JRebel
by Perforce

10th Edition

2022 Java Developer Productivity Report

Java Development Trends and Analysis

Executive Summary

The 2022 Java Developer Productivity Report is an annual report that showcases the top tools, technologies, and development trends shaping the Java ecosystem. In addition, the report measures how advancement in these tools impacts Java development team productivity.

Table of Contents

Foreword

Dear Colleagues,

Welcome to the 2022 Java Developer Productivity Report by JRebel. It's hard to believe but 2022 marks the 10th year of our annual report series. To say the Java landscape has changed since we started this survey in 2012 is a huge understatement, and I look forward to exploring that with you throughout this report.

Ten years ago, we weren't talking about containers, Docker, Kubernetes, microservices, or even currently popular IDEs like IntelliJ and VisualStudio. We were talking about what we might see in Spring 4, how long it might be until the Java 8 release, and arguing over Java's position as the top programming language (though we are still doing that today).

However foreign the results from that first survey may seem today, it's important to note that these big shifts in Java take time. From one year to the next, the market share for a specific Java technology might only shift a percentage point or two. But those movements — and the living history of the Java ecosystem — can be found and analyzed in the data of surveys like ours.

So what can we point to as the big themes for Java in 2022? Namely that productivity, especially in Java development, will be more critical than ever. With analyst groups like Gartner reporting that 50% of skilled technology roles may go unfilled in 2022, development teams and organizations will share a common goal of getting the most from their limited development hours. Whether that's by adding productivity-boosting tools like JRebel or through other means, organizations will find a way to continue creating innovative new Java applications.

We hope the data and analysis we present in this report will help to make your development decisions easier — whether that's deciding on a new Java framework or adopting a new application architecture — and more informed. Our goal since day one has always been to present a survey unique and specific to Java developers everywhere focusing on the things that are most pertinent to them.

As always, we hope you enjoy reading the report as much as we enjoyed making it.

Happy reading,

Curtis Johnson
Product Manager
JRebel by Perforce

About the Survey

The 2022 Java Developer Productivity Report is based on a survey of Java development professionals around the world. The survey, which ran from October 2021 to January 2022, drew a total of 876 responses.

The survey focused primarily on the Java technologies and approaches used in developing Java applications today. We also included questions specific to performance issues, microservices, and CI/CD, as well as respondent demographics and organization firmographics.

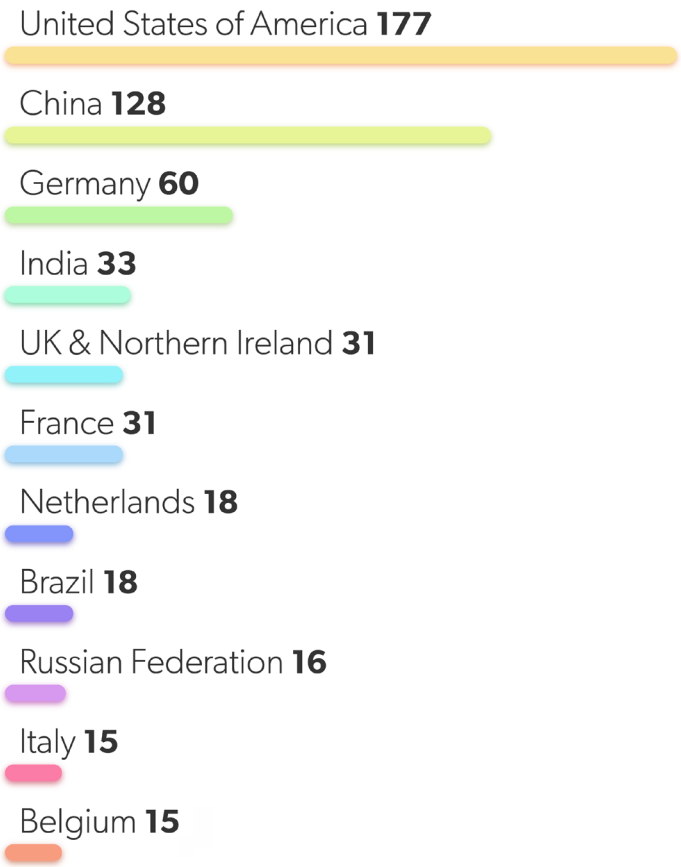
Reporting & Analysis Methodology

For the purposes of this report, we omitted any responses representing less than 1% of the total respondents. For the sake of simplicity in analyzing the results, we rounded each value to the nearest full percentage point. Where beneficial, we used the respondent demographic and organization firmographic data to help form additional insights around the data.

Respondent Demographics and Organization Firmographics

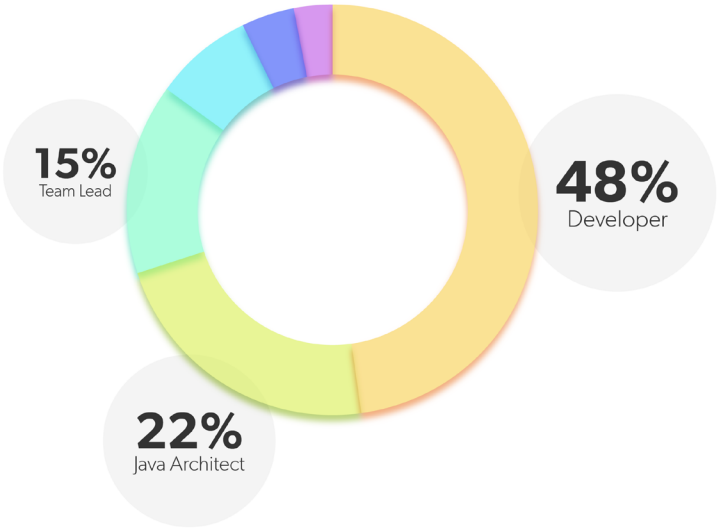
In our first firmographic question, we asked respondents to share the location of their company headquarters. While this isn't indicative of where the bulk of a given company's work is being done (especially with larger companies), it can help us frame the results of the survey within the context of regionality.

Companies were represented from around the world, but the largest contingents (by country) belonged to the Unites States and China.



Job Title

Respondents were then asked to share their job role within their company. Like previous years, the respondents were primarily developers or similar, representing nearly 50% of the overall respondent data. When combined with the second most popular job title among respondents — Java Architect — that number grows to 70% of all respondents. There were also a fair number of leadership roles represented, with team leads representing 15%, and director roles representing 8%.



Developer **48%**

Java Architect **22%**

Team Lead **15%**

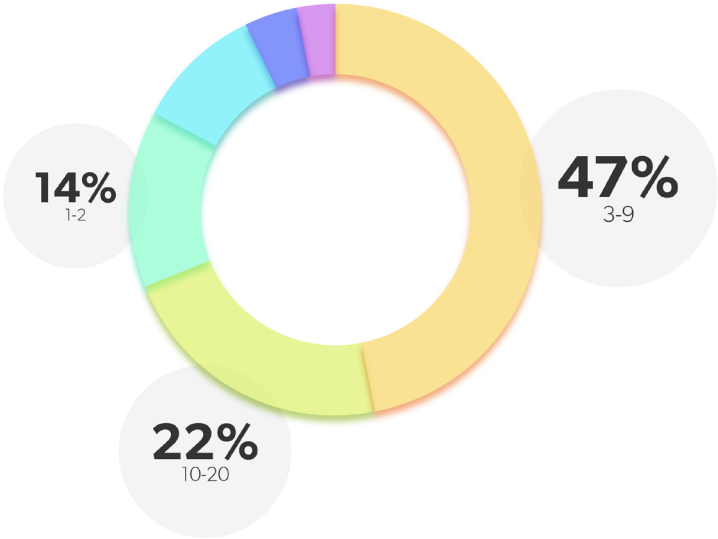
Director/VP/C-Level **8%**

Other **4%**

Consultant **3%**

Development Team Size

Next, we asked respondents to share the size of their development team, with options provided that ranged from 1-2, to 100+ developers. By far the most common development team size was in the 3-9 developer range, which represented nearly half of all respondents. Teams of 10+ represented nearly 40% of respondents, with 22% of the total in the 10-20 developer range. This is expected, as companies have trended more towards dividing teams based on feature sets.



3-9 **47%**

10-20 **22%**

1-2 **14%**

21-50 **10%**

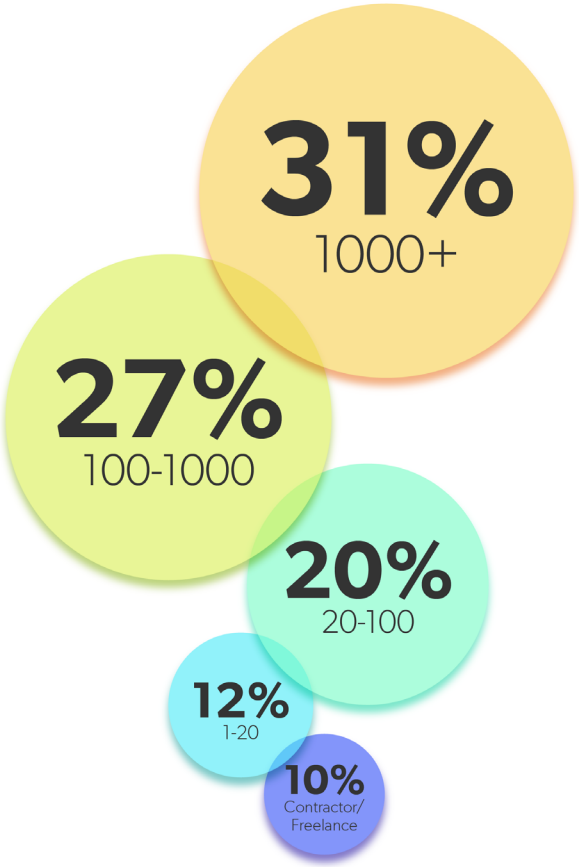
100+ **4%**

51-99 **3%**

Company Size

For our last firmographic question, we asked respondents to share the estimated size of their company, with size broken down into teams of 1-20, 20-100, 100-1000, 1000+, and freelance categories.

Respondents reported that most of their companies were large, enterprise companies, with 31% reporting a company size of over 1000 employees. Mid-sized companies with 100-1000 employees were the next highest group, with 27%. Small companies and startups were represented at 20% and 12%, respectively.



Enterprise company (1000+) **31%**

Mid-sized company (100-1000) **27%**

Small company (20-100) **20%**

Startup (1-20) **12%**

Contractor/Freelance **10%**

Survey Results

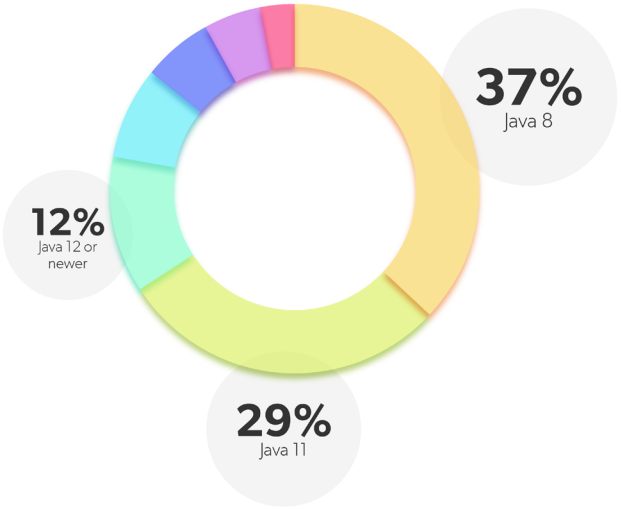
With the demographic questions out of the way, we jumped into the heart of the survey: Java Language and Development trends. These questions also appeared in previous versions of the Java Developer Productivity Surveys, allowing us to judge the adoption trajectory of a few key Java technology versions.

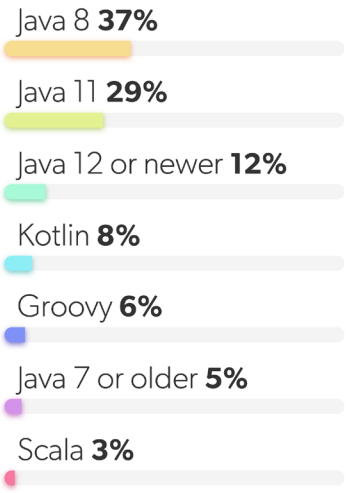
Java Language and Development Trends

For the first question of this section, we asked respondents to share which JDK programming language they're using in their main application, with options that grouped Java versions into Java 7 or older, Java 8, Java 11, and Java 12 or newer. We also included the JVM-compatible languages Kotlin, Groovy, and Scala.

As with many previous iterations of this report, most respondents reported using Java 8 (37%) as their programming language on their primary application, followed by Java 11 (29%), Java 12 or newer (12%), and Java 7 or older (5%). While Kotlin, Groovy, and Scala were the least favored among these choices, combined, they only represented 17% of the total respondents.

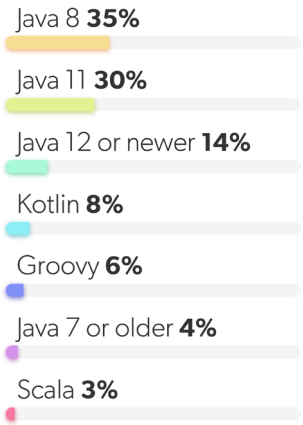
Which JDK Programming Languages are You Using in Your Main Application?



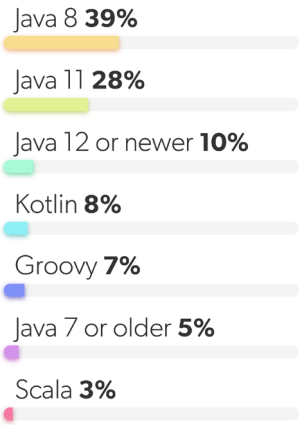


When looking at responses based on company size, Java 8 usage was lower among companies with under 100 employees, while they showed increased adoption of Java versions 11 and newer. Meanwhile, companies with over 100 employees showed more usage of Java 8 when compared to smaller companies.

Companies With Under 100 Employees



Companies With Over 100 Employees

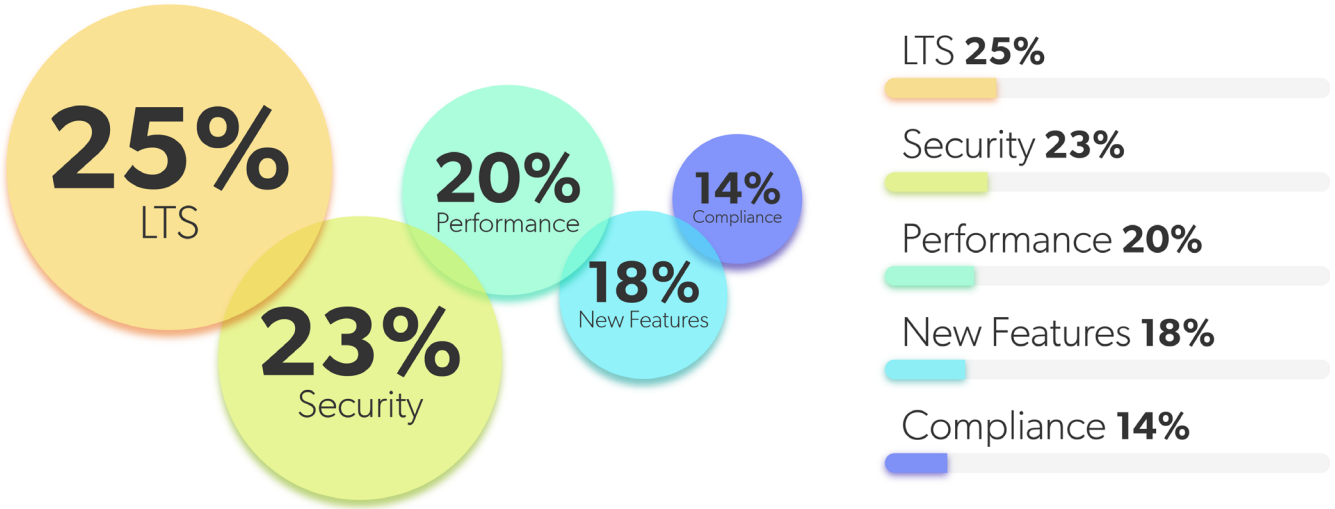


Upgrade Influences

Next, we asked respondents to share the factors that influence their decision to upgrade JDK versions. The answers were largely as expected, with the majority of respondents reporting long-term support as the predominant factor for upgrading JDK versions. After LTS, security and performance were the top factors, at 23% and 20% respectively.

New features (18%) and compliance (14%) were the least popular factors for upgrading.

Which Factors Influence Your Decision to Upgrade JDK Versions?



In segmenting the results by company size, we found that both large and small companies listed LTS as their primary influence on upgrading JDK versions.

These factors have shifted significantly in the past few years, mostly due to modifications to the way Oracle distributes the JDK. With JDKs being released more regularly without LTS, this feature will be something people are more actively looking for when deciding on a JDK.

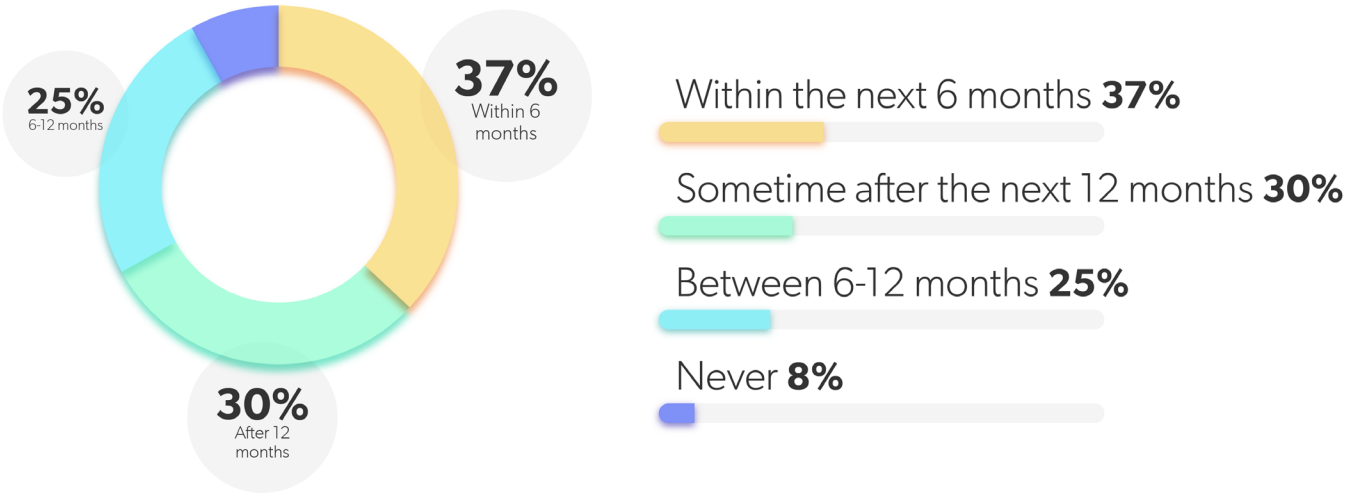
JDK 17 Upgrade Plans

2021 marked the release of the latest LTS Java release with JDK 17. While not as impactful to the overall market as Java 8, it does seem to have some momentum behind it that outpaces other post-Java 8 LTS releases. To help put some stats behind that perceived momentum, we asked respondents to weigh in with when they plan on upgrading to JDK 17.

Among the respondents who knew of plans to upgrade, 37% reported plans to upgrade to JDK 17 within the next 6 months, with a further 25% planning to upgrade within the next 6-12 months. 30% reported plans to upgrade after the next 12 months, while only 8% reported no intention of upgrading to JDK 17.

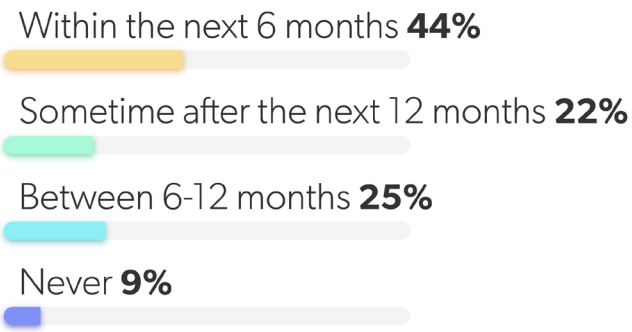
When combined, 62% of those with knowledge of their upgrade plans reported their intention to move to JDK 17 within the next 12 months.

When Will You Upgrade to JDK 17?

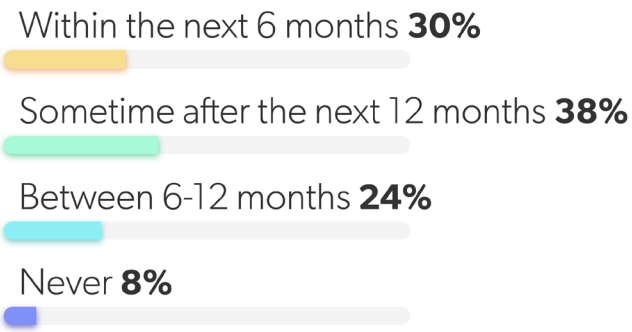


In looking at responses based on company size, respondents from smaller companies (under 100) showed an increased pace of adoption, with 44% planning to upgrade to JDK 17 within the next six months. Respondents from larger companies (over 100) showed a slower pace of adoption, with 38% planning to upgrade sometime after the upcoming 12 months.

Companies With Under 100 Employees



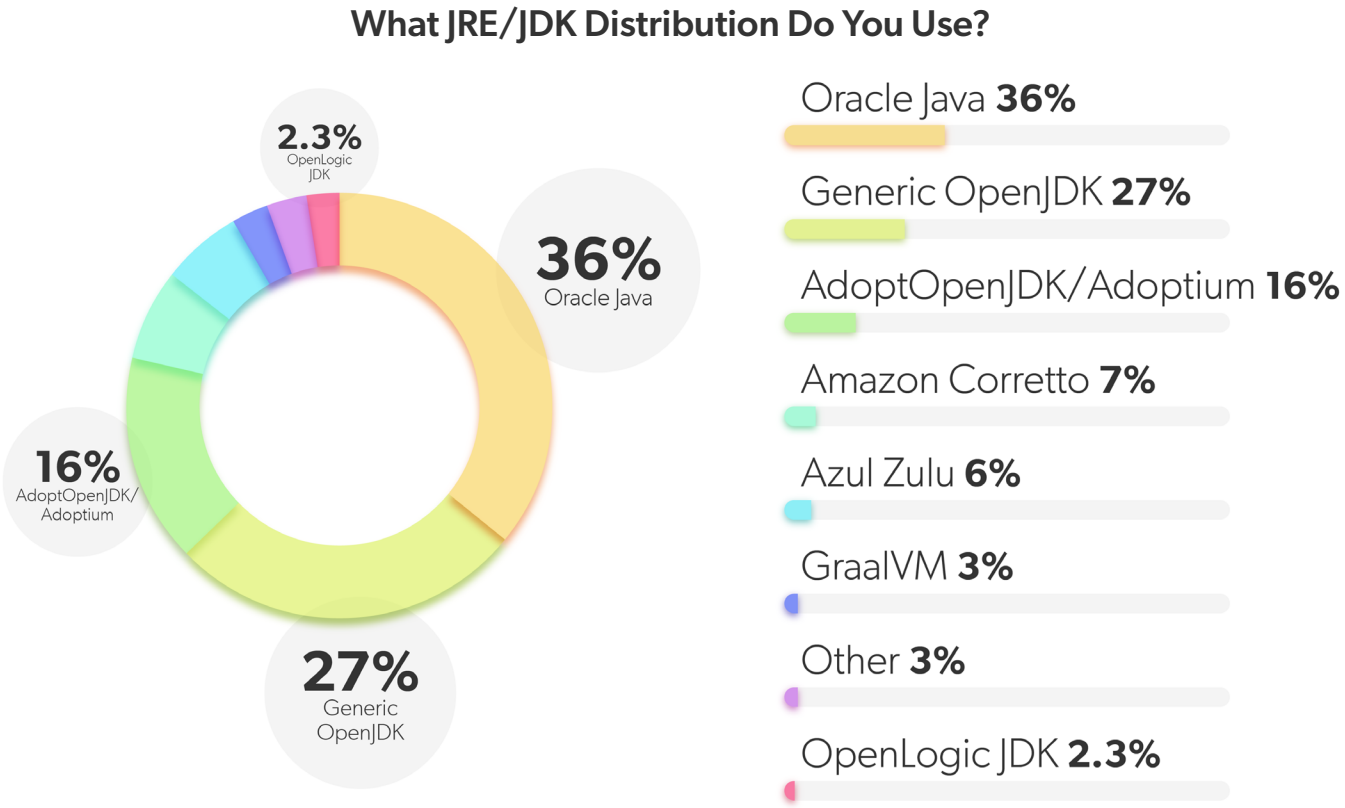
Companies With Over 100 Employees



LTS JDK versions always draw some level of adoption. But JDK 17 seems to be establishing more momentum than previous LTS released (aside from JDK 8) — especially within smaller companies. The slower adoption of new Java versions, including JDK 17, is indicative of the complexity and cost of upgrading large, enterprise Java applications.

JRE/JDK Distributions

Our next question asked respondents to share which JRE/JDK distributions they use. Among this year's respondents, Oracle Java was the distribution of choice at 36%. Generic OpenJDK and AdoptOpenJDK / Adoptium rounded out the top three at 27% and 16%, respectively. **OpenLogic** distributions of OpenJDK had 2.3% representation among survey respondents.

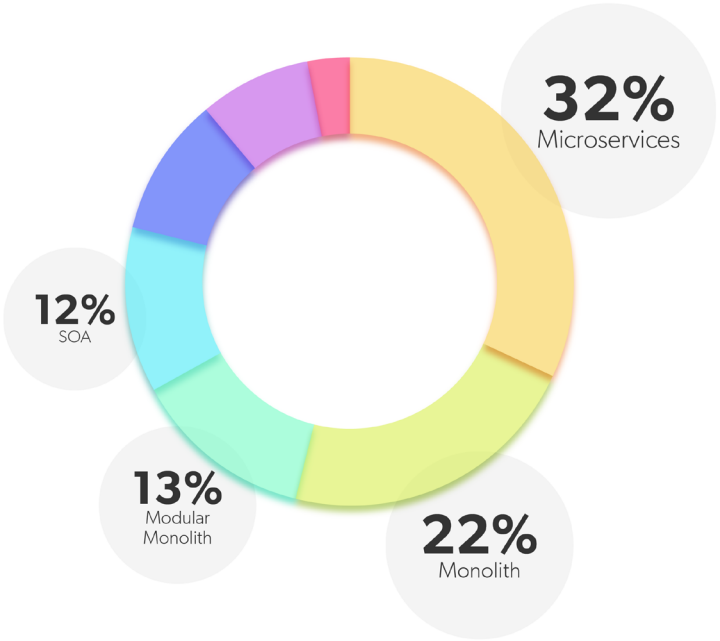


Given the large representation from larger companies in this survey, it's not surprising to an even split between commercial and open source JRE/JDK distributions. Commercial distributions like Oracle Java can give larger organizations an easy way to source patches and updates. For many large organizations, avoiding the hassle of doing that work in-house is worth the price tag.

Java Application Architecture Trends

In our next question, we asked respondents to share the architecture of the primary application they develop. Microservices-based applications were the most popular at 32%, with Monolithic applications at 22%. Next, Modular Monolithic applications made up 13% of responses, while service-oriented architectures came in at 12%.

What Is the Architecture of the Main Application You Develop?



- Microservices **32%**
- Monolith **22%**
- Modular Monolith **13%**
- SOA **12%**
- Desktop App **10%**
- Serverless **8%**
- Other **3%**

Interestingly, larger companies (100+ employees) showed increased adoption of microservices at 36% — a stark contrast to smaller companies (under 100 employees) at 28%.

Companies With Under 100 Employees

- Microservices **28%**
- Monolith **22%**
- Modular Monolith **13%**
- SOA **12%**
- Desktop App **14%**
- Serverless **8%**
- Other **3%**

Companies With Over 100 Employees

- Microservices **36%**
- Monolith **22%**
- Modular Monolith **13%**
- SOA **12%**
- Desktop App **7%**
- Serverless **8%**
- Other **2%**

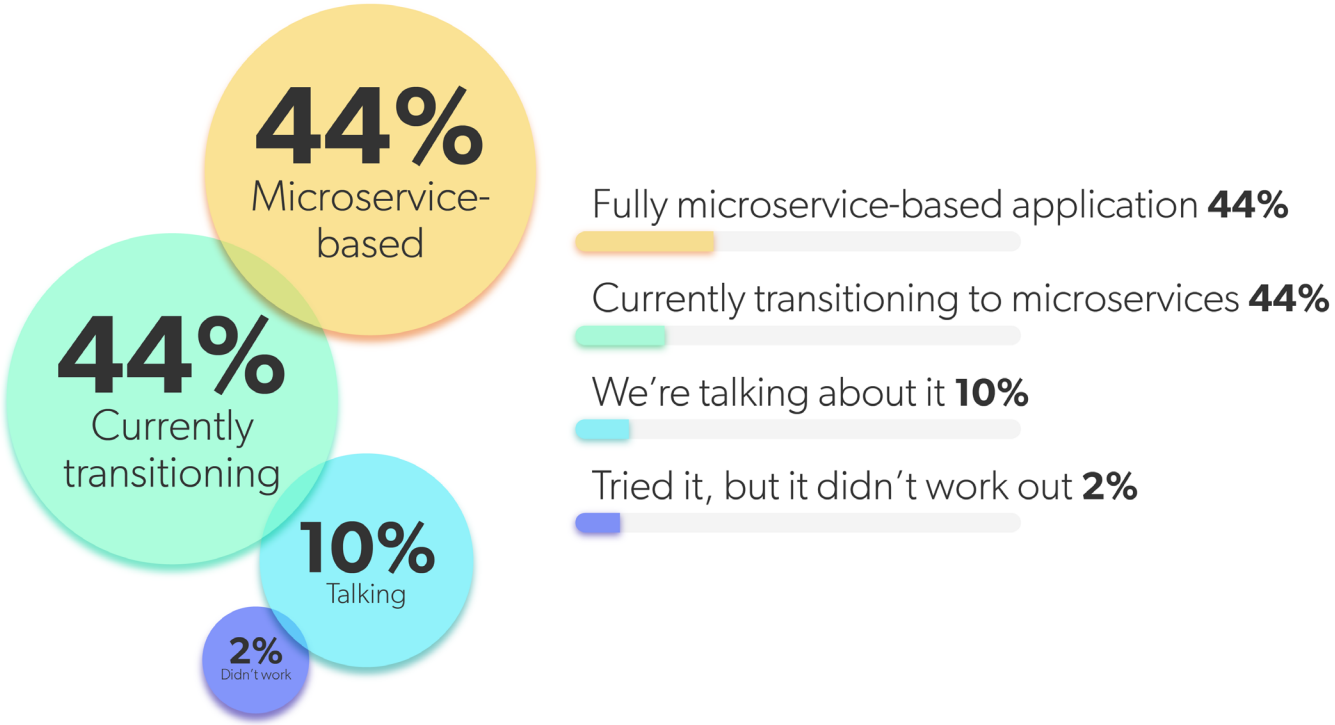
Microservices Trends

For respondents who reported using microservices, we asked a series of questions related to their status in adopting microservices, how they're using microservices, the number of microservices in their application, and choice in microservices framework. Lastly, we asked about the start-up and redeploy times for their microservice-based applications.

Microservice Adoption Status

In this question, we asked respondents to share their status for microservice adoption. The responses showed that most organizations either had fully microservice-based applications, or were currently transitioning to a microservices architecture.

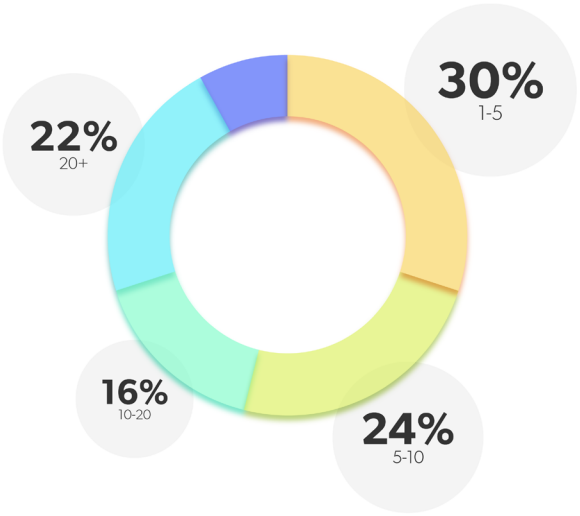
What Is Your Status for Microservice Adoption?



Microservices Per Application

Next, we asked about the number of microservices that comprise the respondents' microservice-based application. 30% reported having 1-5 microservices, while 24% reported having between 5-10 microservices. At the higher end of the scale, 16% reported having 10-20 microservices, and 22% reported having more than 20 microservices in their application. Only 8% reported not having insight into the number of microservices used in their application.

How Many Microservices Do You Have in Your Primary Application?



- 1-5 microservices **30%**
- 5-10 microservices **24%**
- 10-20 microservices **16%**
- 20+ microservices **22%**
- Not sure **8%**

Smaller organizations (under 100 employees) reported applications with fewer microservices in general, with only 29% reporting applications with over 10 microservices. Larger organizations (over 100 employees) while 43% of larger organizations reported applications with over 10 microservices. That number jumps further when looking at organizations with over 1000+ employees, with 50% reporting applications with over 10 microservices.

Companies With Under 100 Employees

- 1-5 microservices **41%**
- 5-10 microservices **24%**
- 10-20 microservices **16%**
- 20+ microservices **13%**
- Not sure **6%**

Companies With Over 100 Employees

- 1-5 microservices **25%**
- 5-10 microservices **24%**
- 10-20 microservices **15%**
- 20+ microservices **28%**
- Not sure **8%**

Companies With Over 1000 Employees

- 1-5 microservices **22%**
- 5-10 microservices **18%**
- 10-20 microservices **15%**
- 20+ microservices **35%**
- Not sure **10%**

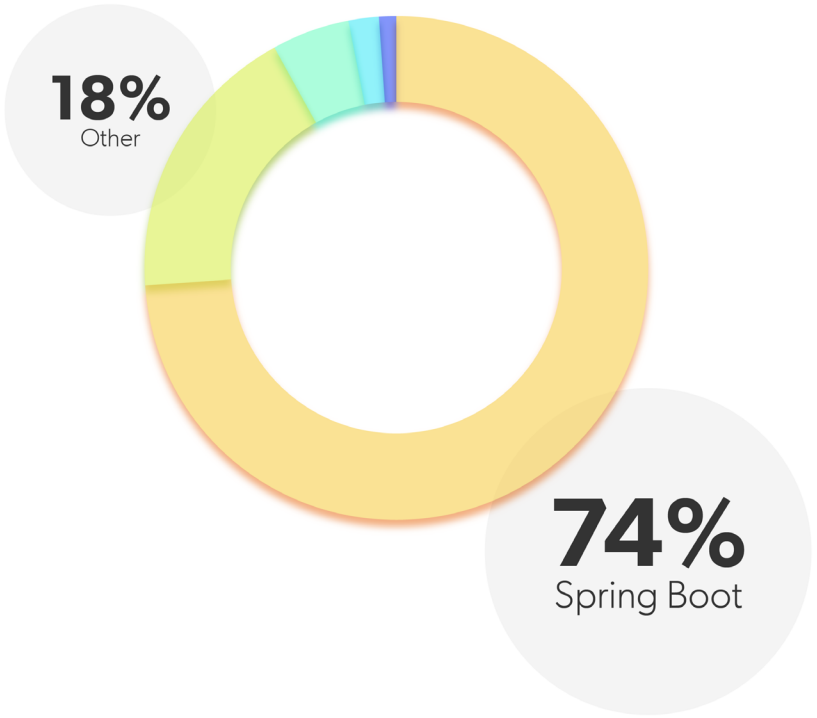
When you consider that only 30% of the respondents from this survey work in enterprise organizations, the number of respondents reporting large microservice-based applications seems to be accurate. Still, it's clear that most organizations are running applications with under 10 microservices. This could mean a few things. Either they're using microservices against the grain, or they're working with relatively small applications.

Microservices Framework Usage

In our next question, we asked respondents to share their top choice in microservice application framework. Not surprisingly, Spring Boot remained the top microservice application framework at 74%. Quarkus, Vert.x, and DropWizard rounded out the top four at 5%, 2%, and 1% respectively.

Common frameworks in the "Other" category included enterprise application specifications like Java EE and Jakarta EE, as well as the vanilla Spring framework. However, the most common "Other" response was that they weren't using a microservices framework at all.

What Microservice Application Framework are You Using on Your Main Project?



Spring Boot **74%**

Other **18%**

Quarkus **5%**

Vert.x **2%**

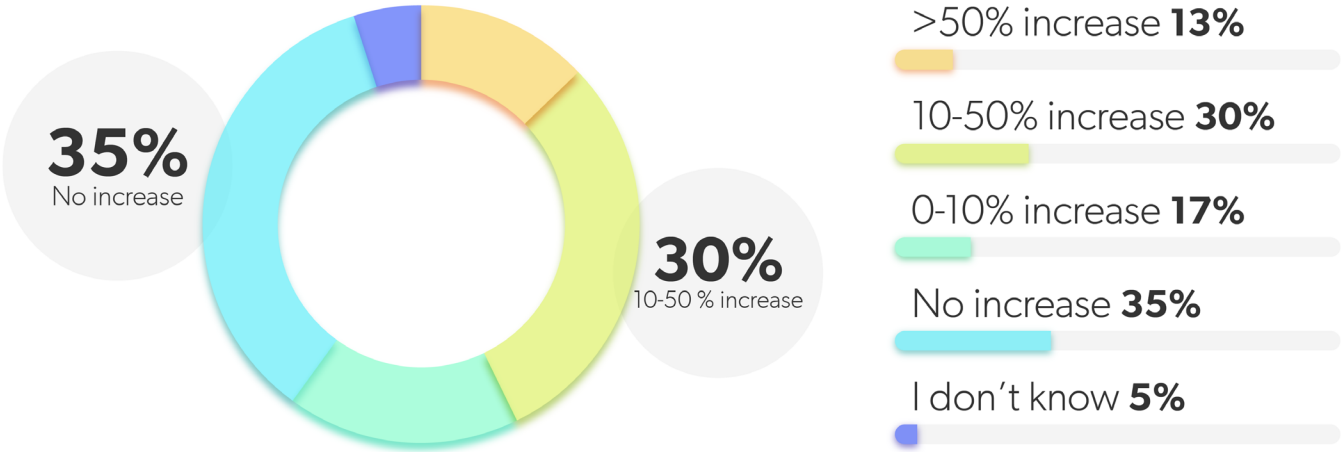
DropWizard **1%**

Microservice Application Start-Up Times

In this question, we asked respondents to estimate the percentage increase in start-up time their microservice-based application has experienced since the time it was created. Our survey found that a combined 60% had experienced an increase, while 35% had not.

Alarmingly, 13% of respondents reported over a 50% increase in the startup time for their microservice-based application since the time it was created.

Have You Experienced an Increase in the Time it Takes to Start Up the Services in Your Microservice Application Since the Original Transition/Creation of the Microservice?



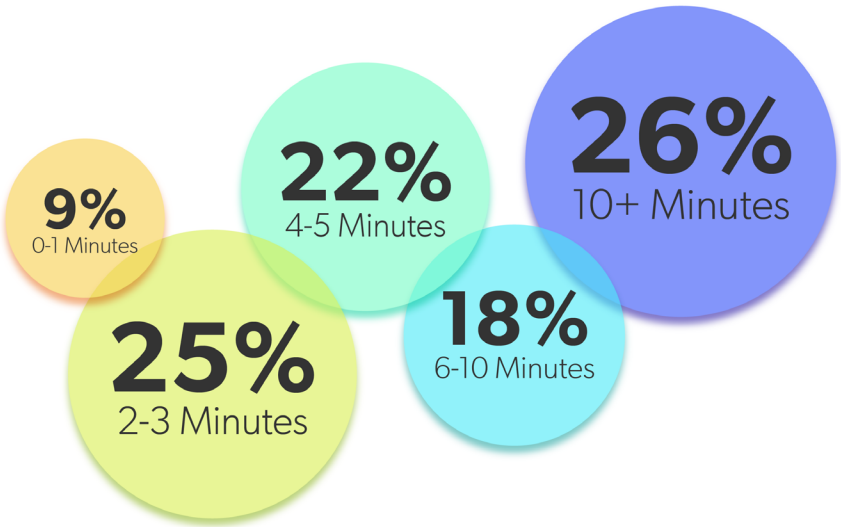
Why the increase? The truth is there is a large variety of different reasons why companies are seeing this increase in start-up times. As microservices application gradually gain modules and as development teams build their applications beyond the initial creation of the application, companies are running into increased start up times much more similar to that shown in the monolithic applications. This being said, most companies decide to make the transition to microservices due to other benefits beyond start-up times and the fast initial start-up times was just an extra bonus experienced with young versions of these applications.

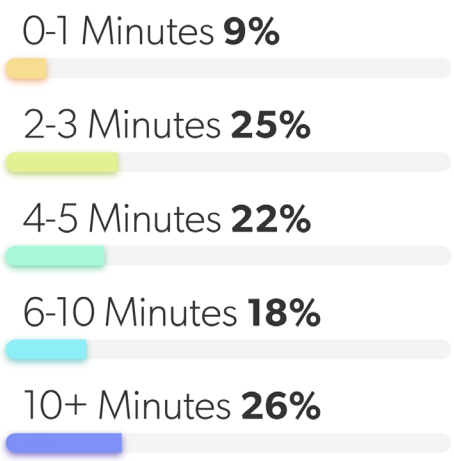
Microservice Application Redeployment Times

In this question, we asked respondents to weigh in with the time it takes to deploy their containerized environment.

Our survey found 44% of respondents reporting a 5+ minute deployment time, with 26% of respondents reporting a deployment time of over 10 minutes.

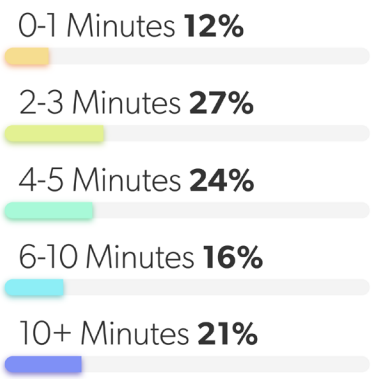
How Long Does it Take You to Remotely Deploy Your Containerized Environment?



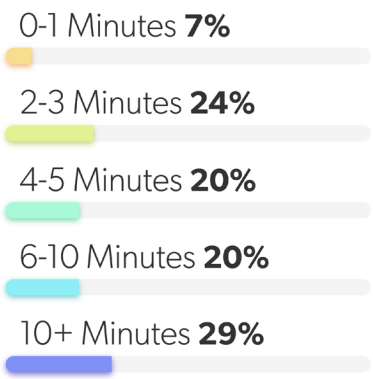


We found larger companies (over 100 employees) experiencing higher redeploy times, with nearly 50% reporting deploy times over five minutes for their containerized environment. Meanwhile, only 37% of smaller companies (under 100 employees) reported times over five minutes.

Companies With Under 100 Employees



Companies With Over 100 Employees



One of the big benefits for adopting microservices was supposed to be the decrease in the amount of time it takes to redeploy the application (specifically, redeploying specific services rather than the entirety of a large, monolithic application). It's interesting to see in this respondent group that this sentiment was inaccurate, and that redeploy times have stayed flat, or even increased in some cases.

Java Technology Trends

At JRebel, we have a vested interest in staying current with the latest trends in Java development technologies. That's why every year we ask Java users to share information about the tools they use to develop Java applications.

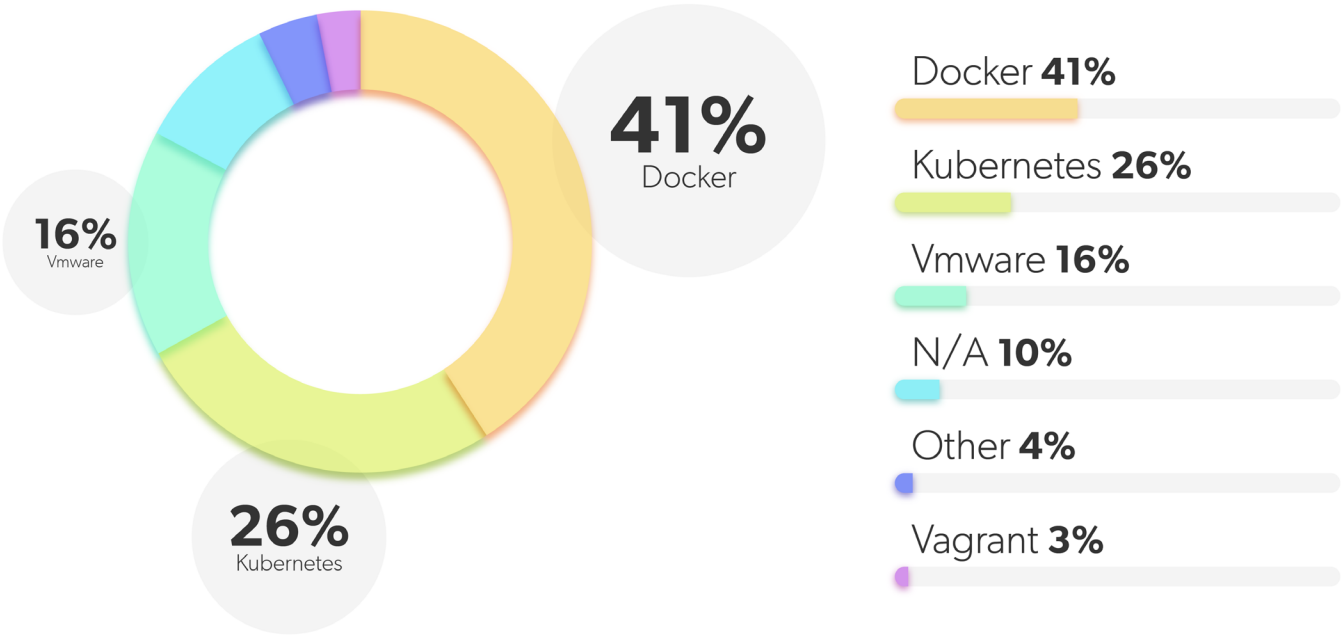
This year, we asked teams to share their tool usage for virtual machines, frameworks, cloud platforms, build tools, and IDEs, as well as how they configure their frameworks.

Java Virtual Machine Platform

For this question, we asked respondents to share the virtual machine platform they use in developing their application. Choices included popular VM platforms like VMWare and Vagrant, as well as popular container and orchestration platforms like Docker and Kubernetes.

Docker was by far the most popular selection, representing 41% of overall respondents. Kubernetes was in second, at 26%, while VMWare rounded out the top three at 16%.

Which Virtual Machine Platform Do You Use?

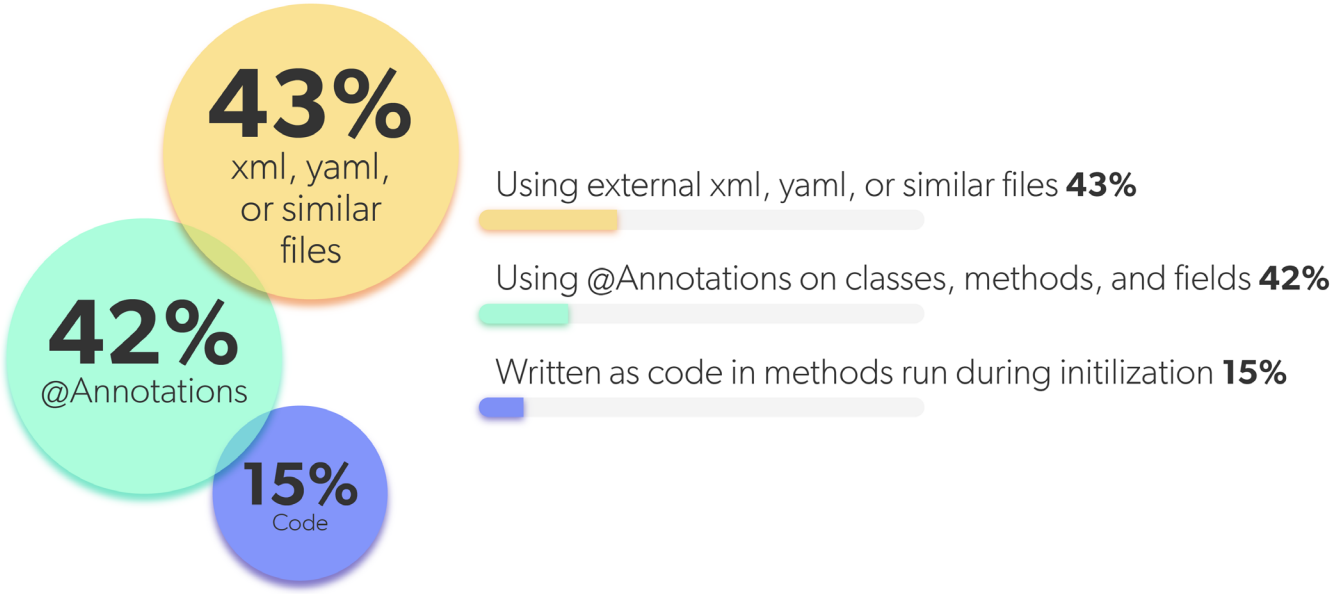


Java Framework Configuration

Next, we asked respondents to share their framework configuration method, and whether they use @ Annotations on classes, methods, and fields; use external xml, yaml, or similar files; or write as code in methods which are ran during initialization.

We found a nearly even split for first between respondents who use external xml, yaml, or similar files (43%) and those who use @Annotations on classes, methods and fields (42%). Configuring frameworks via written code in methods was the least popular response at 15%.

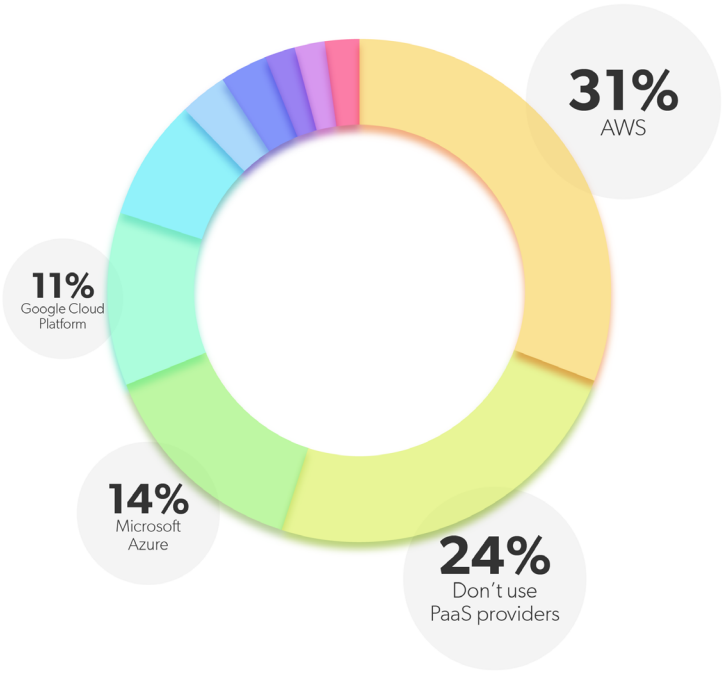
How Do You Configure Most of Your Frameworks?



Java PaaS Provider

In our next question, we asked respondents to share which cloud platform or platform as a service (PaaS) they use with their Java application. The results showed AWS as the far and away favorite at 31%. Those not using a PaaS were in second place at 24%. Azure was the third most popular at 14%, followed by Google Cloud Platform at 11%.

If You Use a Platform, Who is Your PaaS Provider?



- AWS **31%**
- We don't use PaaS providers **24%**
- Microsoft Azure **14%**
- Google Cloud Platform **11%**
- Other **8%**
- Oracle Cloud Platform **3%**
- IBM Cloud **3%**
- SAP Cloud Platform **2%**
- Pivotal Cloud Foundry **2%**
- VMWare Tanzu **2%**

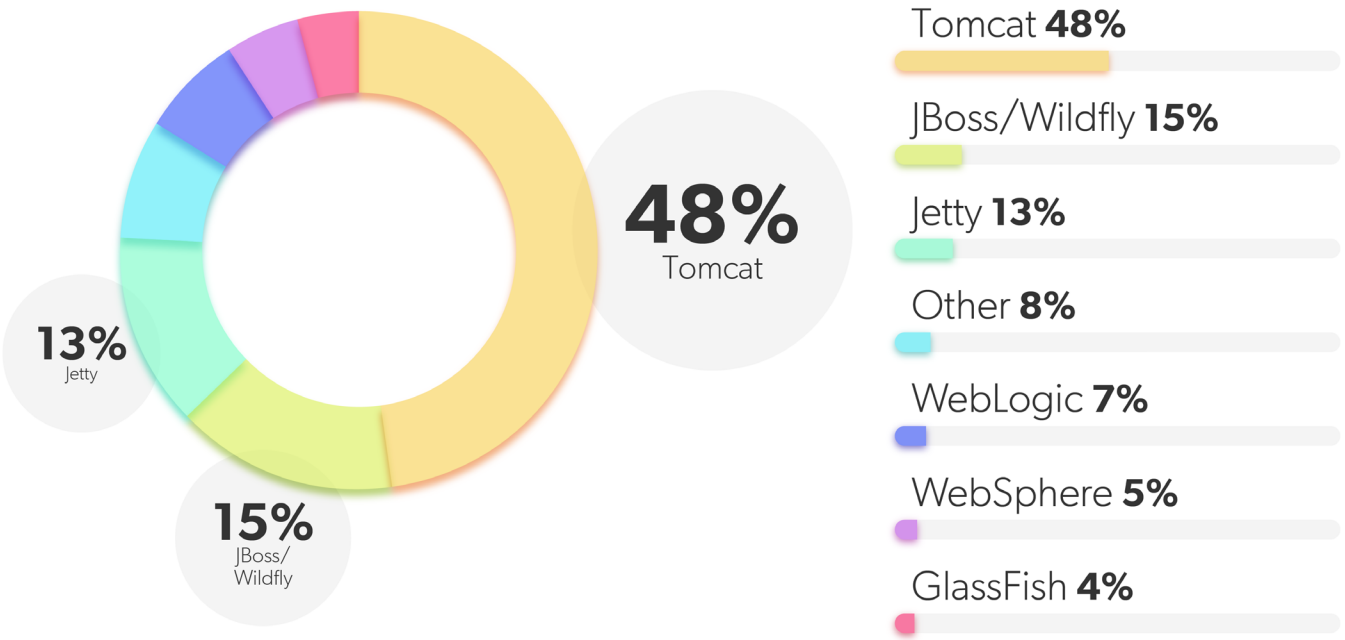
Application Servers

Next, we asked respondents to share details on the application server they use in developing Java applications.

As with previous years, Apache Tomcat was far and away the most popular Java application server at 48% of all responses. Tomcat was followed by JBoss/Wildfly (15%), Jetty (13%), WebLogic (7%), WebSphere (5%), and GlassFish (4%).

Among respondents who selected other, Payara was most common.

What Application Server Do You Use on Your Main Application?

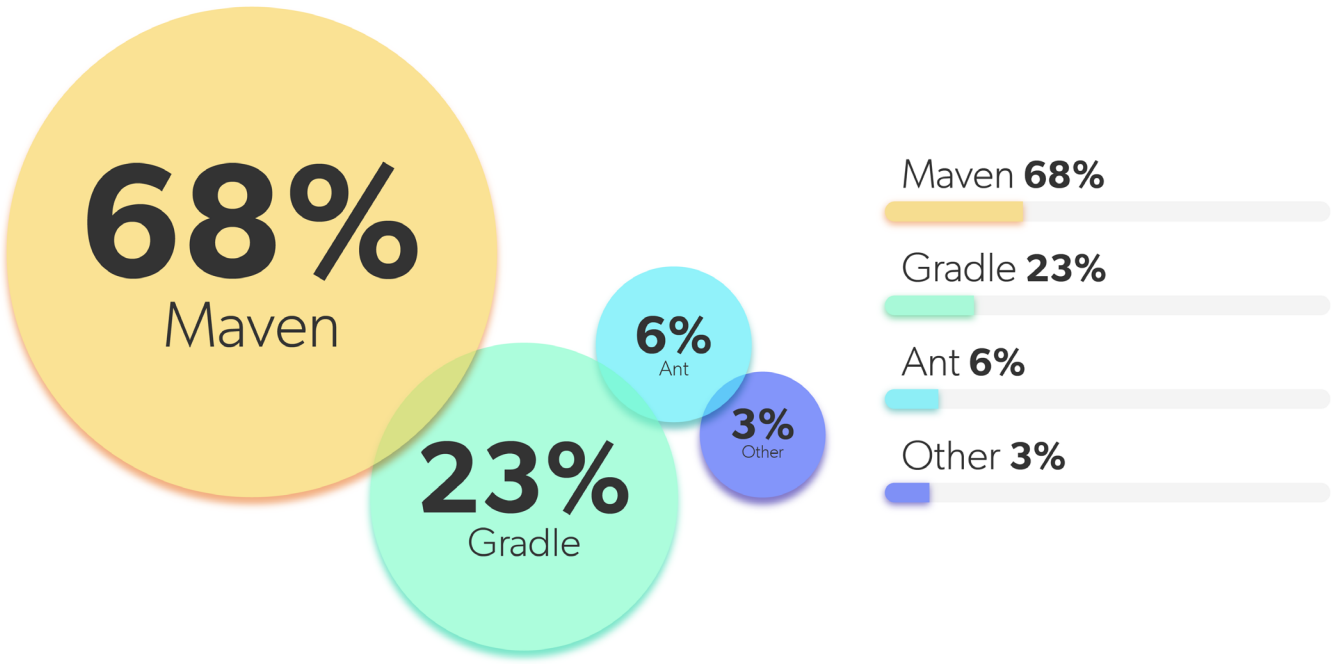


It's not surprising to see Tomcat again topping the list among application servers. There is one slightly misleading stat here, as we feel that the combined Glassfish/Payara number would push it above both WebSphere and WebLogic in terms of overall usage.

Build Tools

In 2022, we again asked respondents to share their build tool of choice. The findings were nearly the same, with Maven at 68% in 2022 and 67% in 2021, Gradle at 23% in 2022 and 20% in 2021, and Ant at 6% in 2022 and 11% in 2021.

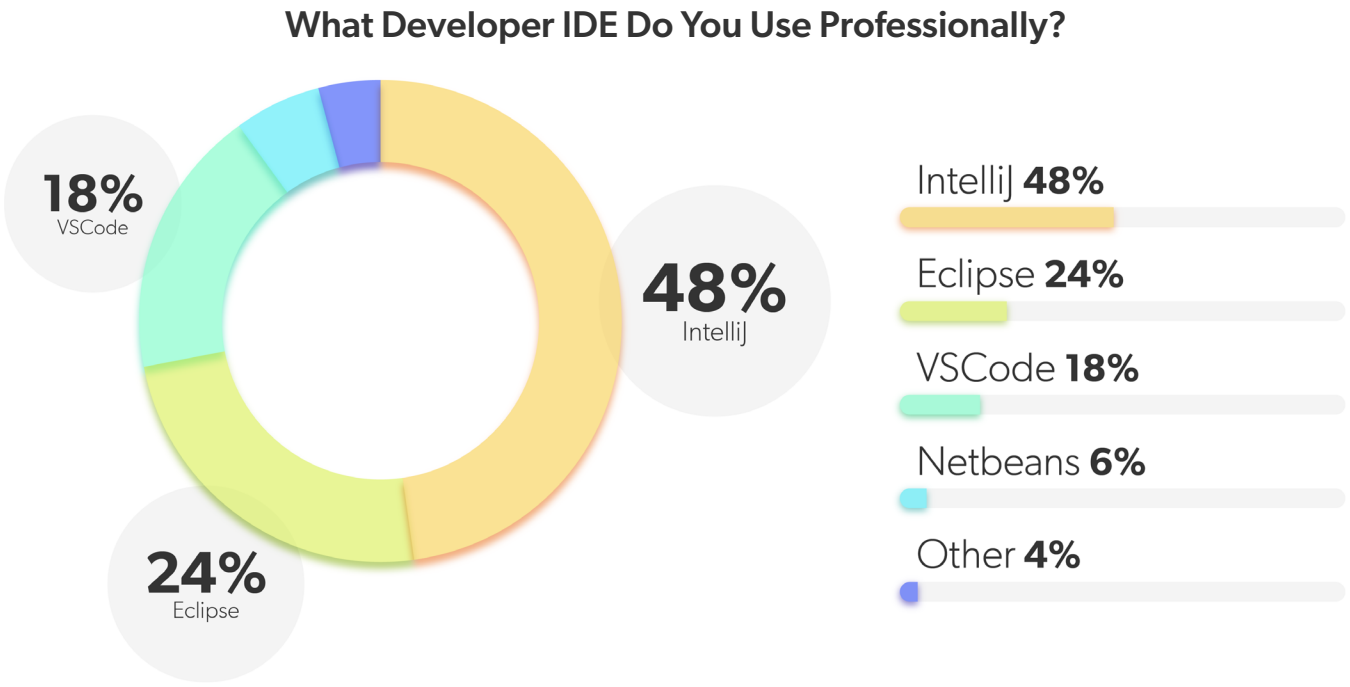
What Build Tool Do You Use in Your Main Application?



Larger companies reported higher usage of Maven when compared to smaller companies, with 71% and 65% reporting use of Maven, respectively.

Java IDEs

Every year we also ask respondents to share their IDE of choice. In 2022, IntelliJ IDEA was again the most popular IDE used in developing Java applications with 48% of respondents choosing it as their IDE of choice. IntelliJ was followed by Eclipse (24%), VSCode (18%), and Netbeans (6%).



VSCode has quickly entrenched itself within a competitive Java IDE landscape. While most people are probably using it in tandem with their primary IDE, it has eaten into the amount of people listing Eclipse as their IDE of choice. (Also, shameless plug, we have a VSCode JRebel plugin coming soon!)

CI/CD Technology Trends

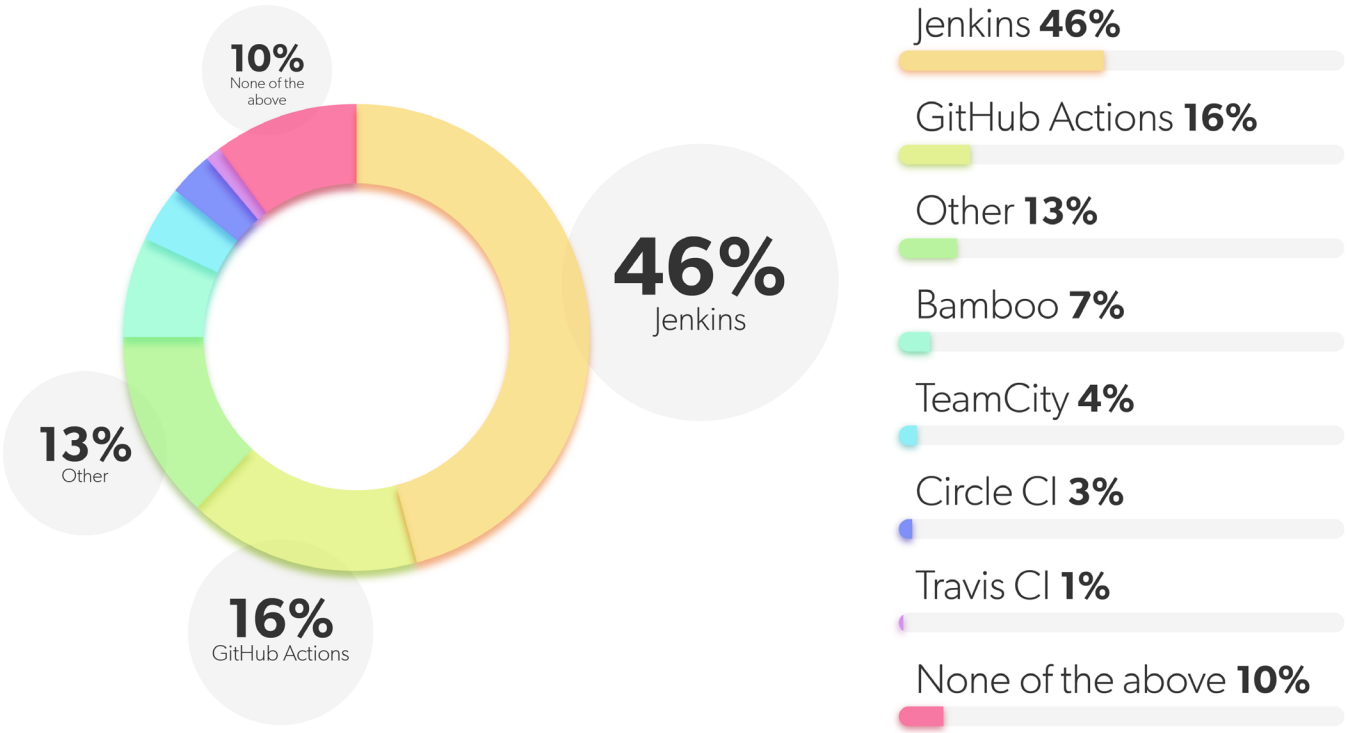
CI/CD and the implementation of DevOps methodologies and tools is a continuing theme across enterprise organizations. In this section of the report, we look at the technologies these organizations are using for CI/CD — with additional questions on build time and frequency.

CI/CD Technology Usage

For our first question of this segment, we asked teams to share the CI/CD technologies they use in their Java applications.

Just like last year, Jenkins was far and away the most popular selection — grabbing 46% of responses overall. GitHub Actions was in second at 16%, followed by Bamboo (7%), TeamCity (4%), Circle CI (3%) and Travis CI (1%).

Which CI/CD Technologies Are You Using?

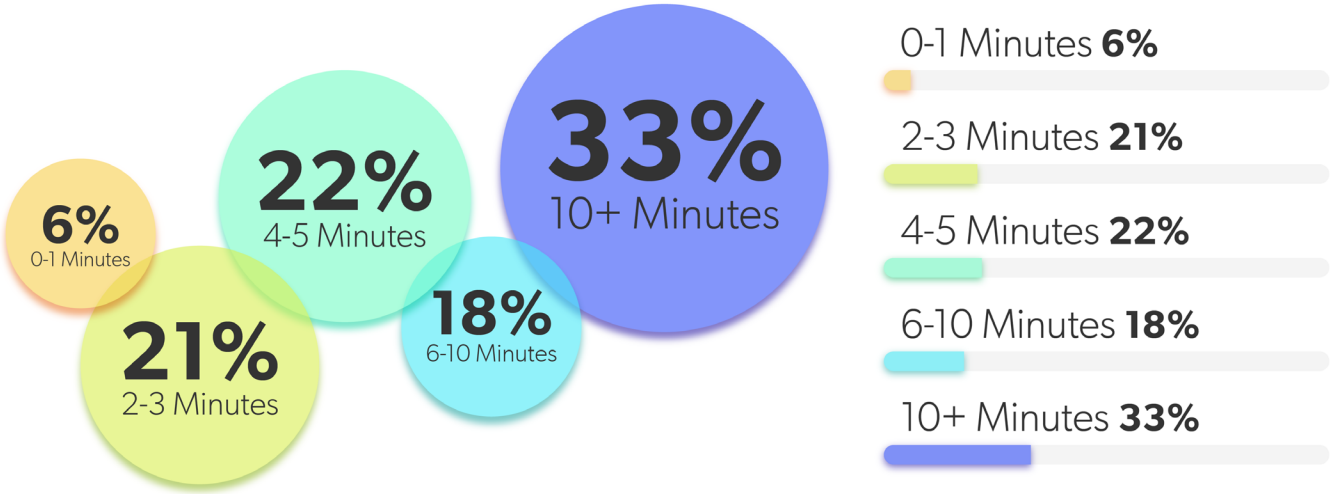


Seeing Jenkins at the top of the list again is no surprise. It would be interesting to see what percentage of them are using Jenkins X. It's also interesting to see differing results between the **2022 State of Open Source Report from OpenLogic** — which listed GitHub Actions as the top technology. It could point to differences in respondents, or differences for the preferred technologies for Java shops.

CI/CD Build Times

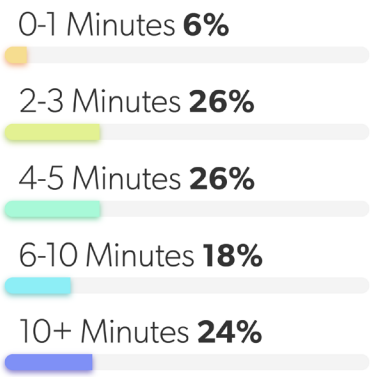
In 2022, we again asked respondents to share the time it takes to complete their CI/CD build. Similar to 2021, respondents reported in with 10+ minutes as the most common response. However, the percentage of respondents for that selection grew from 27% in 2021 to 33% in 2022. All said, teams who reported build times over five minutes represented over half of all respondents at 51%.

How Long Does it Take to Complete Your CI/CD Build?

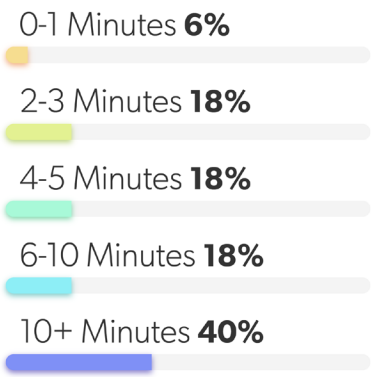


For smaller organizations (under 100 employees), respondents reported faster build times overall, with 58% reporting build times under 5 minutes. When looking at larger organizations (over 100 employees) that rate shrank to 42%.

Companies With Under 100 Employees



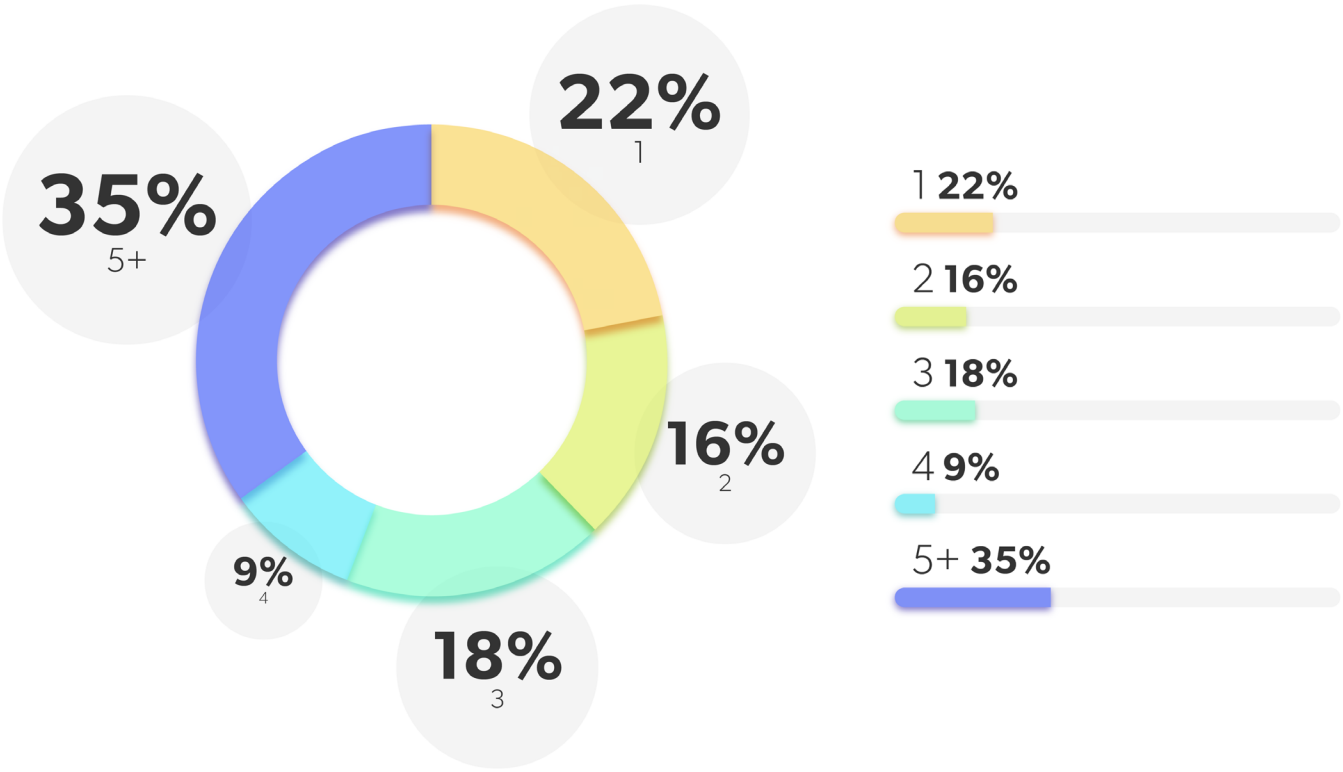
Companies With Over 100 Employees



CI/CD Commit Frequency

Next, we asked respondents to share the number of times they commit code to their CI/CD build per day. Similar to last year, most respondents noted that they committed code at least five times per day.

How Many Times Do You Commit Code to Your CI/CD Build Per Day?



Developer Productivity Trends

One of our favorite parts of our survey each year is asking about redeploy times. For those that aren't familiar with JRebel, we ask about redeploy times because JRebel helps Java developers to completely bypass the redeploy process during Java development.

When you consider the time wasted per redeploy, and the number of times the average Java developer redeploys their application every day, redeploy times can be a costly impediment to efficient Java development.

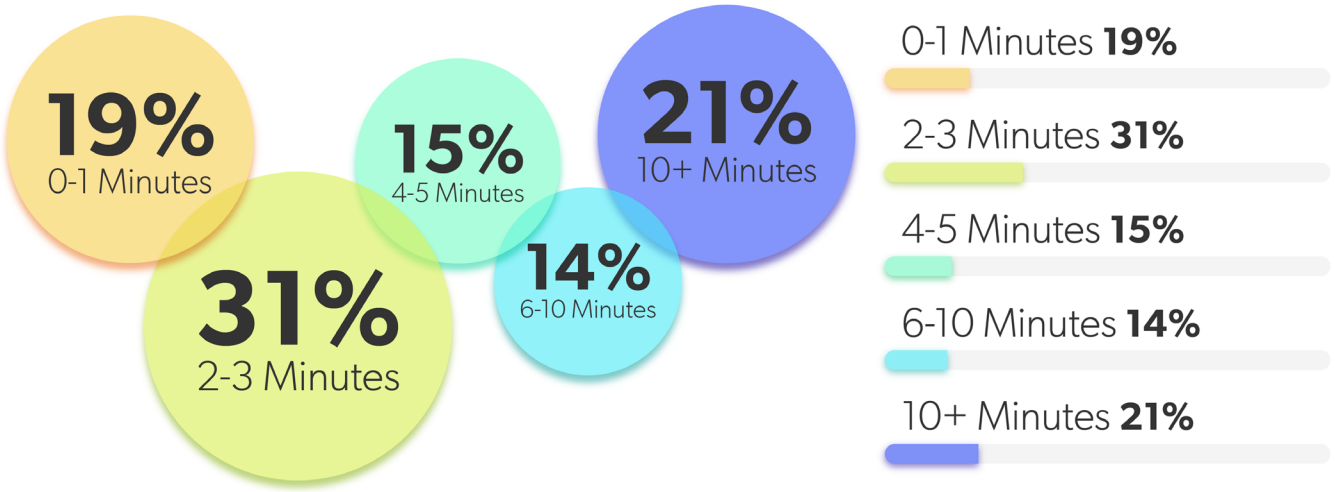
In the next questions, we dive in on those numbers — asking developers to share their experiences with redeploy times, frequency, and what teams would do if they could save the time they waste on redeploys.

Redeploy Times

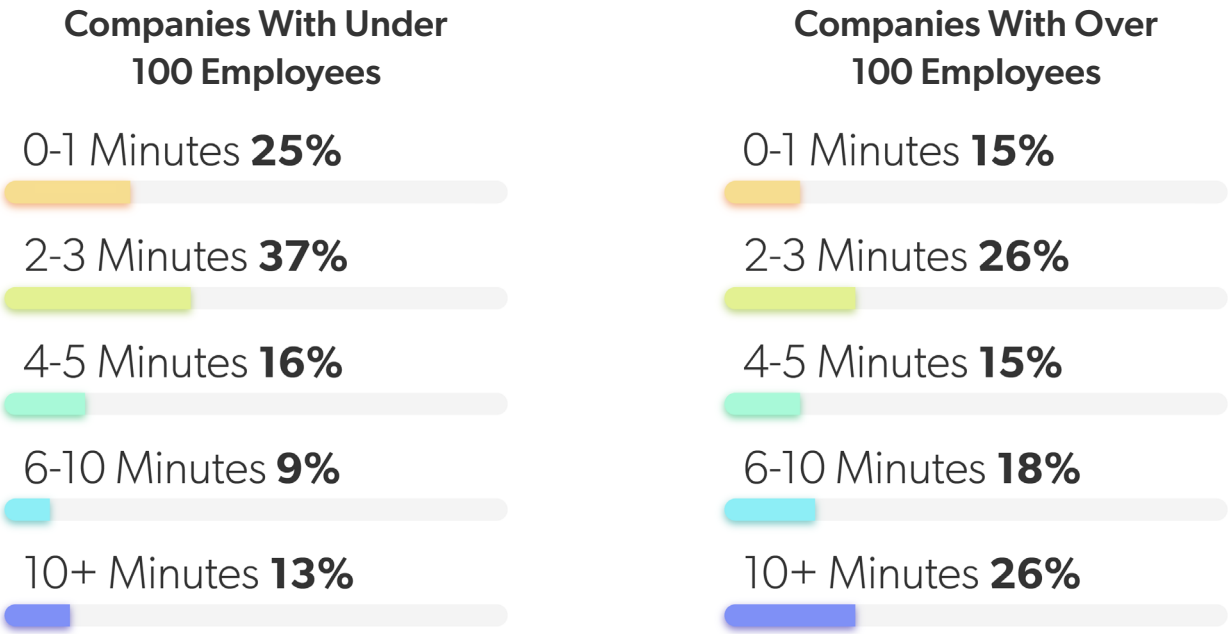
Earlier in the report, we shared the redeploy times for teams using microservice-based applications. For this question, we asked all respondents to share the average redeploy time for their Java application.

Like last year, respondents reported redeploy times between 2-3 minutes as the most common at 31%. 10+ minute redeploys were up slightly from 20% in 2021 to 21% in 2022. All said, teams who experienced over three minutes per redeploy represented 50% of all responses.

After a Code Change in Your Application, How Long Does it Take to Compile, Package, and Redeploy Your Application to a Visibly-Changed State at Runtime?



When comparing redeploy times across organizations of different sizes, we found that 78% of smaller organizations (under 100 employees) reported redeploy times of under 5 minutes, compared to 56% for larger organizations (over 100 employees).



While we see innovations across all areas of the Java landscape every year, redeploy times continue to hurt productivity for Java development teams. With the widespread development talent shortage, skipping these redeploys will be important for organizations who want to get the most out of their existing Java teams (and keep them happy).

What Teams Would Do Without Redeploys

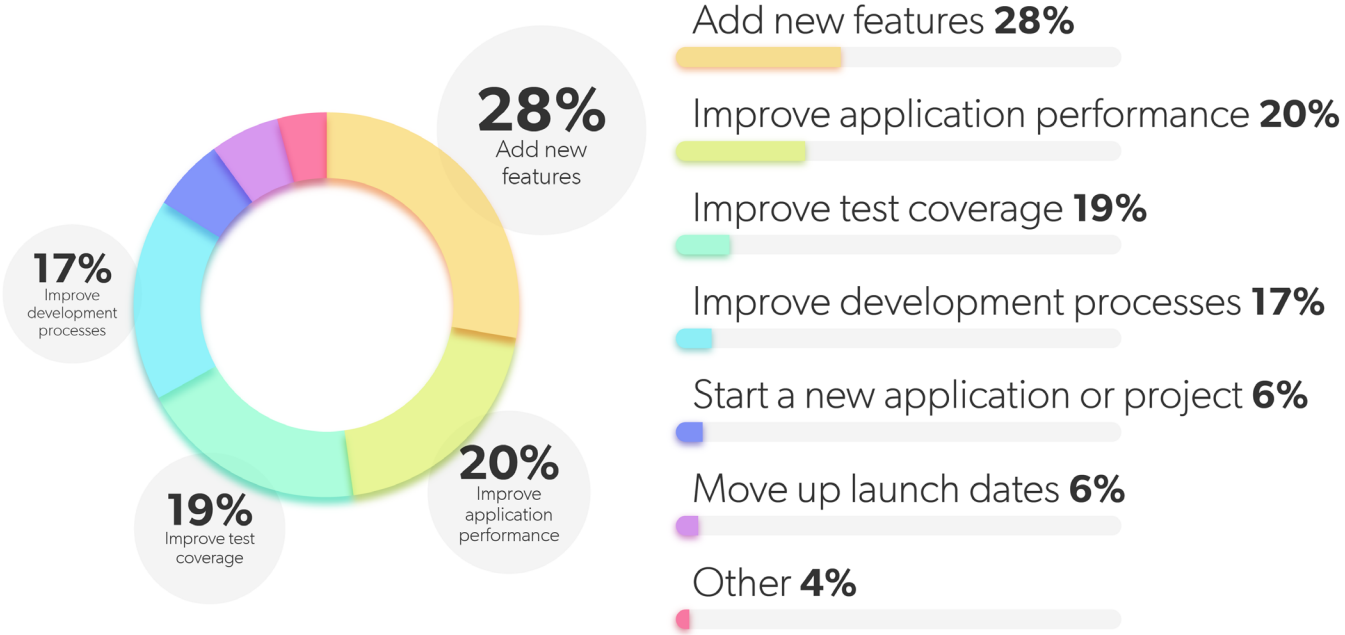
As in previous year's surveys, we ended our survey with a question about what respondents would do if they could save 10% of their development time each workday.

While most people pointed to practical uses, like adding new features (28%), or improving their application performance (20%), the real story of these results was in the "Other" responses.

Highlights included:

- Drink more coffee
- Drink beer
- Party like it's 1999
- Sleep
- Troll

If Your Team Could Save 10% More Time During the Workday, What Would They Do With That Extra Time?



Closing Thoughts

In the 10 years that JRebel has produced the Java Developer Productivity Report, there has been a constant influx of new innovations, trends, and technologies. But with that constant march of innovation comes new and familiar challenges.

In this year's report, we again saw a large number of respondents using or planning on using microservices-based applications. We also saw a significant number of respondents working with CI/CD technologies that help to streamline and automate the build and deploy pipelines.

However, across organizations of all sizes, we again saw that redeploys are a constant threat to developer productivity. And, with qualified developers getting harder to find and retain in an increasingly hot job market, eliminating those redeploys will be instrumental in achieving development goals in 2022 and beyond.

About JRebel

JRebel is a Java developer productivity tool that allows developers to skip redeploys while maintaining application state.

With over 3000 customers around the world, JRebel is trusted by leading brands to improve their Java development productivity, including American Airlines, DellEMC, HBO, Hewlett Packard, Oracle, Volkswagen, and more.

Want to see how JRebel works on your project? Try it free for 14 days with no commitment.

TRY JREBEL FOR FREE

www.jrebel.com/products/jrebel/free-trial