtig ist auch, dass im späteren Betrieb die Operationen gemischt auftreten und damit auch so getestet werden müssen. Es kann hilfreich sein, Messungen zunächst auf einzelnen Dimensionen durchzuführen, um Informationen über die Performance der Basisoperationen zu gewinnen. Bei den Messungen sind auch Einflüsse der Datenlokalität zu beachten, also welche Teilmengen der Kunden werden angefragt, zum Beispiel alle zehn Millionen gleichmäßig verteilt oder mit einer Fokussierung auf 100.000 und in seltenen Fällen die verbleibenden Kunden.

Bilderservice

Ein Service, der Bilder skaliert, ist theoretisch zustandslos. Zu beachten ist, dass Caches jedoch auch einen Zustand darstellen und in der Test- und Anforderungsbetrachtung berücksichtigt werden müssen. Daten für diesen Dienst sind nicht persistent, also sind die reine Anfragegröße der Bilder, die auszuführende Operation sowie die Rückgabegröße wichtig. Dabei kann man zunächst drei Betriebszustände betrachten: die Maximalwerte, den Betriebsdurchschnitt oder die kleinste Operation.

Beispielhafte Anforderung

Dieses Beispiel soll die Überlegungen zu Anforderungen an einen Service oder eine servicebasierte Anwendung illustrieren.

Normalbetrieb

- Durchsatz: 100 Requests pro Sekunde
- Fehlerrate: Keine technischen oder funktionalen Fehler
- Antwortzeiten: P99 150 ms, P99.9 500 ms, maximal 2.000 ms

Der Service soll diese Werte mit der Hälfte der maximal möglichen logischen Einheiten (siehe Kapitel Skalierung) erreichen.

Skalierung

- Maximal: 32 logische Einheiten (16 CPUs pro Einheit, 32 GB Speicher)
- Eine zusätzliche Instanz verbessert die Gesamtleistung um die Leistungsfähigkeit einer einzelnen logischen Einheit
- Fehlerrate: Keine Fehler bei der Skalierung in beide Richtungen
- Dauer: zwei Minuten vom Erkennen des Bedarfs bis Normalbetrieb
- Antwortzeiten während der Skalierung: Antwortzeit P99 500 ms und P99.9 2.000 ms, maximal 4.000 ms

Deployment

- Dauer: maximal zehn Minuten mit der maximalen Anzahl von logischen Einheiten
- Der erreichbare Durchsatz darf sich nicht ändern (100 Requests pro Sekunde)
- Fehlerrate: keine Fehler
- Antwortzeiten: Es gelten die Werte für die Skalierung

Ausfallverhalten

Fällt eine Instanz aus, dann dürfen für eine Minute genau 1/X * Y (X = Anzahl der Instanzen, Y = Gesamtanzahl der Anfragen) fehlschlagen und die Antwortzeiten dürfen für zwei Minuten 25 Prozent höher sein als in der allgemeinen Anforderung beschrieben. Für die nächsten zwei Minuten, um den ausgefallenen Service zu ersetzen, gelten die Anforderungen für die Skalierung. Es dürfen nie mehr als 50 Prozent der maximalen Anzahl logischer Einheiten ausfallen.

Beispiel eines Testvorgehens

Die vorherigen Kapitel diskutieren die Ideen für Anforderungen als Motivation und welche Gesichtspunkte in die Testplanung einfließen sollten. Im Rahmen dieses Artikels kann die Ausführung nicht gründlich diskutiert werden, deshalb sei diese nur als Vorgehensvorschlag erwähnt.

Basisperformance:

Zuerst sollte die Basisperformance bei minimalem Ausbau und ohne Skalierung geprüft werden. Minimaler Ausbau heißt in den meisten Fällen zwei logische Serviceeinheiten, um Redundanz zu gewährleisten. Zunächst wird ohne Parallelität gemessen, um die Baseline (Grundperformance) zu erfassen. Sollten diese bereits über den Zielkriterien liegen, dann kann der Test hier abgebrochen werden.

Basisausbau und Durchsatz:

Im nächsten Schritt sollte man den erreichbaren Durchsatz des Minimalausbaus bei Einhaltung der Zielkriterien ermitteln. Hieraus ergibt sich bereits eine theoretische Hochrechnung, ob die Skalierung es überhaupt ermöglichen kann, Durchsatz und Performancekriterien zu erreichen.

Daten:

Der Einfluss der Datenmenge und -größe sollte vermessen werden, um die Grenzen zu kennen, in denen die Basisperformance gültig ist.

Mindeststabilität:

Immer noch auf der Mindestgröße sollten jetzt erste längere Tests stattfinden, um sicherzustellen, dass Durchsatz und Antwortzeiten stabil sind und keine Fehler auftreten.

Skalierung:

Drei Fragen sollte der Skalierungstest beantworten helfen.

1. Wie skaliert der Service? Was gewinnt man mit mehr Ressourcen?

2. Wann skaliert der Service? Welche Trigger benötigt man und sind diese korrekt?
3. Wie verhält sich der Service, wenn er skaliert, beziehungsweise wie verhält er sich in der kurzen Hochlastphase?

Zieldurchsatz:
Funktioniert die Skalierung, dann kann man prüfen, ob sich der geplante Durchsatz erreichen lässt und ob dabei die geforderten Antwortzeiten erreicht werden.

Stabilität und Robustheit:
Sind die Basisziele erreichbar, dann kann die Langzeitstabilität und das Verhalten bei Störungen geprüft werden. Hier können Methoden des Chaos Testing zum Einsatz kommen.

Überlastverhalten:
Jedes System hat Grenzen, deshalb ist es wichtig zu verstehen, wie sich das System an diesen Grenzen und bei Überschreitung verhält. In den meisten Fällen reicht es aus, wenn das Verhalten vorhersagbar und definiert ist sowie das System von selbst wieder in den regulären Betriebszustand zurückgekehrt.

Deployment:
Essenziell für Microservices sind schnelle und kontinuierliche Deployments. Mit dem Test dieser Deployments unter Last sind das daraus resultierende Verhalten und Auswirkungen auf das Gesamtsystem zu evaluieren.

Performancetests sollten die vorhandenen Logging- und Monitoringtools nutzen, um deren Einsetzbarkeit zu prüfen, und natürlich, um die richtigen Schlussfolgerungen für Verbesserungen zu ziehen. Performancetests ermöglichen auch hier einen ersten Probelauf für den späteren Produktivbetrieb.

Zusammenfassung

Während ein Service selbst oft einfach zu bauen und zu beherrschen ist, ist das Zusammenwirken vieler Services eine komplexe Fragestellung und Herausforderung. Ohne grundlegende Tests der Grundannahmen wie Skalierbarkeit, Zuverlässigkeit und erreichbare Performance können kritische Systeme nicht produktiv genommen werden.

Nachtrag

Microservices und moderne Softwareentwicklung postulieren in vielen Fällen, dass auf Produktion getestet werden soll und gerade die Fähigkeit zu fortlaufenden Deployments hier eine schnelle Fehlerbehebung verspricht. Das schließt einen klassischen Softwareperformancetest vor der Inbetriebnahme oder nach größeren Änderungen allerdings nicht aus. Speziell dann, wenn:

- die Organisation unerfahren mit Services ist,
- neue Technologien zur Anwendung kommen,
- kritische Prozesse ersetzt oder erneuert werden,
- hohe Anforderungen an Durchsatz, Antwortzeiten und Stabilität existieren.

Ein Performancetest kann nicht alle Anwendungsfälle abdecken, sollte aber in der Lage sein, ein Grundvertrauen in den Service oder die Servicearchitektur aufzubauen und damit geschäftliche Risiken zu minimieren.

Dieser Artikel erwähnt einige Testideen, wie das Testen des Einflusses von Daten auf die Performance, nur am Rande, um den Rahmen des Artikels nicht zu sprengen, beziehungsweise lässt Themen wie Sicherheit, Quotas und Multi Tenancy außenvor.

Quellen

[1] „Was sind eigentlich Microservices?" Jaxenter *https://jaxenter.de/was-sind-microservices-40571*

[2] „Was sind Microservices?" Red Hat *https://www.redhat.com/de/topics/microservices/what-are-microservices*

[3] „Software Performance Testing", *https://en.wikipedia.org/wiki/Software_performance_testing*

[4] „Percentile", *https://en.wikipedia.org/wiki/Percentile*

[5] „Was ist Skalierbarkeit?", Autor/Redakteur: Otto Geißler/Ulrike Ostler, 15. August 2019 *https://www.datacenter-insider.de/was-ist-skalierbarkeit-a-852037/*

[6] „What is Chaos Testing?", Ben E. C. Boyter *https://boyter.org/2016/07/chaos-testing-engineering/*

René Schwietzke
Xceptance GmbH
r.schwietzke@xceptance.com

René Schwietzke ist Mitgründer und Geschäftsführer der Xceptance GmbH. Er arbeitet seit ca. 20 Jahren im Bereich Last- und Performancetest und hat viele internationale Kunden und Softwareanbieter unterstützt. Er ist Product Owner für das Xceptance Lasttest-Tool XLT, dass seit 15 Jahren sowohl bei Xceptance als auch bei Kunden erfolgreich im Einsatz ist. Seit 2020 ist XLT Open Source unter der Apache-Lizenz 2.0. Des Weiteren beschäftigt sich René mit Qualitäts- und Performancethemen rund um Programmierung und die Java Virtual Machine.

2. + 3. DEZEMBER 2020

BERLINER EXPERTENSEMINAR

DOAG

NICOLAI PARLOG

Java after eight

KURSÜBERBLICK

In diesem Kurs werden die Java-Versionen 9 bis 15 beleuchtet. Dabei liegt der Fokus auf neuen Sprachfeatures wie Sealed Classes, Records, Text Blocks, switchExpressions und var, aber auch neue und erweiterte APIs sowie JVM- und Performance-Verbesserungen werden theoretisch vorgestellt und mit praktischen Übungen untermauert. Darüber hinaus werden der Migrationspfad von Java 8 zu 11 bzw. 15, der neue Release-Zyklus und die aktuelle Situation um Distributionen und Support besprochen.

ZIELGRUPPE

Erfahrene Java-Entwickler, die sich theoretisch und praktisch auf die neuen Java-Versionen vorbereiten möchten.

VORAUSSETZUNGEN

Einige Jahren praktische Erfahrung mit Java-Entwicklung und sicherere Java-8-Kenntnisse.

www.doag.org/go/expertenseminar_parlog