New Relic

Report

2022 State of the Java Ecosystem

An in-depth look at one of the most popular programming languages

April 2022

Contents

Overview 3

Java 11 is the new standard 3

Java 14 most popular non-LTS version 4

Oracle popularity shrinking, Amazon on the rise 5

Containers run everything around us 5

- Compute settings in containers 5
- Memory settings in containers 6

Garbage in, garbage out 7

Methodology 7

About New Relic 8

Overview

The modern software industry is vast, and there's no shortage of programming languages to choose from. Java is incredibly popular with software developers—and it is used in almost every major industry and economic sector—because it is platform-independent and can move easily from one computer system to another, offers thousands of libraries, and is well supported.

In March 2020, New Relic published its first State of Java Ecosystem Report based on data gathered from millions of applications providing performance data. The recent release of Java 17—the first long-term support (LTS) release since Java 11—provides an opportunity to take a fresh look at that data. To create this report, New Relic anonymized and deliberately coarse-grained the appropriate data to give a general overview of the Java ecosystem. Any detailed information that could help attackers and other malicious parties was deliberately left out of the report.

The goal of this report is to provide context and insights into the state of the Java ecosystem today.

The following categories were examined:

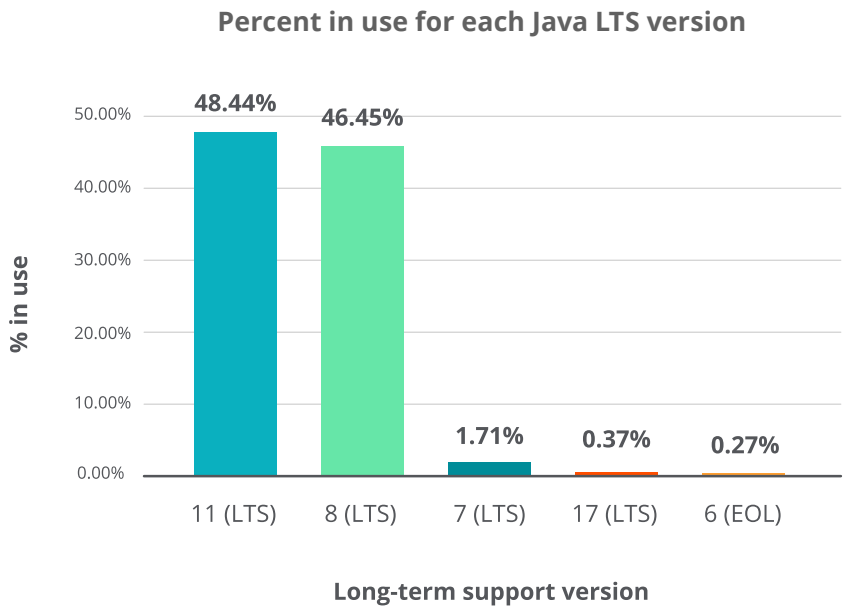
- The most used version in production
- The most popular vendors
- The rise of containers
- The most common heap size configurations
- The most used garbage collection algorithms

Java 11 is the new standard

In 2020, the vast majority of applications remained on Java 8 (84.48%) even though Java 11 had been available for more than a year. Since then, the balance has shifted between these two LTS release versions. More than 48% of applications are now using Java 11 in production (up from 11.11% in 2020) with Java 8 a close second, capturing 46.45% of applications using the version in production.

Java 17 has not climbed the charts, but in the handful of months since its release, it has already surpassed the Java 6, Java 10, and Java 16 releases.

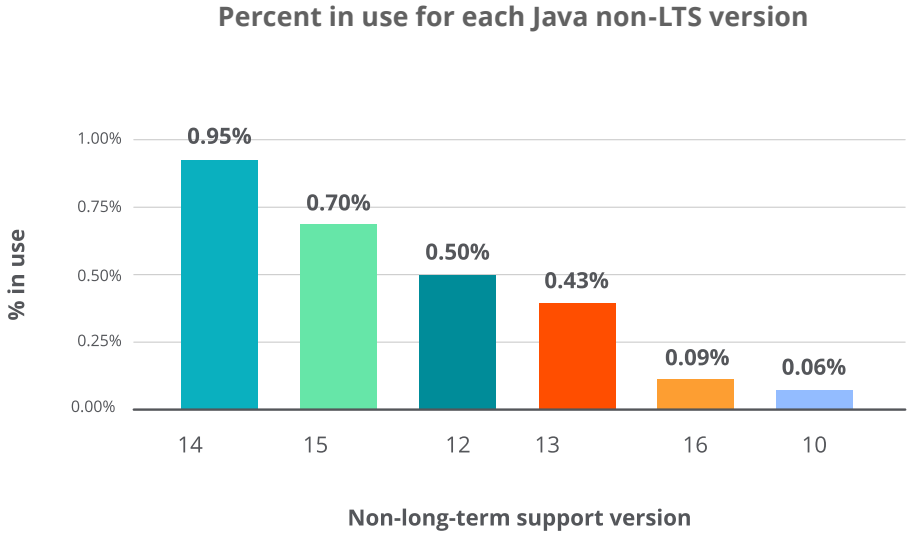
Support for Java 7 is ending in 2022, and yet 1.71% of applications are still using it in production. Meanwhile, Java 6 is no longer supported, but 0.27% of applications are using it. Most of the applications that are using Java 6 and Java 7 are legacy applications that have not been upgraded.



Java 14 is the most popular non-LTS version

Starting with Java 9, the release pattern for the platform changed. Every six months a new version of Java was available, but these versions were only supported until the next release. The intent was to make new features available more often.

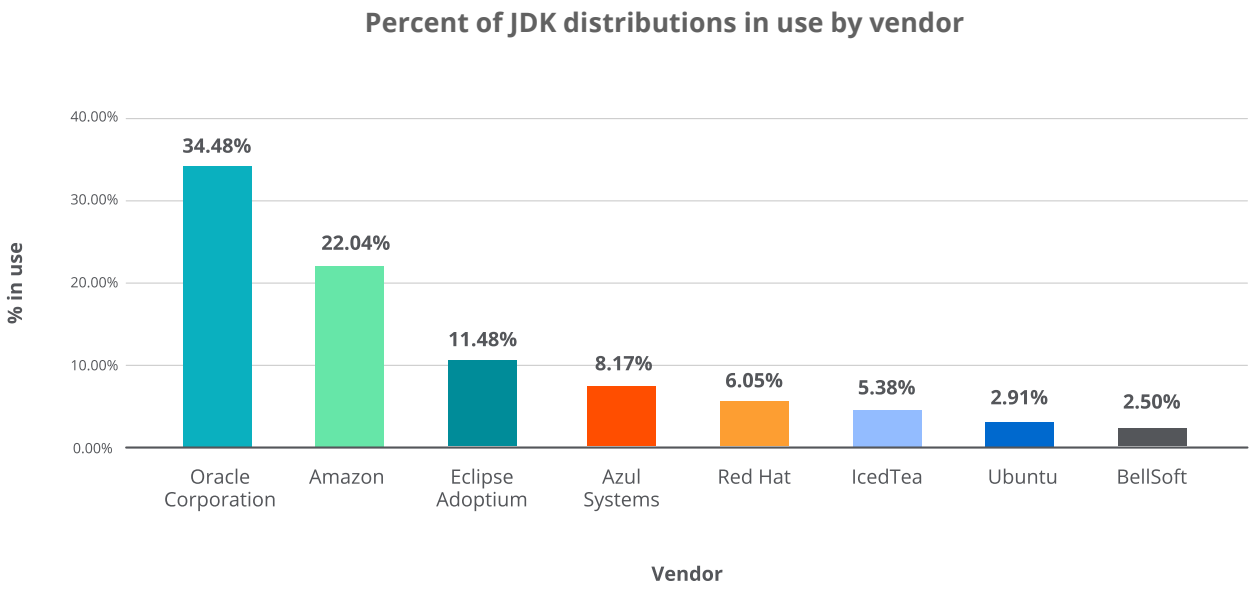
However, uptake for interim, non-LTS Java versions remains extremely low when compared to LTS versions in production with only 2.7% of applications using non-LTS Java versions. While some vendors such as Azul Systems ship patches on some non-LTS versions, most vendors don't. This likely explains the reluctance to upgrade. Out of the non-LTS Java versions in use, Java 14 is the most popular with Java 10 and Java 16 tied for the least popular.



Oracle popularity is shrinking, Amazon is on the rise

Recent years have seen changes in the source of Java Developer Kit (JDK) distributions in use. Where many developers used to get their JDK from Oracle, the open-sourcing of Java in the OpenJDK project has yielded a wealth of options.

The following table shows movement away from Oracle binaries after their more restrictive licensing of their JDK 11 distribution (before their return to a more open stance with Java 17). In 2020, Oracle was the most popular vendor, comprising roughly 75% of the Java market. While they retain the top spot, their share is half what it was. Amazon has climbed dramatically to 22% of the market (up from 2.18% in 2020).



Since November 2021, we've also observed an interesting shift in these numbers beyond the general movement away from Oracle. Prior to the release of Java 17, Eclipse Adoptium and Amazon were in almost identical opposite spots in this list.

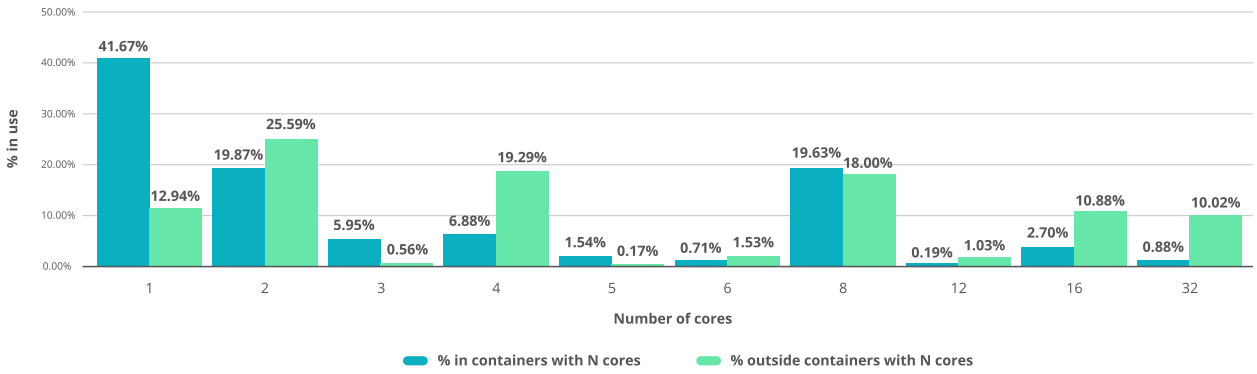
Containers run everything around us

Containerizing applications have become extremely mainstream, and the New Relic Java application data bears out this trend. More than 70% of Java applications reporting to New Relic do so from a container.

Compute settings in containers

Containers impact how people allocate compute and memory resources. For example, the New Relic data shows a much higher percentage of applications running with fewer than four cores when in containers.

Percent of apps running in and outside containers by number of cores

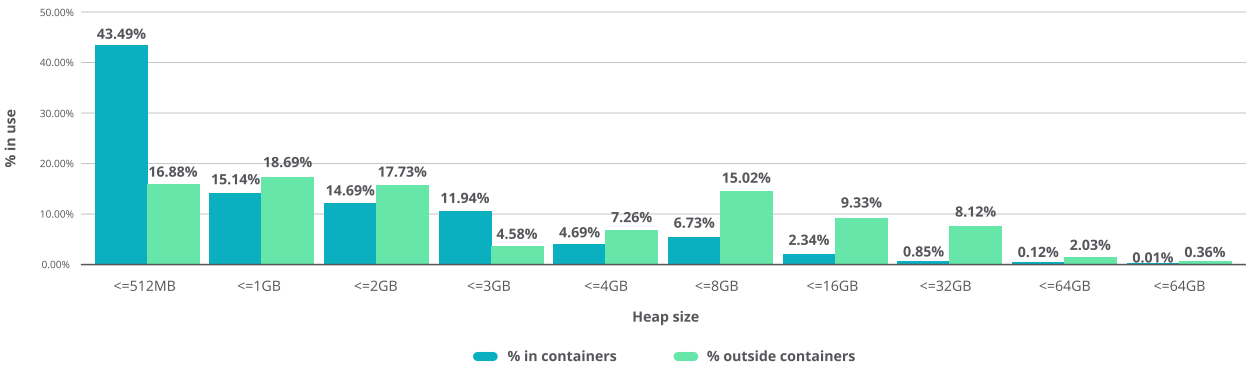


The drive to run smaller makes a lot of sense in cloud environments where people are often deploying containers. But this trend can pose unexpected issues for some applications. In particular, many of the concurrent benefits from the default G1 garbage collector on recent Java virtual machines (JVMs) vanish when running with fewer than two cores. All those single-core instances may as well be using the serial collector—and paying the performance cost of that—but many probably don't even know it.

Memory settings in containers

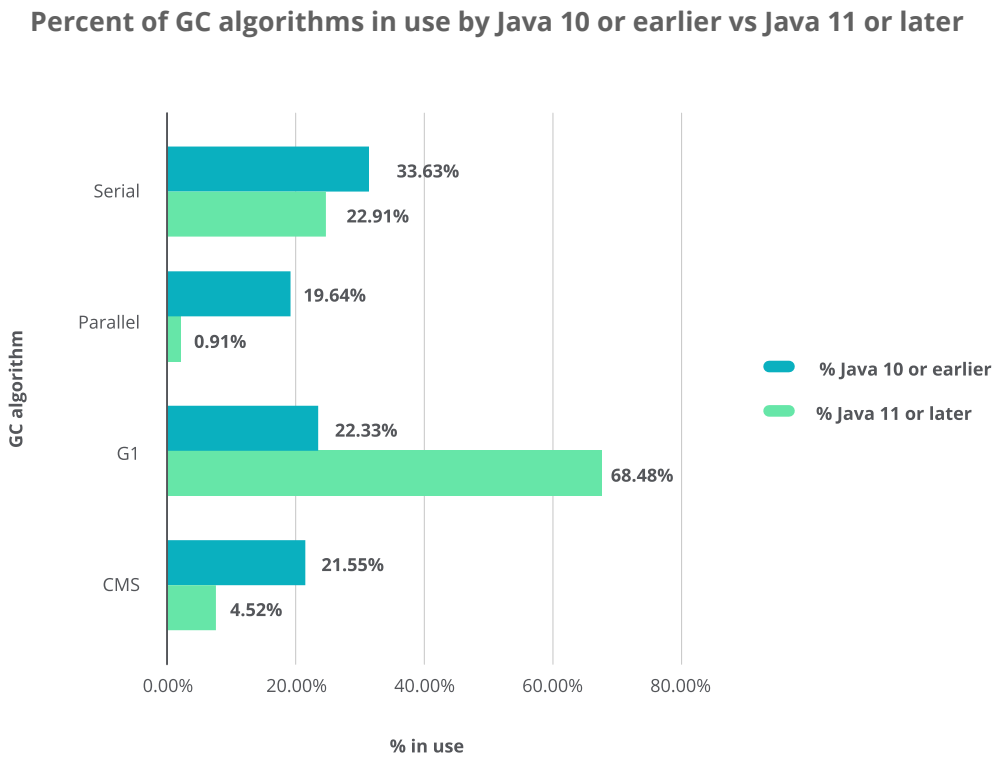
Similar trends surface when comparing memory settings, with a tendency towards smaller instances in containers. The New Relic data shows only about 80% of containerized applications explicitly request an upper bound on JVM memory through either the -Xmx or -XX:MaxRAMPercentage flags. Container awareness features in the JVM, since version 9, mean this probably isn't a safety issue for these applications like it used to be, as long as the JVM is the only process running in each container.

Percent of memory settings heap size running in and outside containers



Garbage in, garbage out

Given its central role in JVM performance, garbage collection (GC) remains a topic of much discussion in the community. New Relic data shows distinct changes to garbage collector usage after Java 8. This isn't surprising given the updated defaults and greater performance available with the G1 collector on Java 11 and later.



G1 is the clear favorite for those who've left Java 8 behind. Other experimental collectors that have appeared post-Java 8 (ZGC and Shenandoah) still show small usage in production systems, but that's to be expected since neither reached production-ready status until recently.

Methodology

The data for this report was drawn entirely from applications reporting to New Relic in January 2022 and does not provide a global picture of Java usage. New Relic anonymized and deliberately coarse-grained the appropriate data to give general overviews of the Java ecosystem. Any detailed information that could help attackers and other malicious parties was deliberately left out of the report.

About New Relic

As a leader in observability, New Relic empowers engineers with a data-driven approach to planning, building, deploying, and running great software. The New Relic observability platform delivers the only unified data platform that empowers engineers to get all telemetry—metrics, events, logs, and traces—paired with the most powerful full-stack analysis tools to help engineers get past the what to uncover the why. Delivered through the industry's only usage-based pricing that's intuitive and predictable, New Relic gives engineers more value for money by helping improve planning cycle times, decrease change failure rates, accelerate release frequency, and reduce mean time to resolution. This helps the world's leading brands including AB InBev, Banco Internacional, Chegg, Gojek, Signify Health, TopGolf, World Fuel Services (WFS), and Zalora improve uptime and reliability, drive operational efficiency, and deliver exceptional customer experiences that fuel innovation and growth.

Start monitoring your Java data today. Install the Java quickstart.