New Relic

Report

Status quo des Java-Ökosystems 2022

Trends und Daten rund um eine der populärsten Programmiersprachen überhaupt

April 2022

Inhalt

Übersicht

Software ist heute ein so allgegenwärtiger Bestandteil unseres Alltags, wie die Auswahl an Sprachen für ihre Programmierung breit gefächert ist. Zu den populärsten gehört definitiv Java: Die Programmiersprache kommt für die Entwicklung von Software für nahezu jeden Industriezweig und quasi alle Sektoren der Wirtschaft zum Einsatz. Dies zunächst aufgrund ihrer plattformunabhängigen Architektur – in Java geschriebene Software kann bei minimalem Aufwand für unterschiedliche Systeme kompiliert werden. Außerdem stehen für sie tausende Bibliotheken zur Verfügung, und nicht zuletzt zeichnet sich Java auch durch umfassenden Support aus.

Erstmals veröffentlicht im März 2020, bietet unser Report zum Status quo des Java-Ökosystems Einblicke auf Grundlage von Performance-Daten aus Millionen von Anwendungen, die auf der Programmiersprache basieren. Mit dem jüngst veröffentlichten Java 17 steht erstmals seit Version 11 auch wieder ein Release der Programmiersprache zur Verfügung, der durch Long-Term Support (LTS) abgedeckt ist. Ein guter Grund also, unsere Erkenntnisse aus dem Jahr 2020 neu zu bewerten. Auch für die vorliegende Ausgabe des Reports haben wir die Daten wieder anonymisiert und nur auf das Gröbste beschränkt, das zur Gewinnung eines allgemeinen Überblicks für das Java-Ökosystem vonnöten ist. Detailinformationen, die für Cyber-Angriffe oder andere böswillige Aktivitäten von Nutzen sein könnten, haben wir dabei ebenfalls komplett ausgeklammert.

Dieser Report bietet Kontext und Erkenntnisse rund um den Status quo des Java-Ökosystems von heute.

Untersucht haben wir dabei folgende Kategorien:

- Am häufigsten in der Produktion genutzte Java-Version
- Populärste Anbieter
- Zunahme bei der Container-Nutzung
- Am häufigsten genutzte Konfigurationen zur Größe des Heap Space
- Am häufigsten zur Garbage Collection genutzte Algorithmen

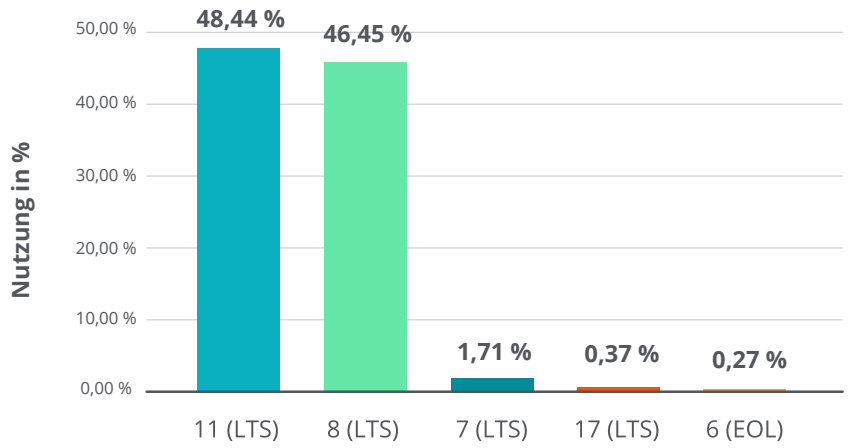
Java 11 – der neue Standard

Java 11 war 2020 zwar bereits seit mehr als einem Jahr verfügbar gewesen. Dominiert wurde das Feld damals jedoch noch klar von Version 8 der Programmiersprache: 84,48 % der Anwendungen basierten damals auf Java 8. Inzwischen haben sich die Mehrheitsverhältnisse zwischen den beiden LTS-Releases jedoch deutlich verschoben. So basieren heute 48 % der in Produktion ausgeführten Java-Anwendungen auf Version 11. Mit diesem deutlichen Zugewinn gegenüber den 11,11 % von 2020 verdrängt sie Java 8 knapp auf den zweiten Platz, das nun nur noch 46,45 % für sich beanspruchen kann.

Java 17 konnte sich dagegen zwar noch nicht bis zur Spitze durchsetzen, dafür aber trotz seiner erst wenige Monate umspannenden Verfügbarkeit bereits jetzt die Versionen 6, 10 und 16 der Programmiersprache hinter sich lassen.

Java 7, für das der Support noch in diesem Jahr eingestellt wird, kommt immerhin noch bei 1,71 % der Anwendungen zum Einsatz. Bei 0,27 % trifft dies auf Version 6 von Java zu, für die der Support bereits ausgelaufen ist. In beiden dieser Fälle gilt jedoch: Die Mehrheit sind um Legacy-Anwendungen, für die keine Upgrades vorgenommen wurden.

Anteilige Nutzung der einzelnen Java-Versionen mit LTS

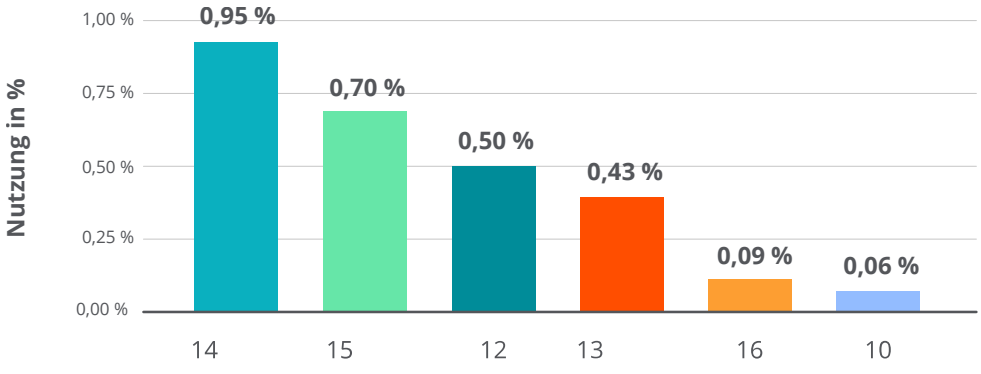


Java 14 – die populärste Version ohne LTS

Die Veröffentlichung von Java 9 wurde begleitet von einer neuen Strategie zur Release-Frequenz der Plattform: Alle 6 Monate wurde fortan eine neue Version der Plattform veröffentlicht, die jedoch nur bis zum Erscheinen des nächsten Release unterstützt wurde. Hierdurch sollten neue Features in kürzeren Abständen verfügbar gemacht werden.

Dies resultierte jedoch nur zwischenzeitlich in einer Zunahme der Nutzung von Java-Versionen ohne LTS: Gegenüber den Versionen mit LTS basieren auf ihnen nur 2,7 % der in der Produktion ausgeführten Anwendungen. Anbieter wie etwa Azul Systems liefern zwar Patches für einige der Versionen ohne LTS aus. Dies ist jedoch eher die Ausnahme, was mit Blick auf diese Versionen auch die geringe Upgrade-Bereitschaft erklären dürfte. So ist Java 14 aktuell die am häufigsten genutzte Version der Programmiersprache ohne LTS. Die Schlusslichter bilden dabei die Versionen 10 und 16.

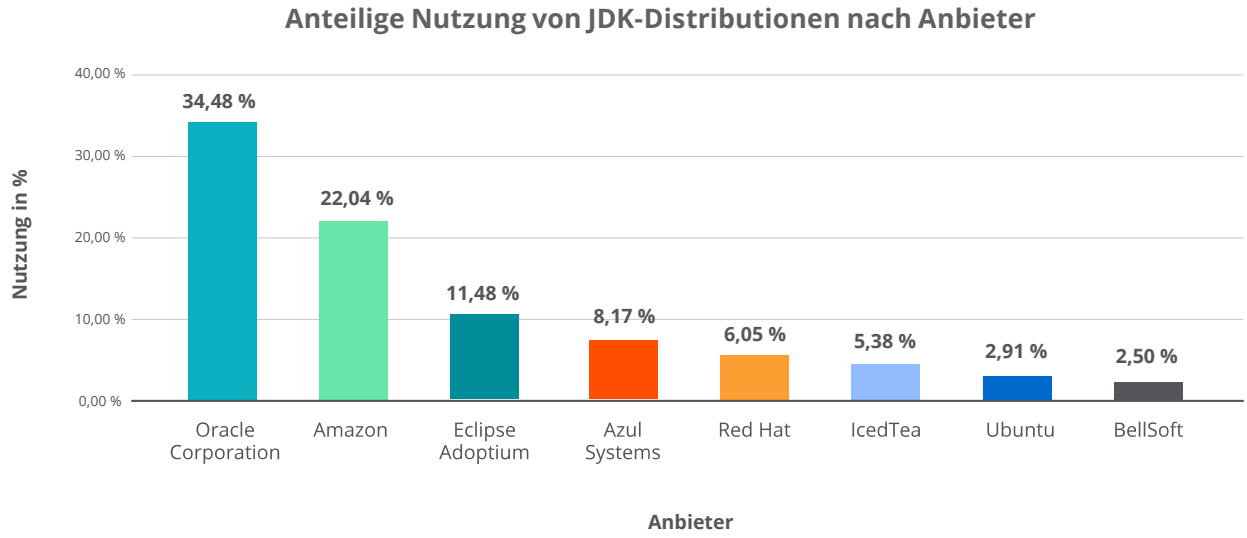
Anteilige Nutzung der einzelnen Java-Versionen ohne LTS



Popularität von Oracle rückläufig, Amazon holt auf

Prägend für die vergangenen Jahre sind Umwälzungen in der Nutzung der Quellen für Distributionen des Java Developer Kit (JDK). Früher noch auf deutlich breiterer Basis eingesetzt, mussten die JDKs von Oracle im Zuge der Vielzahl an Open-Source-Optionen, die im Rahmen des OpenJDK-Projekt rund um Java verfügbar geworden sind, deutliche Einbußen bei ihrer Nutzergemeinde hinnehmen.

Aus dem nachfolgenden Diagramm ist eine Abkehr von den JDKs von Oracle abzuleiten. Ein Grund hierfür dürfte auch die restriktivere Lizenzierungspolitik rund um seine JDK-11-Distribution sein, die Oracle erst im Zuge von Java 17 wieder durch ein offeneres Modell ersetzte. So war Oracle mit rund 75 % Marktanteil im Jahr 2020 der populärste Anbieter von Java-SDKs. An dieser Position rangiert Oracle zwar weiterhin, doch mit gerade einmal der Hälfte des Anteils von damals. Dagegen konnte etwa Amazon erheblich zulegen: Gegenüber 2,18 % im Jahr 2020 liegt sein Marktanteil nun bei ganzen 22 %.



Neben der Abnahme der Nutzung zugunsten der Zahlen aller anderen Anbieter beobachten wir eine weitere interessante Entwicklung, die sich seit November 2021 herauskristallisiert hat: Eclipse Adoptium und Amazon haben seit der Veröffentlichung von Java 17 ihre Ränge mit nahezu exakt den Nutzungszahlen vertauscht, die der jeweils andere Anbieter zuvor verzeichnete.

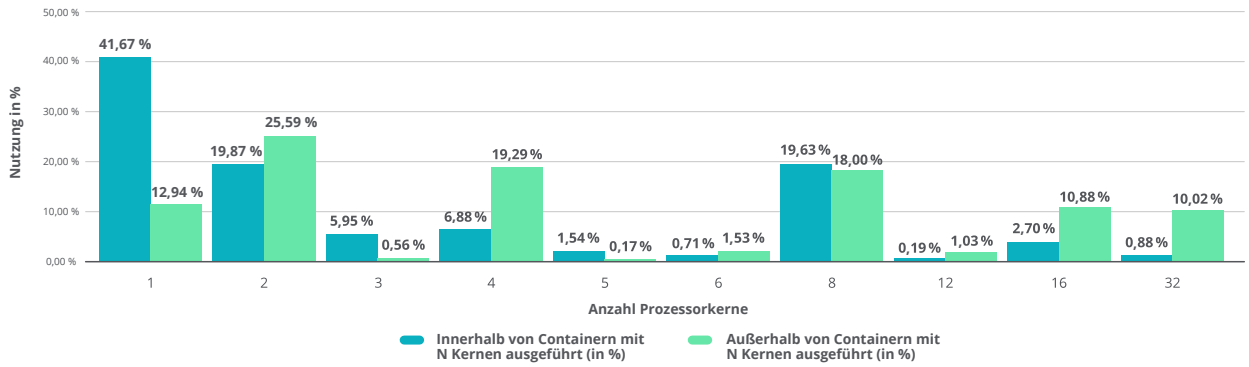
Container als allgegenwärtiges Bild

Die Containerisierung von Anwendungen etabliert sich immer weiter zum Standard – ein Trend, der sich auch in unseren Daten zu Java-Anwendungen deutlich widerspiegelt. So entstammen heute 70 % der Daten, die auf Java basierende Anwendungen an New Relic übermitteln, aus einem Container.

Compute-Konfigurationen in Containern

Container beeinflussen die Nutzungsmuster von Rechen- und Speicherressourcen, so etwa im Hinblick auf die Zahl der zugewiesenen Prozessorkerne: Unseren Daten nach nutzt ein erheblich größerer Anteil der Anwendungen, die in Containern ausgeführt werden, weniger als vier Kerne.

Anteile der innerhalb und außerhalb von Containern ausgeführten Anwendungen nach Anzahl der Prozessorkerne

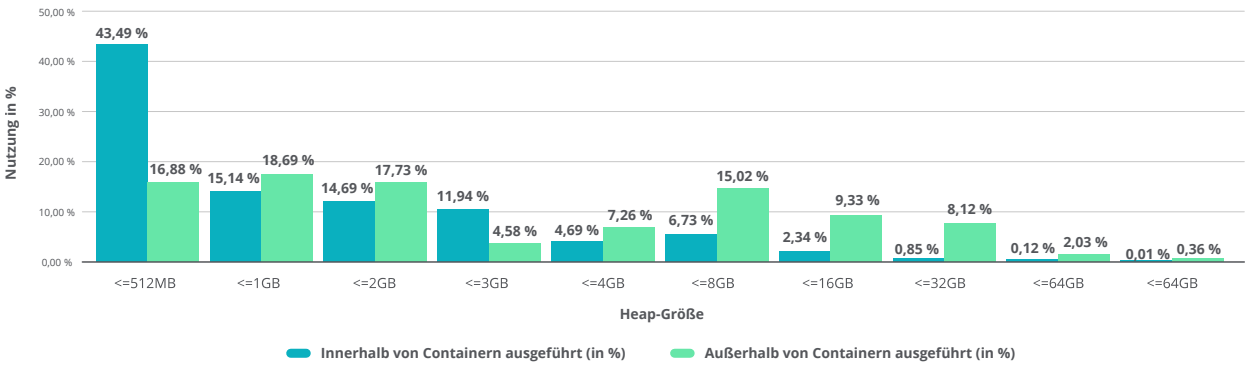


Gerade in Cloud-Umgebungen, in denen Container häufig zum Einsatz kommen, ist es durchaus sinnvoll, die Ressourcennutzung sparsamer zu halten. Bei bestimmten Anwendungen können damit allerdings auch unerwartete Probleme einhergehen. Dies gilt insbesondere für viele der Vorteile der parallelen („concurrent“) Ausführungsmethodik des G1 Garbage Collector: Bei Zuweisung von weniger als zwei Kernen sind diese nicht mehr realisierbar. Denn sämtliche mit nur einem Prozessorkern ausgeführten Instanzen nutzen womöglich auch den seriellen Collector, wofür dann auch die ihm zugehörigen Performance-Kosten anfallen. Vielen ist dies bislang offenbar jedoch noch nicht bewusst.

Speicherkonfigurationen in Containern

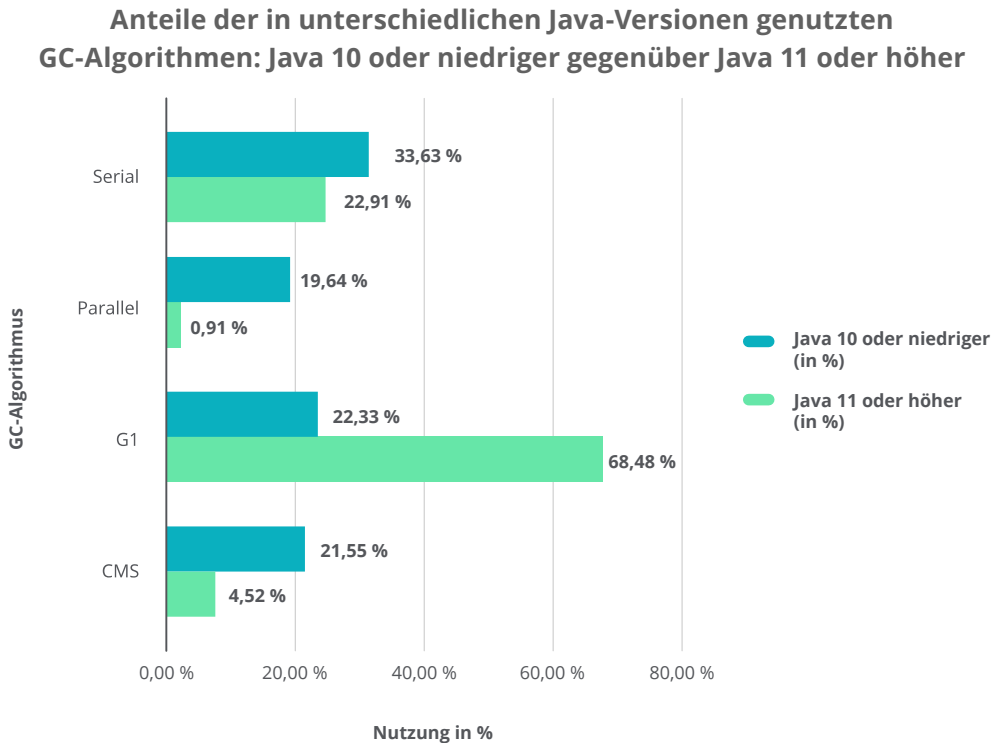
Ganz ähnlich zeigen sich die Trends bei der Zuweisung von Speicherressourcen: Containerisierte Anwendungen sind hier tendenziell sparsamer ausgestattet, wobei gemäß unseren Daten nur für 80 % von ihnen anhand der Flags „-Xmx“ oder „-XX:MaxRAMPercentage“ explizit eine Obergrenze des JVM-Speichers definiert ist. Aufgrund der Features rund um Container Awareness, die Java seit Version 9 für virtuelle Maschinen bietet, ist dies jedoch nicht mehr so wie früher ein Problem für die Sicherheit der Anwendung – vorausgesetzt, die JVM ist der einzige in einem Container ausgeführte Prozess.

Anteil der innerhalb und außerhalb von Containern ausgeführten Anwendungen nach Speicherausstattung gemäß Heap-Konfiguration



Garbage Collectors – wer entsorgt den Abfall?

Garbage Collection (GC) bleibt ob seiner zentralen Rolle für die JVM-Performance auch weiterhin ein viel diskutiertes Thema in der Community. Dabei zeigen unsere Daten: Im Zuge der Veröffentlichung der Folgeversionen von Java 8 haben sich die Nutzungstrends rund um Garbage Collectors deutlich verändert. Unbedingt überraschend ist dies angesichts der Aktualisierungen der Standard-Algorithmen und der gesteigerten Performance, die der seit Java 11 verfügbare G1-Collector bietet, allerdings nicht.



Bei Anwendungen, für die auf Java jenseits von Version 8 gesetzt wurde, ist G1 der klare Favorit. Andere, experimentelle GC-Algorithmen wie ZGC und Shenandoah, die nach Java 8 aus der Taufe gehoben wurden, sind bislang noch weniger von Bedeutung. Dies war jedoch zu erwarten, denn diese gelten erst seit Kurzem als „production-ready“.

Methodik

Die Zahlen in diesem Report wurden allesamt auf Grundlage von Anwendungen ermittelt, die im Januar 2022 Daten an New Relic übermittelt haben. Entsprechend liefern sie kein komplettes Bild der globalen Java-Nutzung. Die Daten wurden anonymisiert und um Details bereinigt, die nicht zur allgemeinen Beurteilung des Java-Ökosystems vonnöten sind. Sämtliche Informationen, die für Cyber-Angriffe oder andere böswillige Aktivitäten von Nutzen sein könnten, haben wir aus diesem Report komplett ausgeklammert.

Über New Relic

Als führender Anbieter von Observability-Technologien unterstützt New Relic die globale Engineering-Community mit einer datenfundierten Methodik für das gesamte Software Lifecycle – von der Konzept-Phase bis zur operativen Umsetzung. In New Relic One erhalten Entwickler:innen die einzige Plattform zur Erfassung sämtlicher Telemetriedaten: Metrics, Events, Logs und Traces. Im Verbund mit umfassenden Analyse-Tools für den gesamten Stack leitet sie New Relic One in kürzester Zeit von einer grundlegenden Situationsanalyse zur genauen Problemursache. Mit dem einzigen klar planbaren verbrauchsbasierten Kostenmodell der Branche setzt New Relic auf absolute Transparenz in jeder Hinsicht und liefert der Engineering-Community so diverse Vorteile: angefangen bei einer effizienteren Cycle-Planung und weniger Ausfallrisiko bei Änderungen über höhere Release-Frequenzen bis hin zu kürzeren Lösungszeiten. Global führende Marken wie AB InBev, Banco International, Chegg, Gojek, Signify Health, TopGolf, World Fuel Services (WFS) und Zalora optimieren so ihre Uptime und Stabilität kontinuierlich und steigern über ihr Kundenerlebnis direkt ihr eigenes Innovationspotenzial und Wachstum.

Setzen Sie noch heute auf Java-Monitoring am Optimum.
Mit dem Java Quickstart für New Relic One.