# Java aktuell

iJUG Verbund
www.ijug.eu

**Topaktuell**
Alle Neuheiten von Java 10

**Aus der Praxis**
Bastlerfreundliche Heim-Automatisierung

**Der neue Trend**
Serverless Java mit Fn Project

# Java ist cool

Ausgabe 04/2018

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 04

adesso business. people. technology.

## SOFTWARE DEVELOPMENT@adesso

# *Von Anfang an Teil des Java-Teams!*

Entwickelt bereits kluge IT-Lösungen bei adesso:
Ihr neuer Kollege Kristof Zalecki | Software Engineer

Sie wollen dort einsteigen, wo Zukunft programmiert wird? Dann sind Sie mit einem Start in unserem Java-Team bei adesso genau richtig. Gemeinsam setzen wir herausfordernde Projekte für unsere Kunden um. Dafür brauchen wir Menschen, die Lust haben, ihr Wissen, ihre Talente und ihre Fähigkeiten einzubringen.

Planen und realisieren Sie in interdisziplinären Projektteams anspruchsvolle Anwendungen und Unternehmensportale auf Basis von Java/JavaScript-basierten Technologien als

**→ (Senior) Software Engineer (m/w) Java**
**→ Software Architekt (m/w) Java**
**→ (Technischer) Projektleiter Softwareentwicklung (m/w) Java**

## CHANCENGEBER – WAS ADESSO AUSMACHT

Kontinuierlicher Austausch, Teamgeist und ein respektvoller, anerkennender Umgang sorgen für ein Arbeitsklima, das verbindet. So belegen wir beim Wettbewerb „Deutschlands Beste Arbeitgeber in der ITK 2018“ den 1. Platz!

Mehr als 650 Software Engineers Java bei adesso, über 120 Schulungen und Weiterbildungen – zum Beispiel in Angular2 oder Spring Boot – sowie ein Laptop und ein Smartphone ab dem ersten Tag warten auf Sie!

GREAT PLACE TO WORK® 2018 Beste Arbeitgeber™ ITK

## IHRE BENEFITS – WIR HABEN EINE MENGE ZU BIETEN:

- Welcome Days
- Choose your own Device
- Weiterbildung
- Events: fachlich und mit Spaß
- Sport-förderung
- Mitarbeiter-prämien
- Auszeit-programm

Es wird Ihnen bei uns gefallen! Mehr Informationen auf **www.karriere.adesso.de**. Olivia Slotta aus dem Recruiting-Team freut sich auf Ihre Kontaktaufnahme: **adesso AG // Olivia Slotta // T +49 231 7000-7100 // jobs@adesso.de**

# Functional Load Testing mit Gatling

*Gerald Mücke, DevCon5 GmbH (Schweiz)*

*Die Hauptaufgabe im Testen besteht darin, Informationen über ein Software-System zu gewinnen; dazu gehören auch Last- und Performance-Tests. Mit funktionaler Programmierung können wir Ereignisse und Szenarien modellieren, die mit klassischen Ansätzen gar nicht oder nur schwer abbildbar sind. Dieser Artikel beschreibt diese Szenarien und zeigt die Möglichkeiten von Gatling auf, sie funktional zu testen.*

Für viele Entwickler besteht Testen in erster Linie aus dem Schreiben automatisierter Unit-Tests. Dies sind jedoch keine Tests im eigentlichen Sinne, denn sie prüfen nur bekannte Bedingungen und Ereignisse ab und werden daher auch „Checks" genannt, die verhindern sollen, dass bekannte oder vorhersehbare Bugs (wieder) auftreten. Testen im eigentlichen Sinne hat daher auch nicht zum Ziel, Bugs zu finden, sondern vielmehr, Informationen über das zu testende System zu gewinnen. Nicht jedes Verhalten ist spezifiziert, nicht jedes unspezifizierte Verhalten ist ein Bug. Ein Bug ist etwas, das jemanden stört, der genug Einfluss hat – Einfluss, eine Entscheidung zu treffen. Informationen sind wiederum essenziell für eine fundierte Entscheidung.

Ein Teilgebiet des Testens ist der Last- und Performance-Test, dessen Zweck es ist, Informationen über die Leistungsfähigkeit eines Software-Systems zu gewinnen. Dabei unterscheidet man zwischen verschiedenen Arten mit jeweils unterschiedlicher Zielsetzung:

- **Soak Testing**
  Ein lang laufender Test mit moderater Last, der vor allem dazu dient, Ressourcen-Lecks wie Memory Leaks zu finden.
- **Stress Testing**
  Ein Test mit Hoch- oder Überlast, um das Verhalten eines Systems im Grenzbereich seiner Kapazität zu ermitteln.
- **Benchmarking**
  Ein Test, um die Unterschiede zwischen Versionen zu erkennen. Dabei geht es vor allem um die Leistungsmessung.

Diese etablierten Praktiken helfen dabei, Informationen über Robustheit, Stabilität oder Leistungsfähigkeit zu gewinnen. Sie sind aber auch ein Werkzeug für eine gezielte Leistungsanalyse mit dem Ziel, Engpässe zu finden und die Leistung des Systems zu verbessern.

## Anatomie eines Last-Tests

Betrachten wir zunächst einmal, wie man einen typischen Last- und Performance-Test entwickelt. Der Last-Test hat drei Hauptbestandteile:

- Seiten-Skripte mit einzelnen Requests pro Seite definieren. Diese lassen sich mit einem Recorder aufnehmen und nach Bedarf nachbearbeiten oder manuell entwickeln.
- User-Szenarien, bestehend aus einer Abfolge von Seiten, mit Verzweigungswahrscheinlichkeiten der Benutzerströme inklusive Eintritts- und Austrittspunkte. Diese Navigationspfade sind oft nicht klar definiert und müssen in Zusammenarbeit mit verschiedenen Stakeholdern ermittelt werden. Das Ziel ist, die Be-

nutzerströme so realistisch wie möglich zu definieren, um möglichst verschiedene Teile des Systems zu belasten.
- Lastprofile definieren die Verteilung der Benutzer über die Zeit, die nachfolgend genauer beschrieben sind.

Für das Lastprofil sind die Qualitätsanforderungen an das System erforderlich. Dabei helfen Anforderungsdokumente oder aber auch historische Daten und Prognosen. Manchmal ist es nötig, verschiedene Stakeholder zu interviewen, um genügend detaillierte Anforderungen zu bekommen.

In einem der Projekte des Autors lautete die spezifizierte Anforderung: „Das System muss in der Lage sein, tausend „concurrent user" mit einer durchschnittlichen Antwortzeit von eineinhalb Sekunden zu bedienen". Das Wesentliche an dieser Anforderung sind nicht die Zahlen, sondern der Kontext, in dem diese Zahlen verwendet werden: „concurrent user" und „durchschnittlich". Wie sich auf Nachfrage herausstellte, waren damit „tausend User innerhalb einer Stunde" gemeint und der Durchschnitt bezog sich auf den „wahrgenommenen Durchschnitt", der in der Literatur häufig der Neunzig-Prozent-Perzentilen entspricht *[1]*, also haben neunzig Prozent der Benutzer eine Antwortzeit von unter eineinhalb Sekunden. Diese Angaben deckten sich wiederum mit historischen Daten.

Anhand dieser Daten würde man nun klassisch ein Szenario entwerfen, bei dem eine Anzahl virtueller User sich durch die Applikation klickt, um das System hinreichend zu durchdringen. Die virtuellen User haben eine bestimmte „Think Time", also eine künstliche Wartezeit zwischen den Requests. Anhand dieser lassen sich virtuelle in reale User umrechnen.

Beim genannten Projekt lag beispielsweise die durchschnittliche Sessionlänge bei 210 Sekunden, mit vier Page Requests pro Session. Damit lag die reale Think Time bei 52 Sekunden. Mit einer Think Time von fünf Sekunden bilden hundert virtuelle User tausend reale User ab. Ein Thread-basierter Lastgenerator wie JMeter benötigte also hundert Threads.

Diese hundert virtuellen User würden so auf eine Stunde verteilt, dass sie das System kontinuierlich beschäftigen. Der Test würde durch eine Ramp-up-Phase eingeleitet, der sowohl die User gleichmäßig verteilt als auch das System aufwärmt. Zweck des Aufwärmens ist es, den Einfluss von nicht-linearem Systemverhalten auf die Messung zu reduzieren, also Verhalten, das sich nicht proportional zur anliegenden Last verändert. Dazu zählen folgende Punkte:

- Just-in-time-Kompilierung und Optimierung
- Befüllen von Caches
- Ressourcen-Initialisierung wie das Laden von Ressourcen oder der Aufbau von Datenbank-Verbindungen
- Füllen von Queues und Puffern auf ein stabiles Niveau

Hat sich das System auf einen stabilen Zustand eingependelt, kann gemessen werden. *Abbildung 1* zeigt die Messung eines elastischen Systems mit konstanter Last. Deutlich zu sehen ist zum einen, wie das System während des Ramp-up schlechtere Minimal-Werte hat, sich also erstmal aufwärmen muss. Auch zu sehen ist, dass das System erst ab etwa vierzig Prozent der Gesamtlaufzeit in einem stabilen Zustand ist, da das System auf die initial steigende Last mit mehreren trägen Skalierungsvorgängen reagiert. Aus Sicht des Endanwenders – und hier kommt der Tester als dessen Advokat ins Spiel – besitzen die Mess-Ergebnisse jedoch nur begrenzte Relevanz.

Im Zeitalter der Cloud sind viele Applikationen als elastische Systeme konzipiert, sie werden also unter Last durch Zufügen weiterer Ressourcen hoch- beziehungsweise bei sinkender Last herunterskaliert, um Kosten und User Experience in einem ausgewogenen Verhältnis zu halten. Ansätze wie Serverless-Computing besitzen sogar keine dedizierten Server mehr und müssen unter Umständen viel häufiger mit einem Kaltstart, also einem nicht-optimierten System zurechtkommen.

Ein weiterer Faktor ist das Nutzerverhalten selbst. Durch die hohe Durchdringung des Internets im öffentlichen Leben ist die potenzielle Zielgruppe sehr groß und vielschichtig und damit schwerer vorauszusagen. Einzelne Ereignisse wie der Black-Friday-Sale oder Online-Werbekampagnen werden durch soziale Medien verstärkt und sorgen für schwer planbare Belastungsspitzen. Aus Perspektive eines Testers lässt sich mit dem klassischen Vorgehen nur schwer die Frage beantworten, wie sich das System in solchen extremen Grenzfällen verhält und wie sich das auf die User Experience auswirkt. Anders formuliert: Wie testet man das nichtlineare Verhalten?

## Auftritt der Statistiker

Zur Abbildung von Verteilungen unabhängiger und unbekannter Ereignisse liefert die Statistik eine Reihe von Verteilungsfunktionen. Die bekannteste ist die Gauß'sche Normalverteilung *[3]*, die Wahrscheinlichkeitsverteilung unabhängiger Ereignisse. Mit deren Hilfe, die mit der in *Abbildung 2* gezeigten Formel berechnet wird, lässt sich eine realitätsnahe Last über einen längeren Zeitraum abbilden,

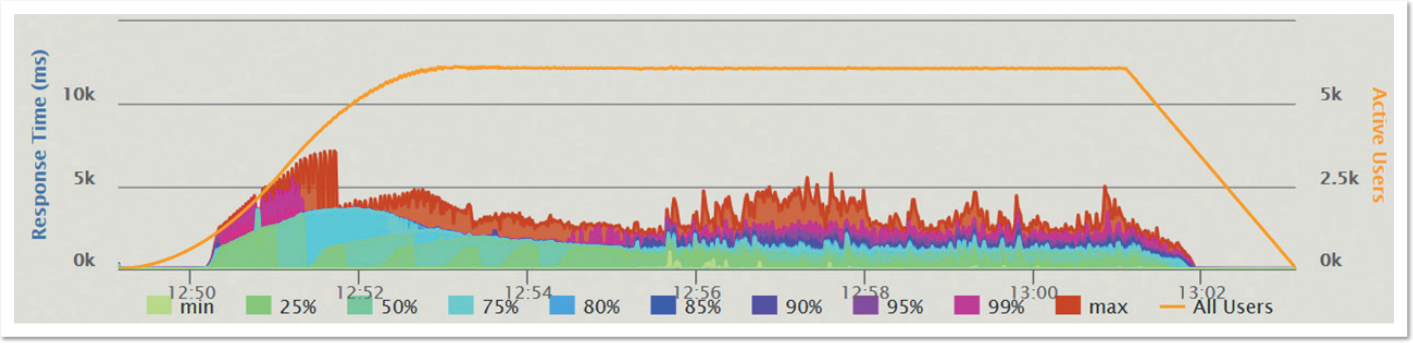


*Abbildung 1: Beispiel Messung mit Ramp-up und konstanter Last*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

*Abbildung 2: Die Gauß'sche Normalverteilungsfunktion*

$$B(k|p,n) = \binom{n}{k} p^k (1-p)^{n-k}$$

*Abbildung 3: Die Binomial-Verteilung*

$$P_\lambda(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

*Abbildung 4: Die Poisson-Verteilung*

```
val page1 = exec(
 http("request_0")
  .get("/")
  .headers(acceptHtml)
  .resources(...))
```

*Listing 1: Seitenskript*

```
val scn = scenario("Simple").exec(page1)
```

*Listing 2: Szenario*

```
scn.inject(
  rampUsersPerSecond(10) to 500 during (5 minutes)
  constantUsersPerSec(500) during (10 minutes)
  )
```

*Listing 3: Lastprofil*

dessen tatsächliche Verteilung unbekannt ist. Als Beispiel sei hier auf die eingangs erwähnte Anforderung von tausend Benutzern pro Stunde verwiesen.

Neben der Normalverteilung sind die Binomial-Verteilung *[4] (siehe Abbildung 3)* sowie die Poisson-Verteilung *[5] (siehe Abbildung 4)*, die einen Grenzfall der Binomial-Verteilung darstellt, als diskrete Funktionen zur Abbildung unabhängiger Zufallsereignisse geeignet. Die Poisson-Verteilung wird dabei häufig zur Modellierung von Ankunftsraten in Warteschlangen-Systemen verwendet, zu denen auch Computersysteme gehören.

Diese statistischen Verteilungen setzen allerdings voraus, dass jedes Ereignis, also die anzutreffenden Benutzer, unabhängig voneinander eintritt. Mit Thread-basierten Lastgeneratoren wie JMeter lassen sich derartige Szenarien nur schwer abbilden. Zum einen sind die Konfigurationsmöglichkeiten sich kontinuierlich verändernder Lasten zu begrenzt, zum anderen leiden Thread-basierte Lastgeneratoren an der sogenannten „coordinated omission" (koordinierte Auslassung); Lastgenerator und das zu testende System bilden also über eine Rückkopplung einen geschlossenen Kreislauf. So sind die Requests, die ein Lastgenerator-Thread für einen virtuellen User absetzen kann, von den Antworten des vorhergehenden virtuellen Users abhängig. Für die Abbildung statistisch unabhängiger Ereignisse ist diese Art der Lastgenerierung also nicht geeignet.

## Gatling

Ganz anders ist die Situation mit Gatling. Es basiert auf Event-getriebener, asynchroner Lastgenerierung. Jeder Request wird unabhängig von vorherigen Ereignissen von einem Timer ausgelöst und es besteht keine Rückkopplung des zu testenden Systems auf den Lastgenerator. Damit lassen sich unabhängige Ereignisse gemäß den beschriebenen statistischen Verteilungsfunktionen, aber auch anderen Funktionen, abbilden.

Ein weiterer wesentlicher Vorteil von Gatling ist, dass die Szenarien in einer Programmiersprache – im Fall von Gatling in Scala – geschrieben sind. Die drei eingangs beschriebenen Komponenten eines Last-Tests sind in Gatling:

- Seiten-Skripte, die eine Abfolge von Requests beschreiben *(siehe Listing 1)*
- Szenarien, bestehend aus seiner Abfolge von Seiten *(siehe Listing 2)*
- Lastprofile, die als Abfolge von „InjectionSteps" definiert werden *(siehe Listing 3)*

```
def continuousUserRate(
duration: FiniteDuration,
     totalUsers: Int,
     distrFun: (Int) => (Double => Double),
     stepFun: (Duration) => (Int) = (d) => d.toMinutes.toInt):
List[InjectionStep] = {
  val steps = stepFun(duration)
  val stepDuration = duration / steps
  def fun(x: Double) : Double =
       totalUsers * distrFun(steps)(x) / stepDuration.toSeconds
  List.range(0, steps).map(
       step => rampUsersPerSec(fun(step)) to fun(step + 1) during stepDuration)
}
```

*Listing 4: Generierung der „InjectionSteps"*

„InjectionSteps“ im Lastprofil beschreiben, wie viele User in welchem Zeitrahmen aktiv sind und dem Szenario folgen. Gatling bietet eine Reihe von Basis-Steps an. Für klassische Lastszenarien sind das unter anderen:

- **rampUsersPerSecond**
  Steigert die Ankunftsrate linear während eines bestimmten Zeitraums
- **constantUsersPerSecond**
  erzeugt in regelmäßigen Abständen neue Benutzer, die am System eintreffen

„InjectionSteps“ werden als Liste an das Szenario übergeben. Diese kann aber auch das Ergebnis eines Funktionsaufrufs wie „def f()= _ => List[InjectionStep]“ sein. Auf diesem Wege lassen sich aus den beschriebenen mathematischen Funktionen Lastprofile für Gatling generieren.

Da Gatling von Hause aus keine funktionsbasierte Lastgenerierung anbietet, sind die mathematischen Funktionen durch ein Mapping auf die Basisschritte anzunähern. Naheliegend ist hier eine Abfolge von „RampUserPerSecond“-Steps, was hinreichend genau für einen Last-Test ist. Die Funktion in *Listing 4* übernimmt das Erzeugen dieser Liste für einen definierten Zeitraum und eine bestimmte Anzahl Gesamtbenutzer. Der dritte Parameter ist die Verteilungsfunktion für eine spezifische Anzahl von Schritten. Der optionale vierte Parameter bestimmt die Länge der linearen Schritte, wobei der Default-Wert bei einminütigen Schritten liegt.

Mit der Funktion aus *Listing 4* lässt sich aus einer beliebigen mathematischen Funktion, wie der Gauß-Verteilungsfunktion aus *Listing 5*, eine Liste von „InjectionSteps“ erzeugen und auf ein Gatling-Szenario anwenden, wie *Listing 6* zeigt.

## Beispiel-Messungen

*Abbildung 5* zeigt das gleiche System, das in *Abbildung 1* mit konstanter Last getestet wurde, mit einer normalverteilten Last. Bei einer effektiv geringeren Zahl von Usern sind die Antwortzeiten höher; insbesondere im Bereich der höheren Last zur Mitte des Tests steigen die minimalen Antwortzeiten mit der Last weiter an. Dementsprechend sind auch die Perzentilen der Antwortzeiten, die für die Definition sinnvoller Anforderungen oder SLAs wichtig sind, höher als bei dem Szenario mit konstanter Last.

Wiederum deutlich verschärfter sieht die Situation bei der Binomial-Verteilung aus *(siehe Abbildung 6)*. Die Gesamtzahl der Nutzer ist gleich wie bei *Abbildung 1*, jedoch sind die Auswirkungen einer schnell steigenden Ankunftsrate gravierender. Die einzelnen Zacken bei den hohen Perzentilen sind das Ergebnis von Garbage Collections, die sich zur Lastspitze zu Timeouts für den User aufschaukeln. Deutlich sind auch die Auswirkungen des Scale-Downs auf die Antwortzeiten ab der zweiten Hälfte zu erkennen.

## Fazit

Mit dem eventbasierten Lastgenerator Gatling lassen sich mit mathematischen Funktionen Lastprofile definieren, die näher an realen Situationen liegen. Die dadurch gewonnenen Erkenntnisse

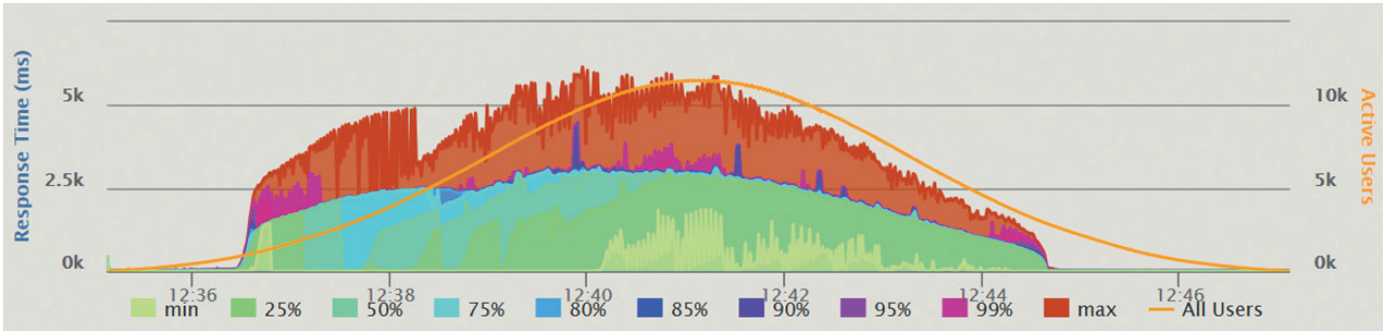


*Abbildung 5: Antwortzeiten bei Gaußverteilung*

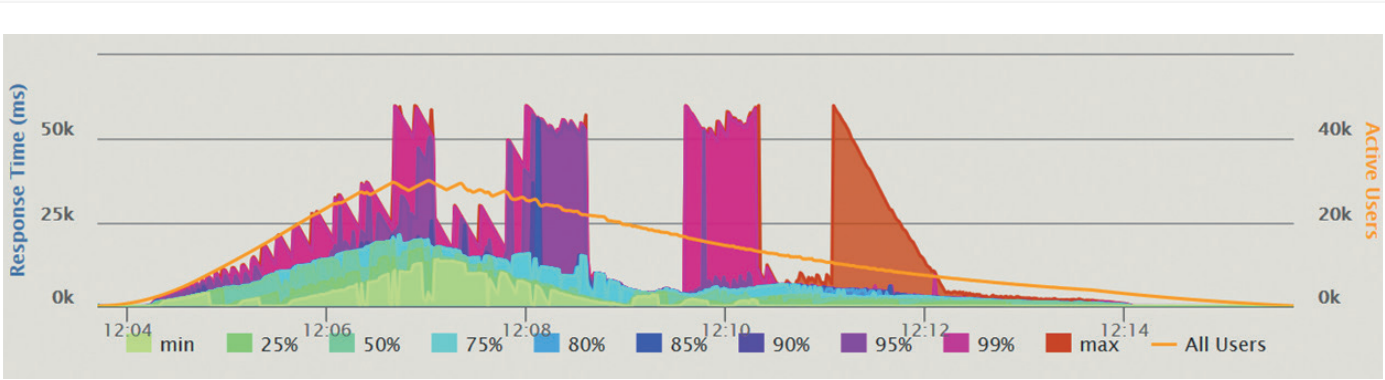


*Abbildung 6: Antwortzeiten bei Binomial-Verteilung*

bieten bessere Rückschlüsse auf die zu erwartende User Experience unter Last als mit herkömmlichen Methoden zur Modellierung konstanter Lasten. Außerdem können mit statistischen Verteilungsfunktionen Szenarien zum Testen elastischer Systeme entwickelt werden.

Wie alle Testpraktiken hat auch das funktionale Load Testing seine Grenzen und ersetzt daher nicht das klassische Vorgehen, sondern ergänzt es vielmehr.

```
def gauss(sigma: Double, mu: Double)(x: Double): Double
= {
  (1 / (sigma * sqrt(2 * Pi)))
  * exp(-0.5 * pow((x - mu) / sigma, 2))
}
```

*Listing 5: Die Gauß-Verteilungsfunktion in Scala*

```
def gaussDistr(sigma: Double = 2, muPercent: Double = 0.5)
              (duration: FiniteDuration, totalUsers: Int) =
  continuousUserRate(duration, totalUsers,
                     steps => gauss(sigma, steps * muPercent))

setUp(
  ExampleScenario.helloWorld
                 .inject(
    gaussDistr()(10 minutes, 30000)
  )).protocols(httpServer)
```

*Listing 6: Erzeugung der InjectionSteps mithilfe der Gauß-Funktion*

## Weitere Informationen


[1] Java Performance, The Definitive Guide, Scott Oaks, 2014
[2] User Experience, Not Metrics, Scott Barber, 2006:
*http://www.perftestplus.com/resources/UENM4.pdf*
[3] *https://de.wikipedia.org/wiki/Normalverteilung*
[4] *https://de.wikipedia.org/wiki/Binomialverteilung*
[5] *https://de.wikipedia.org/wiki/Poisson-Verteilung*
[6] *https://gatling.io/docs/2.3/general/simulation_setup*



**Gerald Mücke**
gerald.muecke@devcon5.ch


Gerald Mücke ist ein Java-Begeisterter mit mehr als zwölf Jahren Branchenerfahrung. Er begann seine Karriere als Entwickler von Testautomatisierungs-Werkzeugen für einen der weltweit führenden Hersteller von Enterprise-Software. Nachdem er in verschiedenen Funktionen, Branchen und Projekten gearbeitet hatte, bietet er heute als unabhängiger Berater mit seiner eigenen Firma DevCon5 (CH) Beratungs-Dienstleistungen in den Bereichen Software-Entwicklung und -Test an. Er interessiert sich insbesondere für Test-Automation, Performance-Testing und DevOps.


ORAWORLD
Das e-Magazine für alle Oracle-Anwender!
EOUC
EMEA ORACLE USERGROUP COMMUNITY
• Spannende Geschichten aus der Oracle-Welt
• Technologische Hintergrundartikel
• Leben und Arbeiten heute und morgen
• Einblicke in andere User Groups weltweit
• Neues (und Altes) aus der Welt der Nerds
• Comics, Fun Facts und Infografiken
Jetzt Artikel einreichen oder Thema vorschlagen!
Bis 9. August 2018
Jetzt e-Magazine herunterladen
www.oraworld.org

# Would you like to develop **Android apps** for more than 8 million unique visitors per month?

# JOIN OUR TEAM!

"Hi, I'm Sami and I work for 1&1 as an Android Software Developer. My team and I are responsible for developing Android applications for our brands GMX and WEB.DE, which are rated as "top developer" apps in the Google Play Store. My work consists of maintaining a clean code, building new features and being up to date with all the technologies of our apps.

Applying agile methods, we mainly use Java and Kotlin to develop our features, which means we can easily react to customer requests. The strong relationships between team members and the knowledge we share with each other is what makes working in our team so special. Are you interested in working with us? We are looking forward to meeting you." **sami@1and1.com**

MEMBER OF united internet

1&1, GMX and WEB.DE are brands of the United Internet AG – a stock-listed company with more than 9,000 employees and an annual sales volume of approximately 4 billion Euro.

DESIGN YOUR CAREER

top EMPLOYER GERMANY 2018 CERTIFIED EXCELLENCE IN EMPLOYEE CONDITIONS

**0721 91374-6891 · www.design-your-career.com**