JRebel

2021

Java Developer Productivity Report

Java Tools, Technologies,
and Application Performance

Table of Contents

Introduction 03

Data 04

Respondents 05

Technologies 09

Application Performance 21

Developer Productivity 23

Microservices 28

CI/CD 32

Summary 36

Introduction

Welcome to the 2021 Java Developer Productivity Report by JRebel. Now in our 9th year of compiling this report, the findings are based on a survey of Java developers from around the world and provide valuable insights into how Java development teams are using languages, tools, and technologies in their day-to-day work, as well as how the Java community evolves from year to year.

By having a greater understanding of the trends in the Java development community, and the undercurrents of the Java ecosystem, we are better able to proactively address and solve development issues with our tools, JRebel and XRebel.

Of particular interest are the pain points of Java developers, challenges they experience, and whether current tools are sufficient in tackling these issues.

In order to collect the data, we ran a public survey and invited members of the Java development community to participate. And, while the composition of our results varies from year to year, developers have always been the primary focus and response segment for these surveys.

Similar to last year, a key focus of this year's report is the impact of microservices in the Java ecosystem. Also of interest this year was to measure the change in prioritization each year when it comes to microservices adoption and evaluate how the Java community has responded to the adoption of said microservices.

New to this year's report are a few questions on time savings – what developers do while waiting for redeploys, and what time that would be used for if redeploys were skipped. Also added this year are questions on CI/CD builds including technologies used, time spent, and frequency of code commitments. We hope to be able to build a better understanding of the daily life of a Java developer.

We hope that you enjoy the report and look forward to joining in the discussions (and arguments) that it will generate. Feel free to send us your thoughts on Twitter (@JRebel_Java).

Data

Our report is comprised of data collected during a public survey conducted from August through November 2020. We received 876 total responses.

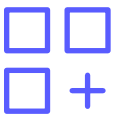
The survey focused primarily on technologies used, performance issues, microservices, continuous integration and continuous delivery (CI/CD), and respondent and company firmographics.

Java Technologies

For technologies, we surveyed the usage of IDEs, JRE/JDK distributions, primary programming language, app architecture, database choices, and virtualization tools.

Java Performance

For performance, we surveyed common performance issues, average redeploy times, and how developers spend their time.



Java Microservices

For microservices, we surveyed application architecture type, adoption status, code visibility satisfaction, troubleshooting satisfaction, and challenges facing microservices developers.

CI/CD

For CI/CD, we surveyed what technologies are used, how long it takes to complete a CI/CD build, and how many times respondents commit code to a CI/CD build per day.

Java Developer and Company Firmographics

We surveyed for job title, company size, and development team size.

Respondents

Not surprisingly, one of the more important things in conducting a Java developer survey is making sure that Java developers participate. But beyond job role, we also asked respondents to provide information on their development team size and company size.

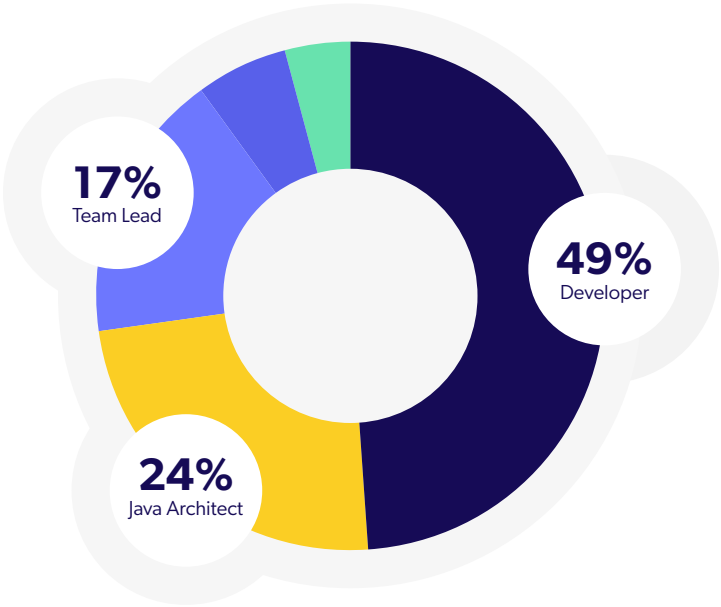
Job Titles

For this survey, we asked respondents to pick which job title best matched their current role.

Not surprisingly, the largest share of our respondents self-identified as Java Developers at 49% of those surveyed. 17% of respondents identified as Team Lead, while Java Architects and Consultants identified at 24% and 4%, respectively. Lastly, 6% of respondents identified as Director, Vice-President, or C-Level.

Overall, the vast majority of our respondents serve in technical roles, which makes us confident that the data collected provides great insight into the development world, which is the focus of this survey. Also, it is great to get feedback from leaders within these companies.

What is your job title?



Developer **49%**

Java Architect **24%**

Team Lead **17%**

Director / VP / C-Level **6%**

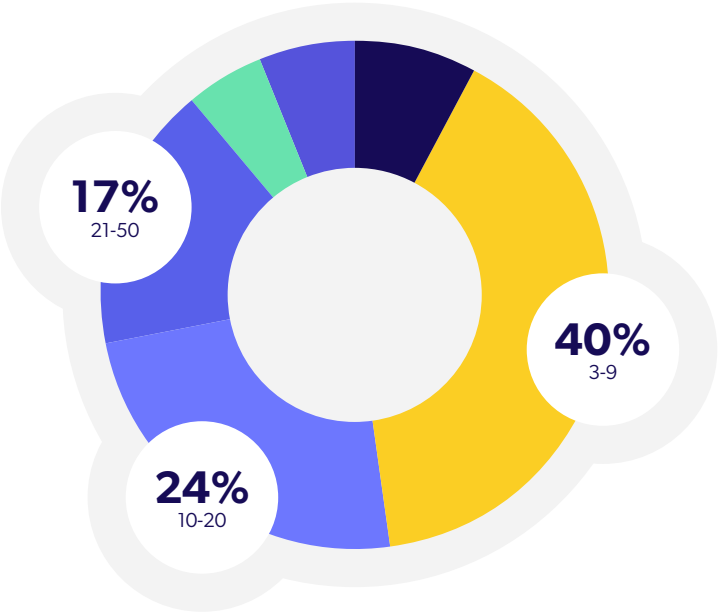
Consultant **4%**

Team Size

Next, we asked about Java development team size. Team size can have a bearing on how processes are performed during development and which tools and technologies are used. Team sizes are typically kept relatively small at most companies due to an increased desire for more agile development. This is also indicative of a transition to more microservice environments where developers work on much smaller segments of code.

The results showed respondent team size largely following previous survey results, with 40% of respondents reporting a team size of three to nine people. Respondents working in teams of 10-20 and 20-50 people measured in at 24% and 17%, respectively. Teams of one to two people made up 8% of respondents, with teams of 50-100 and 100+ people comprising 5% and 6%, respectively.

What is the size of your development team?



1-2 **8%**

3-9 **40%**

10-20 **24%**

21-50 **17%**

51-99 **5%**

100+ **6%**

Company Size

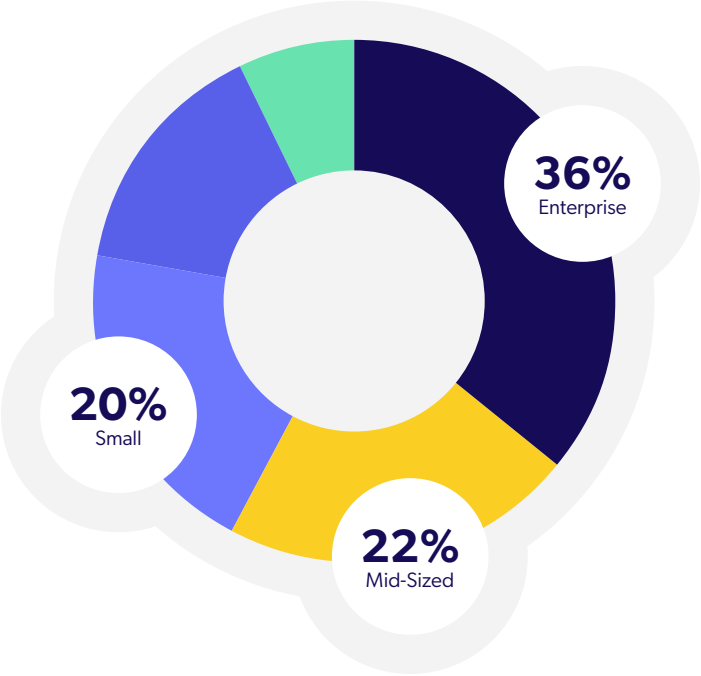
Company size can give valuable information on how to frame questions regarding Java technology choices. We asked respondents to classify the company they work for into 6 categories:

- Small Company
- Midsized Company
- Enterprise Company
- Contractor/Freelance
- Startup
- Unemployed

A little more than one-third (36%) of respondents reported working in an Enterprise company. Respondents reported working in midsized and small companies at 22% and 20%, respectively. The remaining respondents all fell primarily in startup companies at 15%, with contractor/freelance making up the remaining 7%.

Lastly, in a bit of good news, no respondents reported being out of work. This shows promise that even in this tough economic climate due to the COVID-19 pandemic, there is always demand for developers. This also speaks to the continuing health and popularity of the Java language.

What is the size of your company?



Enterprise Company (1,000+) **36%**

Mid-Sized Company (100-1,000) **22%**

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

Startup (1-20) **15%**

Contractor / Freelance **7%**

Technologies

In our questions on Java technologies, we looked at everything from Java language usage, to application architectures, build tools, virtualization, deployment models, and more.

Language

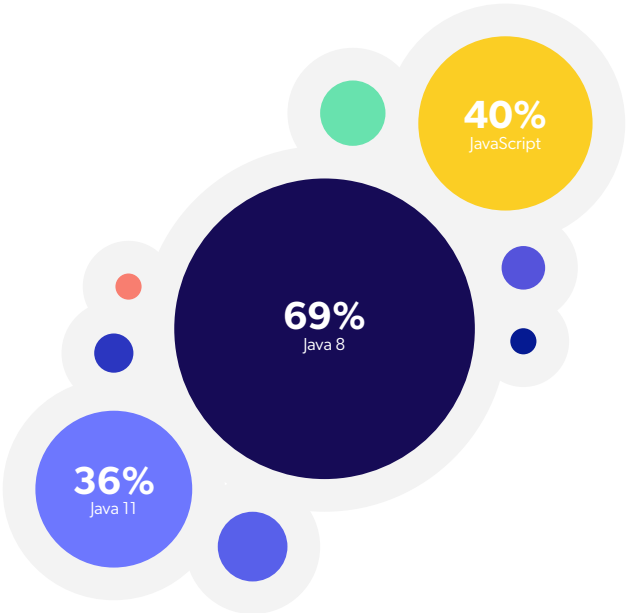
At 69%, the majority of respondents reported using Java 8 as the programming language of choice in their main application, which was expected. This is an increase from 58% of respondents using Java 8 last year. JavaScript and Java 11 followed with 40% and 36%, respectively.

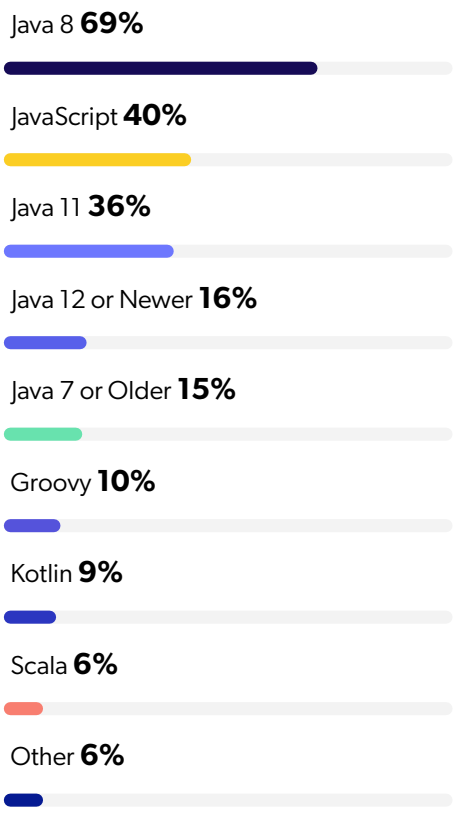
Note: Respondents were able to select multiple answers if more than one programming language was used in their application.

16% of survey respondents reported using Java 12 or newer, with another 15% reporting using Java 7 or older. Groovy (10%), Kotlin (9%), Scala (6%), and other languages (6%) round out the remainder of the responses.

For us, it wasn't a big surprise to see Java 8 as the dominant programming language. In our experience and observations, this is by far the most commonly-used Java version. It is also the Java version released before the transition to more regular releases (every six months) that we currently are experiencing. We believe that there will be a gradual transition to Java 11, or potentially Java 17 when it is released, because of the long-term support provided for these Java releases.

What Java programming language are you using in your main application?





Application Architecture

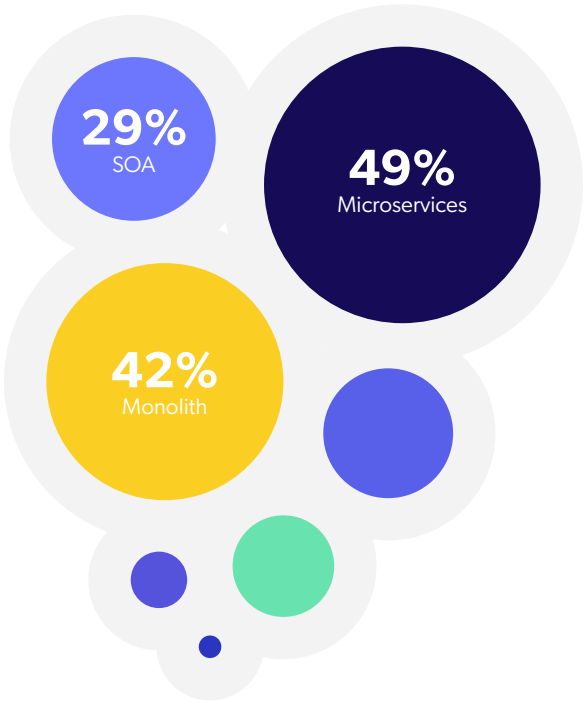
Application architecture is the first step towards creating applications. Companies regularly struggle to identify the best way in which their application should be built. And, with around half (49%) of the respondents now working on microservices-based applications, you can see that effect throughout our survey results.

It is also important to note that microservices use has remained steady from last year's data, where 51% responded using microservices. This may indicate that microservices adoption has become less of a priority within the Java community, or perhaps such a transition was put on hold this year with the current economic climate. We will explore this point in further questions later in this report.

Note: Respondents were allowed to select more than one answer.

Behind microservices, monolith is the next most-used architecture at 42%. SOA (29%), mobile (23%), and desktop (18%) follow behind. Serverless (10%) and other architectures (4%) make up the rest of the reported usage.

What is the architecture of the main application you develop?



Microservices **49%**

Monolith **42%**

SOA **29%**

Mobile **23%**

Desktop App **18%**

Serverless **10%**

Other **4%**

Application Server

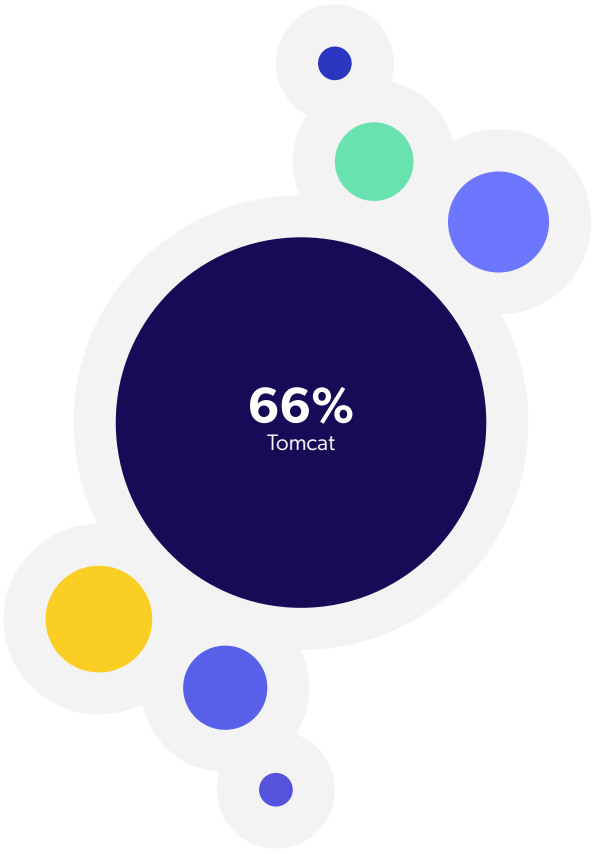
There does not seem to be a whole lot of variety in application servers as, just like last year, Tomcat is dominating the usage among our respondents, increasing to 66% of respondents this year as compared to 61% in last year's survey.

The next highest application server was JBoss/WildFly at 19%, with WebLogic (18%), Jetty (15%), and WebSphere (14%) taking up the next largest shares. Lastly, Glassfish and other servers were each represented by 6% of respondents.

It is clear that Tomcat is doing something right to garner such a large share of the market, and even increase its reported use in a year-over-year comparison. This is likely due to Tomcat being highly compatible with other platforms like Hybris, Spring Boot, Docker, AWS, and others.

Note: Respondents were allowed to select more than one answer if more than one application server was used.

What application server do you use on your main application?



Tomcat **66%**

JBoss/WildFly **19%**

WebLogic **18%**

Jetty **15%**

WebSphere **14%**

GlassFish **6%**

Other **6%**

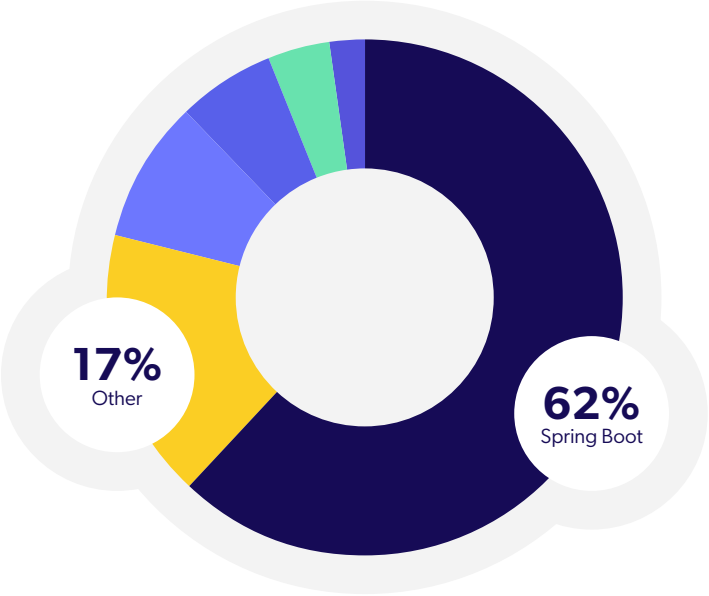
Framework Technologies

This question asked participants to select the application frameworks and technologies being used on their main project.

At 62%, most respondents were working with Spring Boot. This is a decrease from last year's 83% of respondents, though it is clear that Spring Boot is still the dominant choice in this category, largely due to adoption of microservices within the Java community. We aren't surprised by this decrease as we believe this year's results are more indicative of Java community.

Respondents using DropWizard increased to 8% (up from 1% last year), while Quarkus grew to 6% (up from less than 1%), and Micronaut and Vert.x grew to 4% and 2%, respectively (both up from roughly 1% last year).

What application framework are you using on your main project?



Spring Boot **62%**

Other **17%**

DropWizard **9%**

Quarkus **6%**

Micronaut **4%**

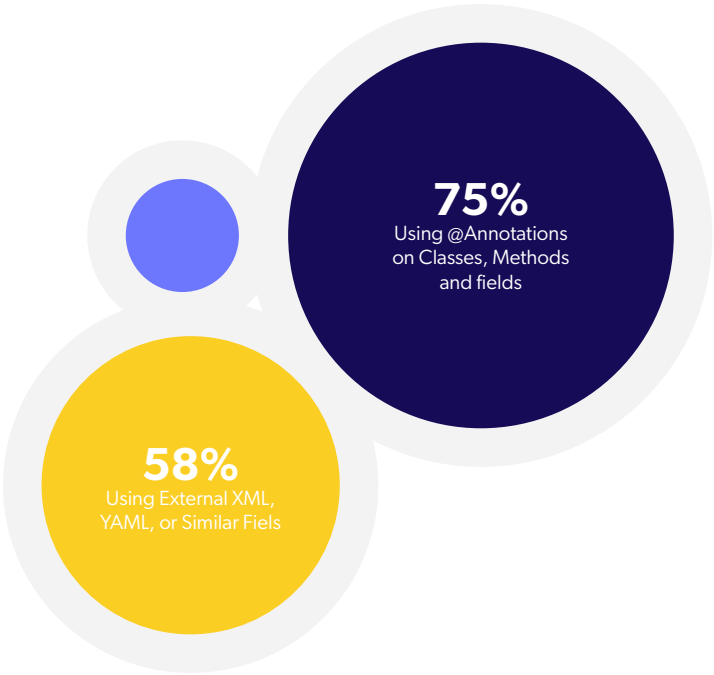
Vert.x **2%**

Framework Configuration

For framework configuration, we found that most respondents were using @Annotations on classes, methods, and fields, or using external xml, yaml, or similar files.

The answers remain quite similar to prior year's data. 58% of users reported using external xml, yaml, or similar files. 75% reported using @Annotations on classes. 22% were configuring with code added to methods that run during initialization.

How do you configure most of your frameworks?



Using @Annotations on Classes, Methods and fields **75%**

Using External XML, YAML, or Similar Files **58%**

Written as Code in Methods Run During Initialization **22%**

IDE

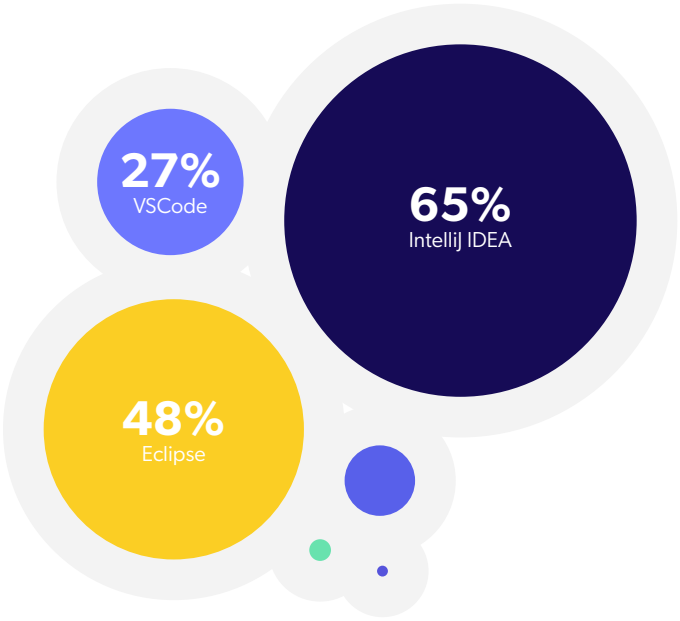
In this question, we asked developers to list the integrated development environments they use professionally. Likely, there is no technology that is more important to developers than IDEs.

As recently as a few years ago, developers would have questioned the value of spending money on an IDE. However, last year's data indicated a change of mindset which continues this year.

With 65% of respondents using IntelliJ IDEA, 48% using Eclipse, and 27% using VSCode, it is apparent that Java users feel the need for a feature rich IDE in IntelliJ.

The next most used IDE, NetBeans, comes in at 13% with browser-based and other IDEs coming in at 4% and 2%, respectively.

What developer IDE do you use professionally?



IntelliJ IDEA **65%**

Eclipse **48%**

VSCode **27%**

NetBeans **13%**

Browser-Based IDE **4%**

Other **2%**

JRE/JDK Distribution

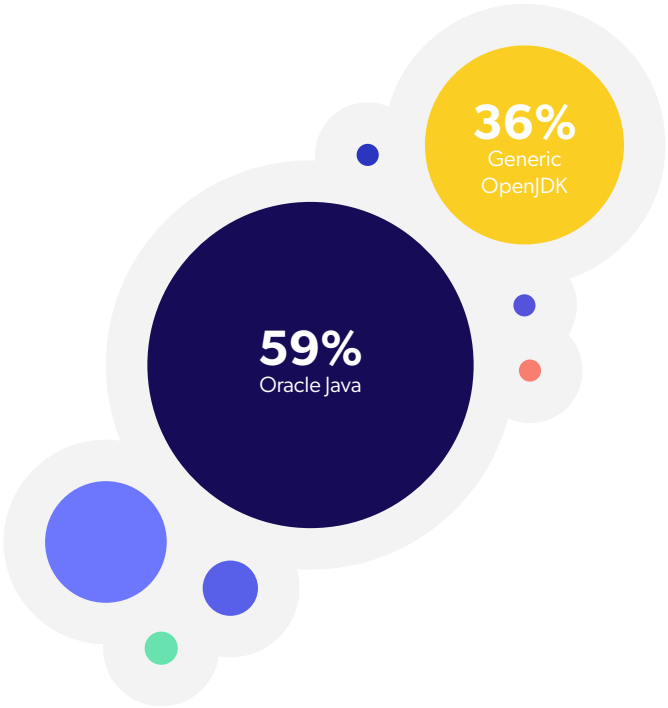
In this survey question, we asked respondents to select the JRE/JDK distribution they use. Last year, nearly 50% of respondents were using Oracle Java, and that number grew this year to 59%.

We expected to see more transition from Oracle to other Java distributions given the high licensing costs from Oracle, but this is not the case. We believe that this has a lot to do with the amount of enterprise developers that filled out the survey. Enterprise companies are slower to transition software arcitecture than smaller, more agile companies.

Aside from Oracle Java, 22% of respondents reported using AdoptOpenJDK, another 10% reported using Amazon Corretto. Distributions such as Azul Zulu, GraalVM, OpenLogic, and others all were reported between 4% and 6% usage.

Note: More than one answer could be selected.

What JRE/JDK distribution do you use?



Oracle Java **59%**

Generic OpenJDK **36%**

Adopt OpenJDK **22%**

Amazon Corretto **10%**

Azul Zulu **6%**

GraalVM **4%**

OpenLogic JDK **4%**

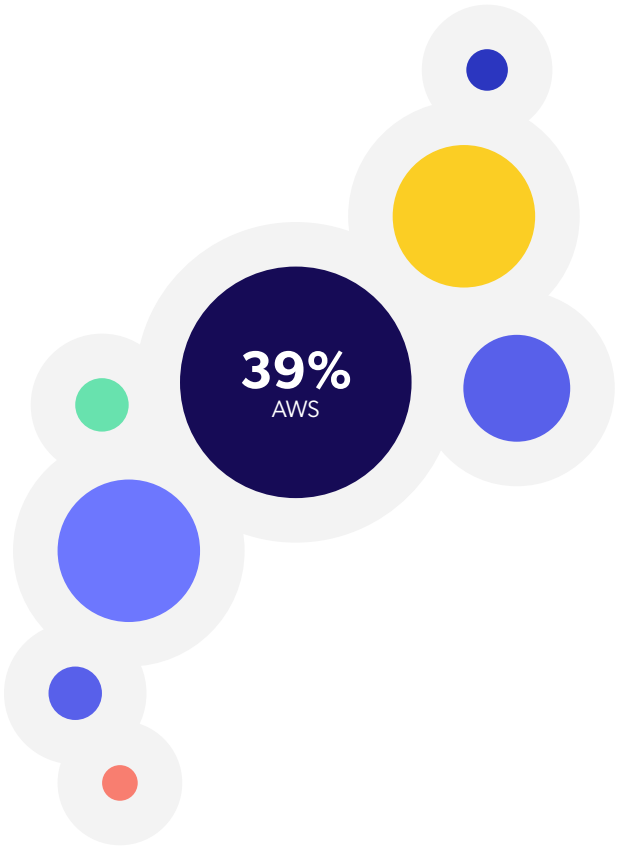
Other **4%**

PaaS Provider

In this question, we asked respondents to select which PaaS provider they used. 24% of respondents reported not using a PaaS provider at all. Of those that do, AWS continues to be most popular choice with 39% of users. Microsoft Azure and Google Cloud follow with 24% and 18% usage, respectively.

Lesser-used providers include IBM Cloud with 9%, SAP Cloud at 9%, Pivotal Cloud Foundry at 7%, and others making up 6%.

If you use a platform, who is your PaaS provider?



AWS **39%**

Microsoft Azure **24%**

We Don't Use PaaS Providers **24%**

Google Cloud Platform **18%**

IIBM Cloud **9%**

SAP Cloud Platform **9%**

Pivotal Cloud Foundry **7%**

Other **6%**

Database

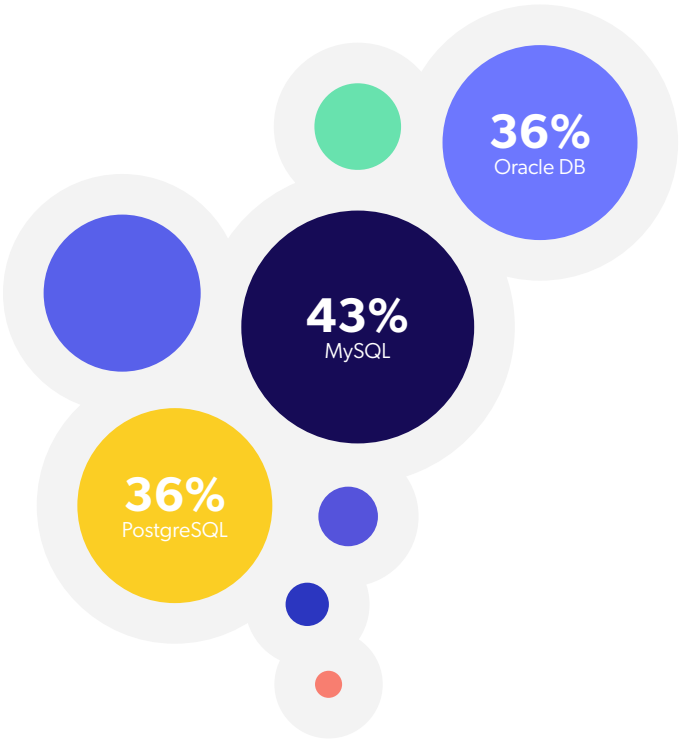
In this question, we asked respondents to select the database(s) they're currently using. The most popular was MySQL at 43%, with Oracle DB and PostgreSQL both coming in at a close second 36% each.

Next up was MongoDB with 29% of respondents reporting usage. Other SQLs (a catch-all for databases not explicitly listed in the answer choices) was next most with 16% of respondents.

The least-used databases included NoSQL, Cassandra, and Neo4j with 11%, 8%, and 5% usage, respectively.

Note: More than one answer could be selected.

What databases are you using?



MySQL **43%**

Oracle DB **36%**

PostgreSQL **36%**

MongoDB **29%**

Other SQL **16%**

Other NoSQL **11%**

Cassandra **8%**

Neo4j **5%**

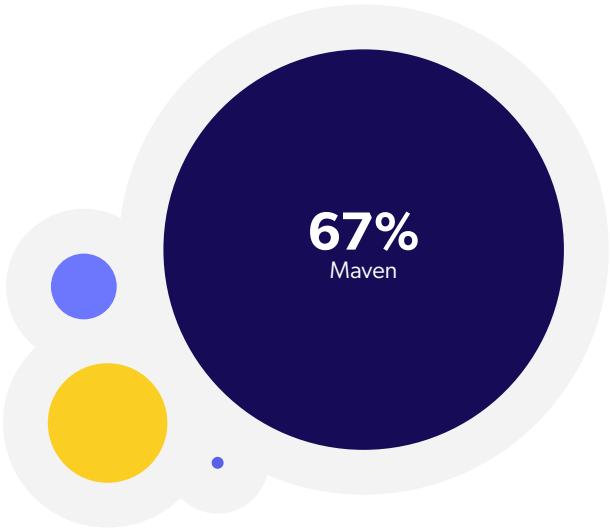
Build Tools

For build tools, we asked developers to pick the tool they use on their main application. Expectedly, just like last year, Maven and Gradle were the most used tools.

However, in a change from last year, Maven has emerged as the tool of choice. Whereas last year the usage between these two was relatively split, at 47% for Gradle and 44% for Maven, this year's data shows a shift to 67% usage of Maven. This variation was expected, as there was a decrease in Android application development this year, and Android greatly lends itself to Gradle builds.

Ant usage trailed both at 11% of respondents, while other tools make up the remaining 2%.

What build tool do you use in your main application?



Maven **67%**

Gradle **20%**

Ant **11%**

Other **2%**

Virtualization Tools

In this question, we asked developers which virtualization tools they use. Far and away the most commonly used tool was Docker at 57%, though it has come down from 74% usage last year. Considering the percentage of respondents using microservices, this wasn't a big surprise that Docker is still used by more than half.

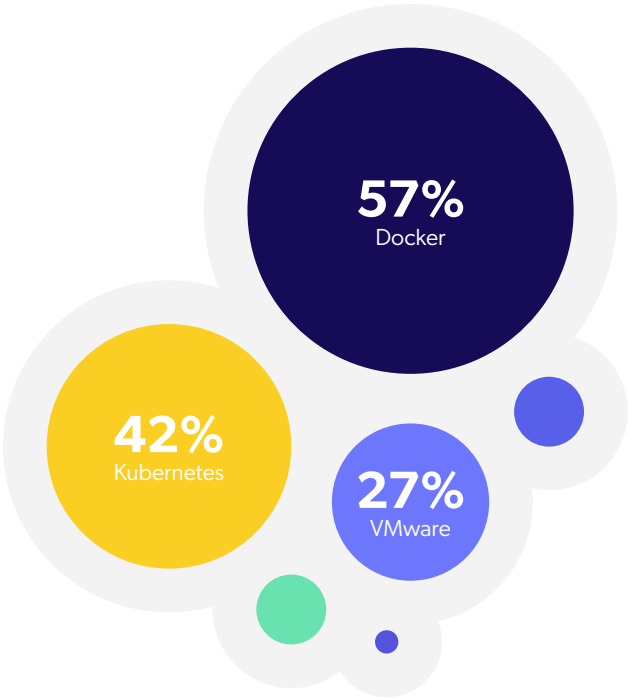
Also not surprising was seeing Kubernetes as the second most popular virtualization tool at 42%, which is an increase from 35% last year.

VMware also made a jump from last year's 18% usage, up to 27% this year. Vagrant (12%) and other tools (4%) make up the last two.

Lastly, 12% of respondents reported not using a virtualization tool.

Note: More than one answer could be selected.

Which virtual machine platform do you use?



Docker **57%**

Kubernetes **42%**

VMware **27%**

N/A **12%**

Vagrant **12%**

Other **4%**

Application Performance

As XRebel is a tool that solves performance issues for developers, one of our favorite questions to ask developers is where they're experiencing performance issues.

Application Performance

For the purposes of this survey, we broke these common performance issues into six categories:

- Long application response time
- Memory leaks
- High CPU usage
- Too many open connections
- Excessive IO queries
- Other

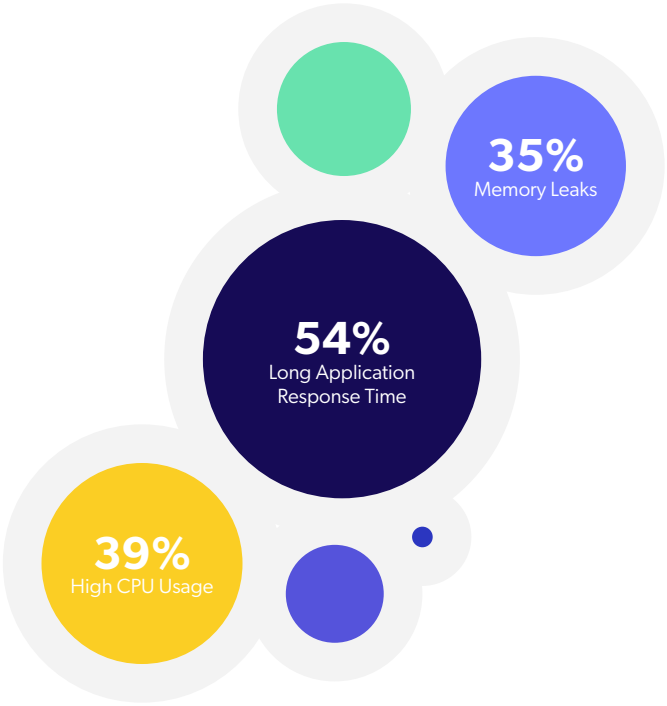
In cases where the respondent marked other, they were asked to specify the issue.

Responses remain largely unchanged from the prior year's report. Like last year, the most common performance issue was long application response time at 54% (on par with 55% reported last year). This again marks a continuing trend for developers, and again aligns with microservices adoption.

The next highest was high CPU usage at 39% of respondents. Memory leaks marked the third most common issue at 35% of respondents, with too many open connections and IO queries coming in at 26% and 19%, respectively.

4% of respondents chose other, with no statistically notable answers from those who wrote in.

What is the most common performance issue you run into in your application?



Long Application Response Time **54%**

High CPU Usage **39%**

Memory Leaks **35%**

Too Many Open Connections **26%**

Excesive IO Queries **19%**

Other **4%**

Developer Productivity

With a survey coming from JRebel by Perforce, it should come as no surprise that we are curious about developers and their productivity, as our flagship product helps developers save time by skipping rebuilds and redeploys.

Average Redeploy Times

No matter how long they take, redeploys take too long, especially when compounded by the frequency of redeploys and the number of developers on your team.

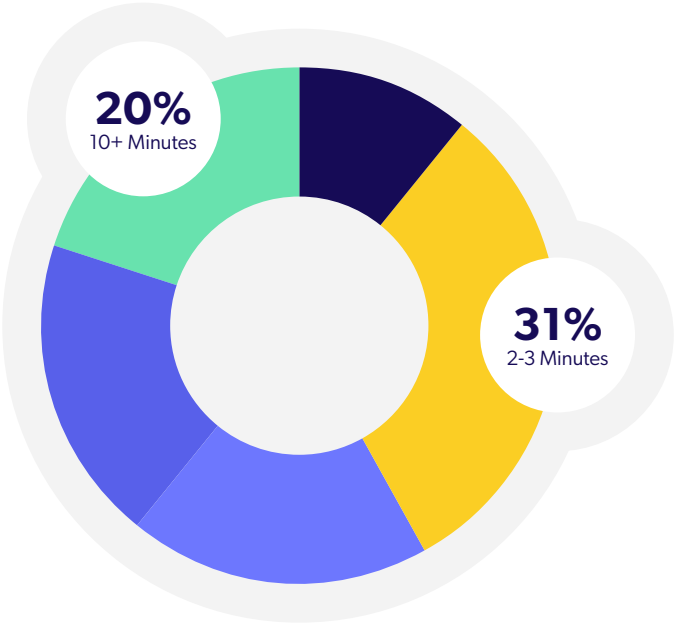
In this survey question, we asked developers to place their redeployment time onto a scale of zero minutes to ten minutes plus.

At the top, just like last year, the most common redeploy time was between 2-3 minutes at 31% of respondents. Whereas last year the second-highest choice was 0-1 minutes (at 20%), that has shifted this year to a near three-way tie for all choices of four minutes or greater, each of which had between 19% and 20% of respondents. Rounding out the list is 0-1 minutes at 11%.

These results are particularly surprising. With nearly half of respondents in a prior question reporting that microservices is the architecture of the main application they develop, it is interesting to see that 59% are experiencing reload times over four minutes, and 20% are reporting reload times greater than ten minutes!

There are two potential reasons behind this. One is that microservices grow in size, and the impact on the average developer increases, the longer they develop and create the application. Second, there are often microservices running on remote virtualization machine like Docker, which we discussed earlier.

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?



0-1 Minutes **11%**

2-3 Minutes **31%**

4-5 Minutes **19%**

6-10 Minutes **19%**

10+ Minutes **20%**

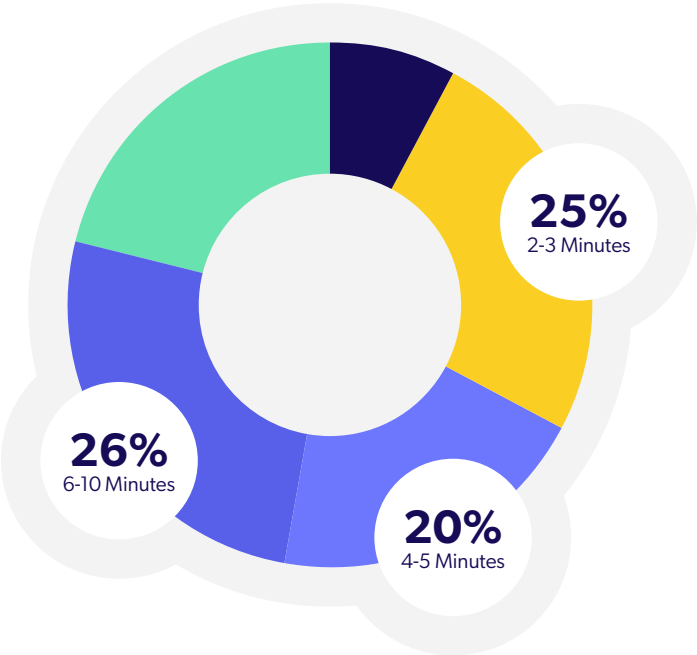
Containerized Environment Deployment

In this question, we asked developers how long it takes to remotely deploy their containerized environment on a scale of zero minutes to ten minutes plus.

There was a relatively even distribution of answers across the five choices, with 6-10 minutes reported at 26%, followed by 2-3 minutes at 25%, and 4-5 minutes and 10+ minutes both reported at 21%.

Lastly, only 8% of the respondents are reporting a deployment time of one minute or less. This shows a critical part of the process developers experience when they commit code to their containerized environment.

How long does it take you to remotely deploy your containerized environment?



0-1 Minute **8%**

2-3 Minutes **25%**

4-5 Minutes **20%**

6-10 Minutes **26%**

10+ Minutes **21%**

How Redeploy Time Is Spent

With the surprisingly long redeploy times provided in the previous questions, we wanted to know what developers are doing while waiting.

The most popular answer, with more than one-third of respondents, was grabbing a coffee with 35%.

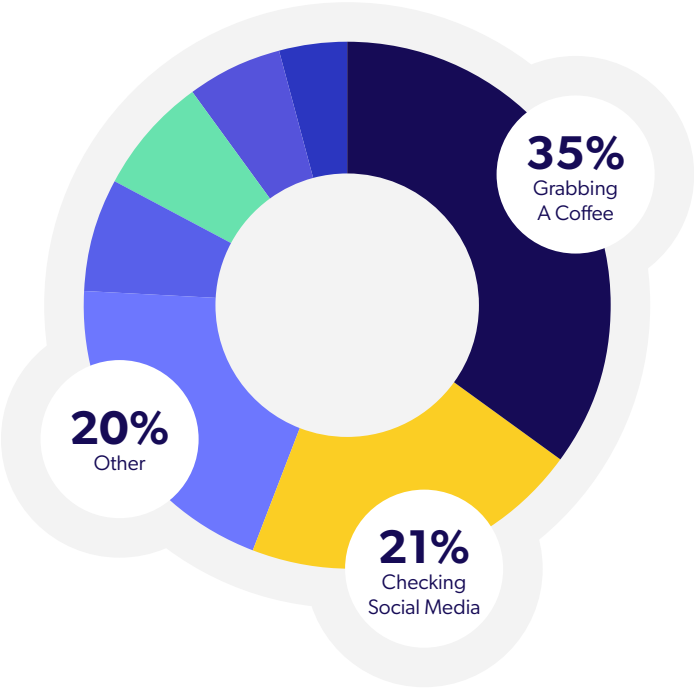
After that, there is a steep drop-off to the next most common answer of checking social media with 21%.

20% of respondents wrote in their own answers, with variations of working on other tasks or checking email making up the majority of their open responses.

Checking in with family followed with 7% of respondents, matched by taking a nap with 7% as well. Those either must be long redeploys or short naps!

Lastly, 6% answered that they walked their dog, followed by 4% who played video games.

While waiting on a redeploy, what are you typically doing?



Grabbing A Coffee **35%**

Checking Your Twitter/Facebook/Instagram **21%**

Other **20%**

Checking In With Family **7%**

Taking A Nap **7%**

Walking Dog **6%**

Playing Video Games **4%**

What Would Be Done With More Time

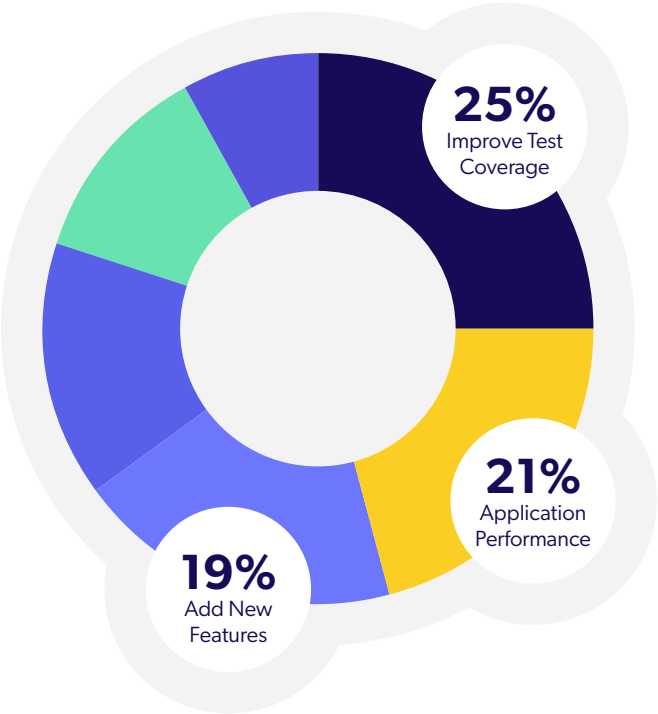
Now that we know what developers are doing while they are waiting for redeploys (some are more productive than others), we were curious to know how their extra time would be spent if their team could save 10% more time during the work day.

The most common answer was improving test coverage with 25% of respondents. This is followed by improving application performance (perhaps addressing the issues that were brought up in a previous question) at 21%.

19% indicated that they would add new features, while another 15% would spend that time improving the development process.

Some would accelerate their release cadence, with 12% moving up launch dates. Lastly, 8% would start a new application of project.

If you team could save 10% more time during the work day, what would they do with that extra time?



Improve Test Coverage **25%**

Improve Application Performance **21%**

Add New Features **19%**

Improve Development Processes **15%**

Move Up Launch Dates **12%**

Start a New Application or Project **8%**

Microservices

You can't enter any conversation in the Java community these days without hearing a mention of microservices. As evidenced in a prior question, this application architecture has become a popular choice among Java application developers.

However, as we'll explore in the next few pages, while microservices provide advantages, they do not come without challenges. In the pages that follow, we'll take a look at microservices adoption in Java development.

Microservices Adoption

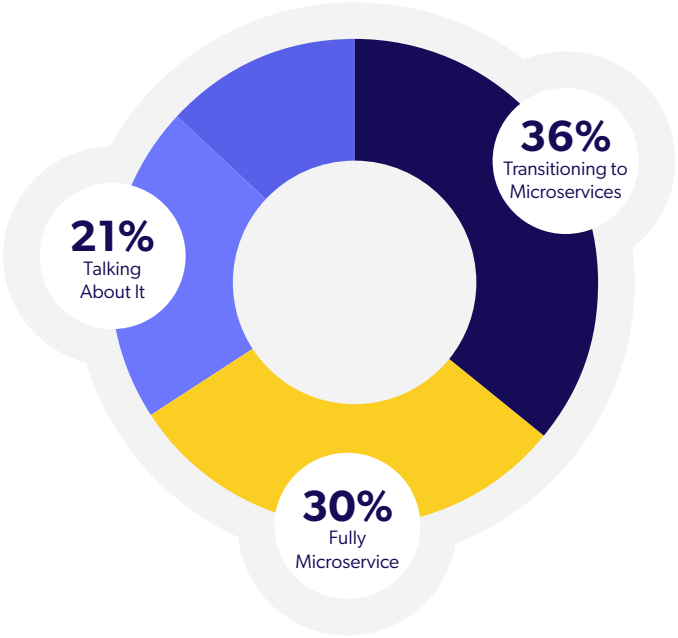
In our survey, we found that nearly 66% of our survey respondents were either working in, or actively transitioning to, microservices. This is just a slight increase over last year, where 63% of respondents fit into these two categories.

An additional 21% were actively talking about adopting microservices, the same as last year, making it 87% of our survey respondents (compared to 85% last year) are either in microservices or considering adoption. Only 13% do not have microservices in their plans.

The consistency of the data in a year-over-year comparison leads us to start questioning how microservices are affecting the architecture of the applications being developed, and whether we are seeing more prioritization put towards different aspects of applications from the standards created for microservices.

Lastly, like last year, we are most surprised that 30% of respondents are using fully microservice-based applications.

What is your status for micro services adoption?



Currently Transitioning to Microservices **36%**

Fully Microservice-Based Application **30%**

We're Talking About It **21%**

Not In Plans **13%**

Microservices Usage

Next, we wanted to know how many microservices developers had in their primary application.

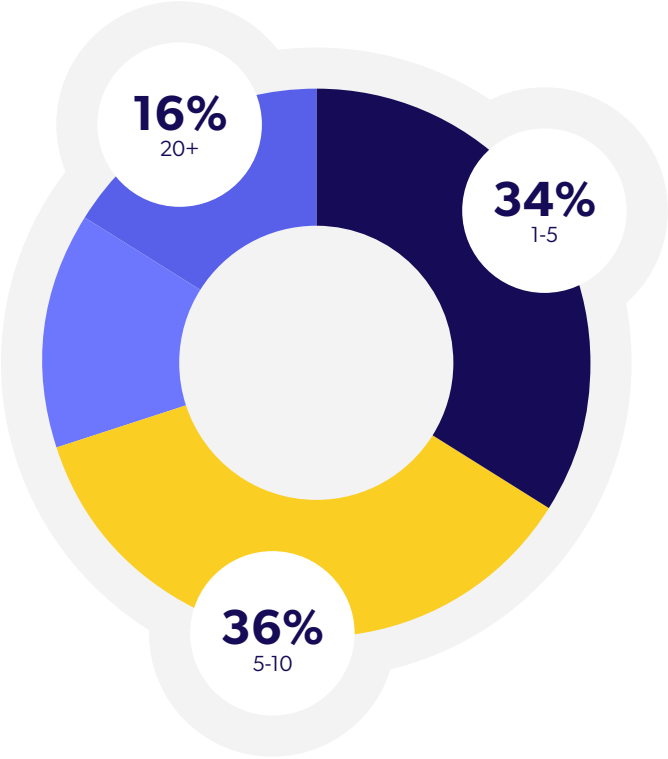
Answers were provided on a scale of 1 microservice to more than 20 microservices.

The most common answers, with nearly the same amount of responses, were 5-10 microservices and 1-5 microservices at 36% and 34%, respectively.

This was followed by 20 or more microservices at 16% and 10-20 microservices at 14%.

This is interesting feedback as the amount of applications developers are working with seem to vary considerably. The amount of microservices lends to a better understanding of the amount of an application that is broken down into microservices. Considering the fact that most of the respondents are working on enterprise applications, applications with smaller amounts of microservices are definitely suggesting less microservices being actually implemented into the application.

How many micro services do you have in your primary application?



1-5 Microservices **34%**

5-10 Microservices **36%**

10-20 Microservices **14%**

20+ Microservices **16%**

Challenges With Microservices

Given the increasingly wide adoption of microservices among developers, there are no shortage of challenges that come with it.

Our survey asked developers for their biggest challenge in microservices development, and provided the following categories:

- Setting up the development environment locally
- Troubleshooting inter-service functionality issues
- Troubleshooting inter-service performance issues
- Scaling and monitoring in production
- Performance of the distributed system
- Other (Please Specify)

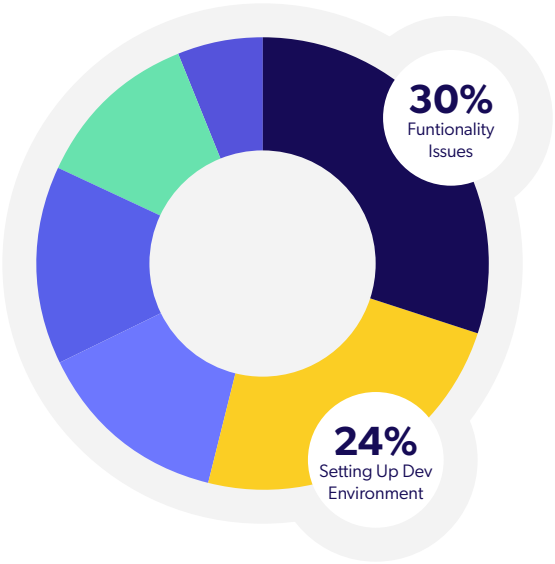
Troubleshooting inter-service functionality issues, at 30%, was the most widely reported challenge. XRebel provides developers with the ability to understand the way in which your application communicates between the different services, allowing the ability to fix those inter-service functionality issues while developing.

This is followed by 24% challenged with setting up the development environment locally. This is a bit surprising, considering the amount of microservices using Docker, but it definitely shows the problem presented by creating a complex microservice application.

Coming next, both with 14% of respondents are troubleshooting inter-service performance issues and scaling and monitoring in production.

Lastly, performance of the distributed system was reported by 12% of respondents, followed by 6% specifying other issues in an open response, with no statistically significant complaints among them.

What's the biggest challenge while developing microservices?



Troubleshooting Inter-Service Functionality Issues **30%**

Setting Up the Development Environment Locally **24%**

Troubleshooting Inter-Service Performance Issues **14%**

Scaling and Monitoring in Production **14%**

Performance of the Distributed System **12%**

Other **6%**

CI/CD

In a new section this year, we asked a few questions about Continuous Integration / Continuous Deployment (CI/CD) in Java application development.

Our queries were similar to the line of questions seen in previous sections of this report, wanting to know the tools that were used and the amount of time spent.

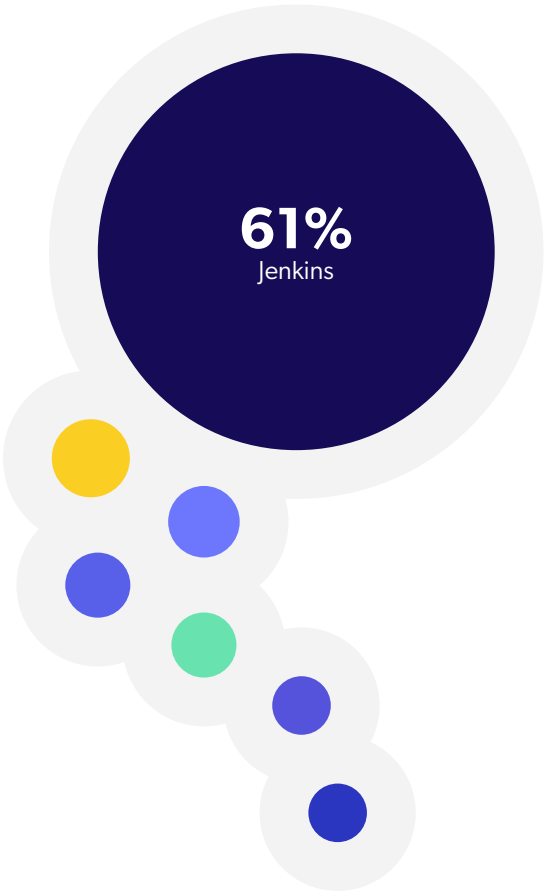
CI/CD Technologies

We are always curious about what tools Java developers are using, even when the results are not surprising.

Respondents were asked what CI/CD technologies they are using, with an overwhelming majority of 61% using Jenkins, nearly six times as many as the next most used technology.

After Jenkins, respondents were split relatively equally among the other answer choices: Bamboo (12%), TravisCI (11%), other technology (10%), no CI/CD technology (10%), TeamCity (9%), and Circle CI (9%).

Which CI/CD technologies are you using?



Jenkins **61%**

Bamboo **12%**

Travis CI **11%**

N/A **10%**

Other **10%**

TeamCity **9%**

Circle CI **9%**

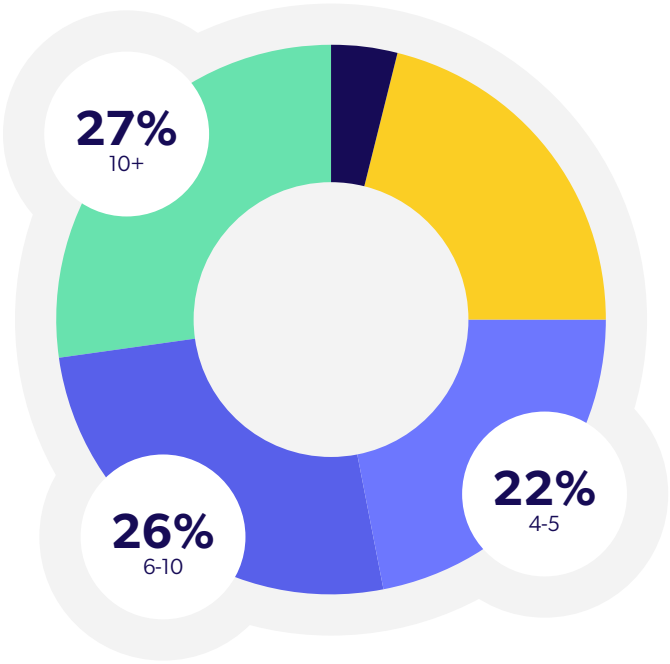
CI/CD Time

You know by now that we love this kind of question. Respondents were asked to provide their CI/CD build time on a scale of zero minutes to ten or more minutes.

CI/CD builds aren't particularly fast, so we are not surprised that less than 4% of respondents have a build time of 0-1 minute.

The remaining respondents were rather evenly split, with 10+ minutes and 6-10 minutes leading the way at 27% and 26%, respectively. These are followed by 4-5 minutes at 22% and 2-3 minutes at 21%.

How long does it take to complete your CI/CD build?



0-1 Minute **4%**

2-3 Minutes **21%**

4-5 Minutes **22%**

6-10 Minutes **26%**

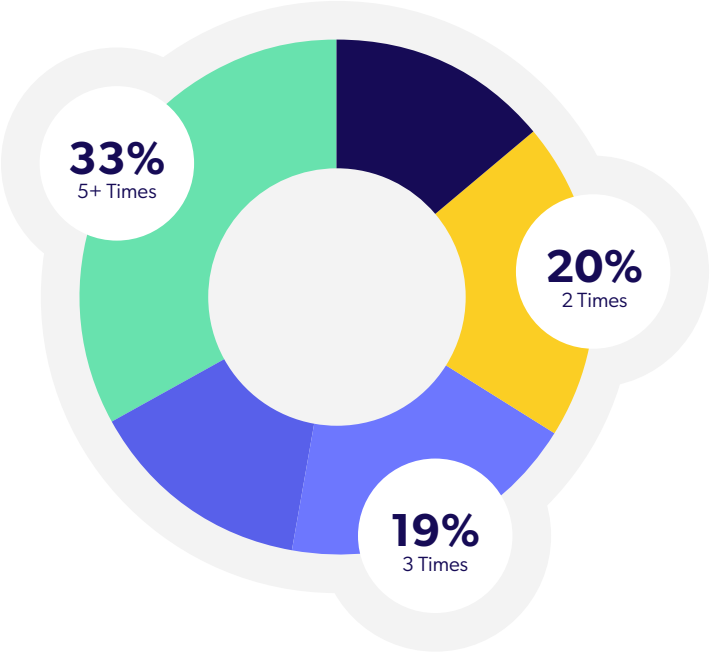
10+ Minutes **27%**

CI/CD Coding Frequency

For the final question in this survey, we asked developers how many times they commit code to their CI/CD build per day, with answer choices ranging from one to five or more.

The most popular answer was five or more commitments of code to the CI/CD builds per day. Next most was two commitments at 20%, three commitments at 19%, and one and four commitments each receiving 14% of responses.

How many times do you commit code to your CI/CD build per day?



1 Time Per Day **14%**

2 Times Per Day **20%**

3 Times Per Day **19%**

4 Times Per Day **14%**

5+ Times Per Day **33%**

Summary

In this report, we explored the tools and technologies used by Java professionals, what issues they are facing in development, how they are spending their time while waiting for redeploys (and how would use their time, if they had more), the adoption and effect of microservices, as well as a few data points about CI/CD.

Summary

Tools and Technologies

Just like last year, we continued to see the effects of microservices in the tools and technologies that were most used. Spring Boot, Tomcat, and Docker remain the tools of choice among developers.

Developers are clearly maintaining usage of Java 8, with very few enterprise companies transitioning to the latest long-term support version. We will see how development teams respond as the Java community moves further away from Java 8.

Application Performance

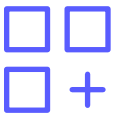
Like the above category, our findings in this area remain largely unchanged from last year. A long application response time continues to be the most common performance issue, with memory leaks and high CPU usage as the next most common.

This further strengthens our belief that developers will continue to be asked to test application performance during development. This is often asked without any insight into how their code is impacting the application performance, though a tool like XRebel can provide visibility in this scenario.

Developer Productivity

As is always the case, the biggest time issue for Java developers is the amount of time spent waiting for redeploys. These issues aren't disappearing with the adoption of microservices. Nearly 90% of developers are waiting over two minutes per redeploy, 60% waiting over four minutes, and 40% waiting for longer than six minutes.

All of this downtime leads to a lack of productivity, with only 20% of respondents indicating that they are performing other work tasks during redeploys.



Microservices

While there has been a clear shift to microservices within the Java community — from the tools used, the composition of teams, and challenges faced — it appears that we may have reached a temporary saturation as adoption has leveled off in comparison to last year.

We are curious to see if adoption inches forward at all in future surveys. In any case, the wide adoption of microservices continues to bring challenges, which will only grow as more technologies and integrations are introduced as the Java community evolves.

CI/CD

Not surprisingly, the data indicated that the most dominant tool of choice, Jenkins, continues to be just that.,Build times are as expected. Of note in this section, however, is that many developers are making regular commits of code (more than five per day) to their CI/CD build.

Improving Java Development

Want to build better Java applications faster? Our two Java development solutions, JRebel and XRebel, help Java developers seamlessly find, fix, and instantly check code changes in Java-based applications.

JRebel

About JRebel

JRebel lets developers skip redeploys while maintaining application state. This allows developers to make changes to code and instantly check to see if those code changes fixed the issue.

Try JRebel for Free

Want to see how JRebel works on your project? Try it free for 10 days with no commitment.

Try JRebel For Free

XRebel

About XRebel

XRebel can help developers to find and fix performance issues during microservices development. It enables developers to easily understand application structure, and which parts are used in executing their code.

Try XRebel for Free

Want to see what XRebel can do for your project? Click the button below to get a free, 10-day trial.

Try XRebel For Free