JRebel

2020 Java Developer Productivity Report

Java Tools, Technologies, and Application Performance

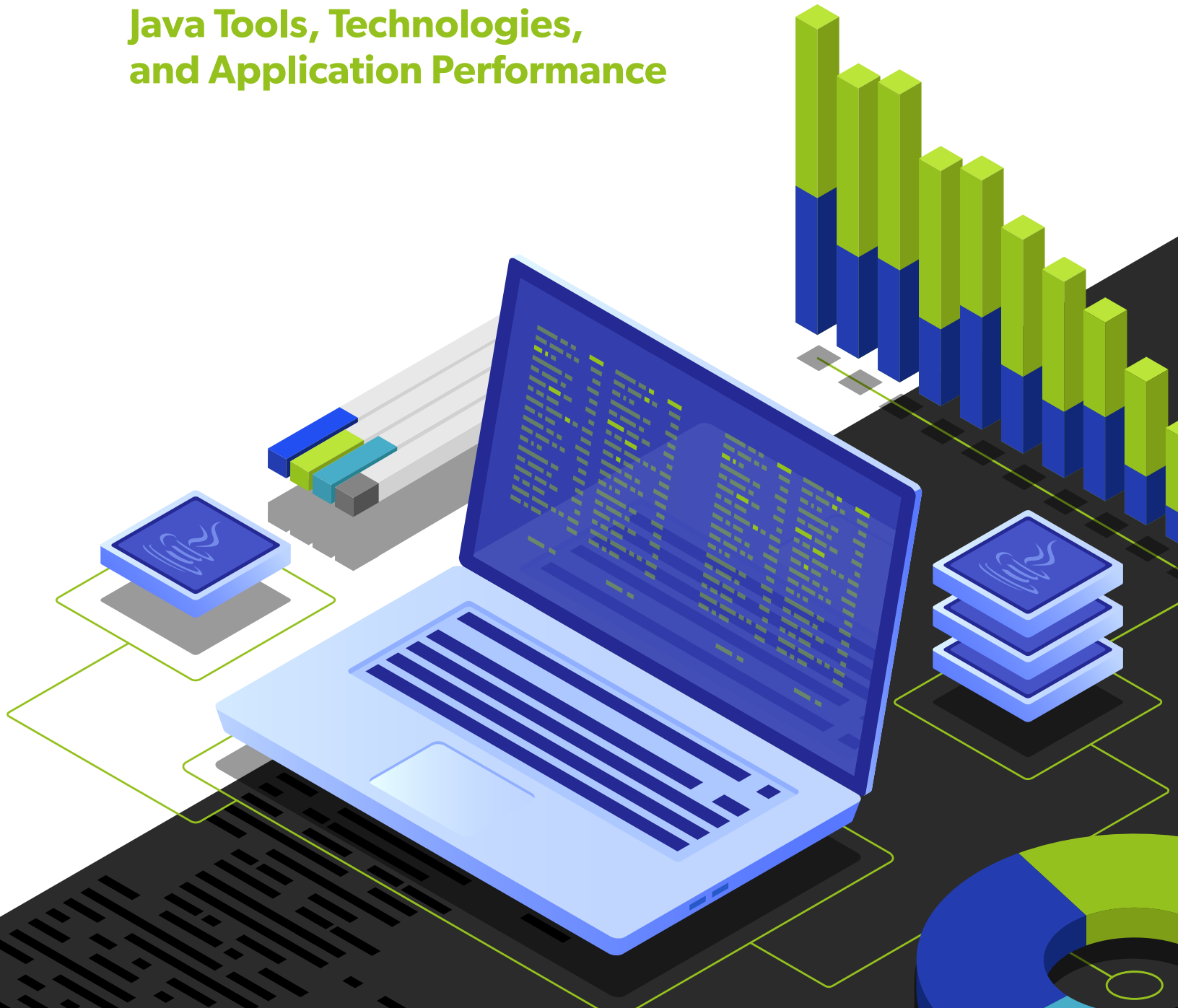


Table of Contents

2020 Java Developer Productivity Report: Introduction

Welcome to the 2020 Java Developer Productivity Report by JRebel. This report is based on a survey of Java developers from around the world and gives valuable insights into how Java development teams are using languages, tools, and technologies in their day-to-day work.

One of the key focuses in this year's report is the impact of microservices adoption in Java for everything from DevOps considerations to technology stacks and the potential challenges developers can experience.

Java Developer Productivity Report:
Overview

To the reader,

One of the best traditions we have here at JRebel is our regular developer survey reports. Now in our 8th year of compiling these reports, our motivation is largely the same — we want to understand how the Java community is evolving.

The JRebel team has always been devoted to providing the best level of understanding of the Java development community. Developers are what motivate us towards bigger and better things within the community.

Seeing which Java technologies developers are using — and how Java technology usage changes over time — helps us to understand the undercurrents of the Java ecosystem.

Beyond Java technology usage, we want to learn more about the problems developers experience during development and whether or not they feel existing technologies are helping to solve these problems.

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.

Because we promote the survey via social media, we understand that many of the respondents may have closer ties to our community than the typical Java developer. Still, we believe that these reports can give a significant and valuable glimpse into the Java development community as a whole.

We hope that you enjoy the report and we look forward to joining in the discussions (and arguments) that it will generate.

Thanks for reading,
The JRebel Team

Java Developer Productivity Report:
Data

Our report is comprised of data collected during a public survey conducted from September through November 2019. We received 399 total responses.

The survey focused primarily on technologies used, performance issues, microservices, and respondent/company information.

JAVA TECHNOLOGIES

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

JAVA PERFORMANCE

For performance, we surveyed common performance issues, average redeploy times, project requirements, performance verification method, and rate of performance problems reaching production.

JAVA MICROSERVICES

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

JAVA DEVELOPER AND COMPANY INFORMATION

For respondent and company information, we surveyed for job title, company size, and development team size.

JAVA DEVELOPER SURVEY METHODOLOGY

To make the report more scannable, we created graphs that disregard outliers. For this report, we considered any question choice with fewer than 1% of respondent choice to be an outlier. The goal of this was to help us to focus more on the big picture of the results without spending too much time discussing a technology that .003% of our respondents use.

Java Developer Productivity Report: 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 experience level with Java, development team size, and company size.

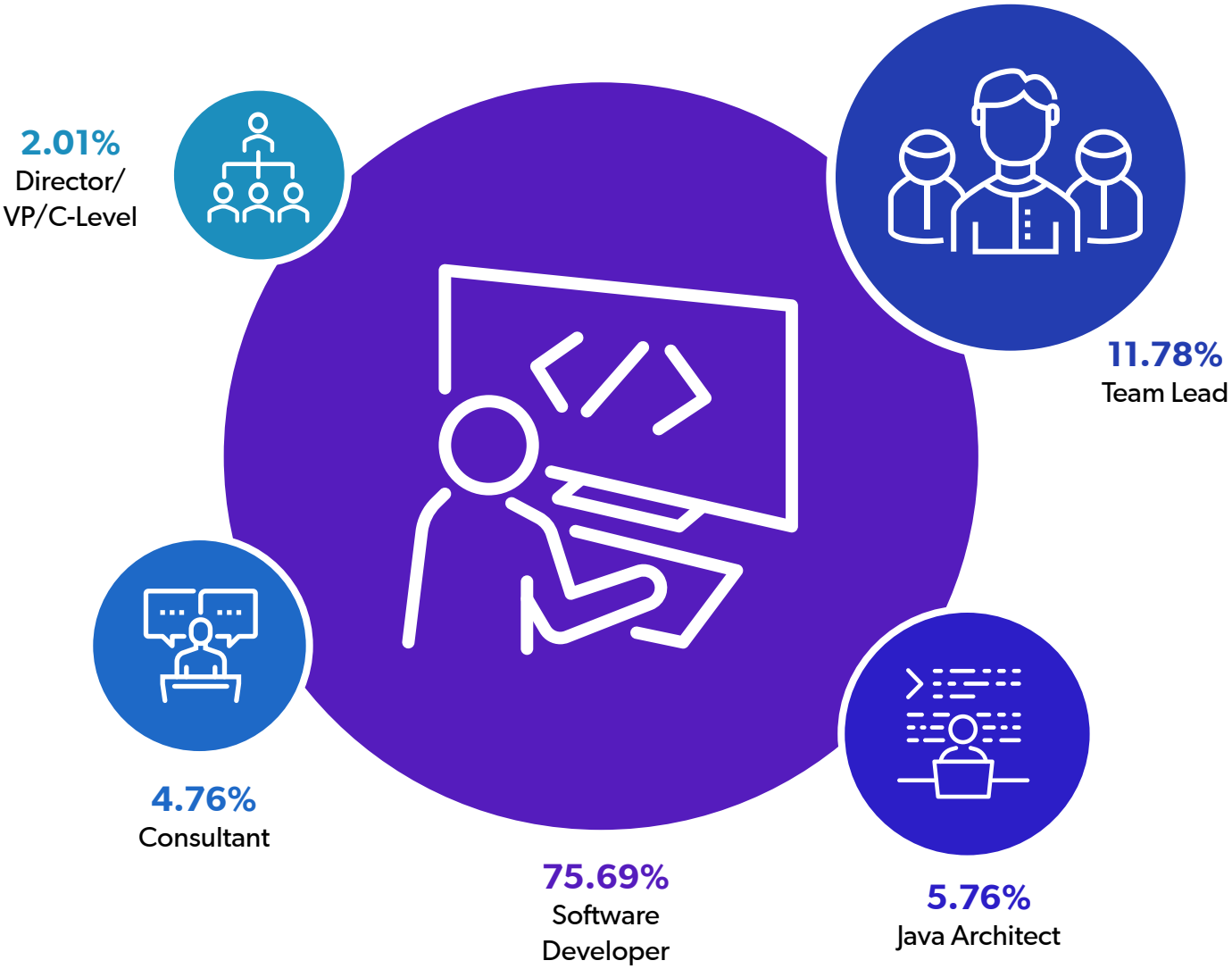
Job Titles

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

Not surprisingly, the majority of our respondents self-identified as Java developers at 76% of those surveyed. 12% of respondents identified as Team Lead, while Java architects and consultants identified at 6% and 5% respectively. Lastly, 2% of respondents identified as Director, Vice-President or C-Level executives.

The vast majority of our respondents serve in technical roles, which makes us confident that the data collected is highly relevant to the intention of our survey.

Which job title best matches your current role?



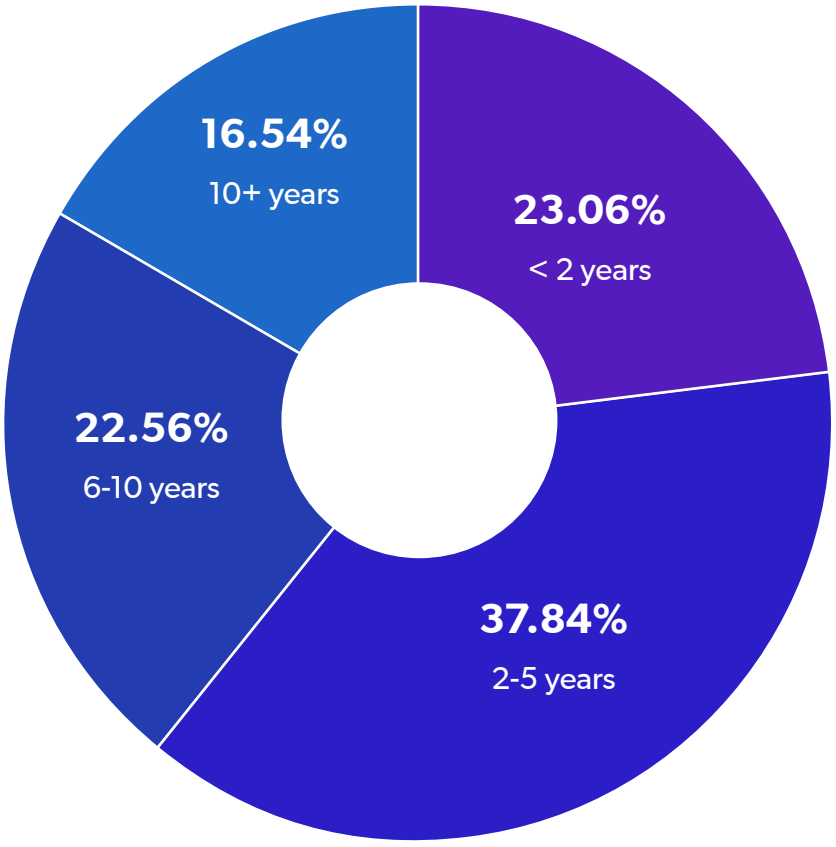
Experience

Experience can provide additional clarity into the respondent data we collected. While we can gain some understanding of developer knowledge via job title, knowing how many years of experience the average respondent has with Java gives further clarity of their authority in their Java ecosystem.

Our survey respondents were very experienced, with the majority of respondents having over two years of experience in working with Java. At 38%, respondents most-commonly reported having two to five years of experience. Respondents that had either under two years or between six and ten years experience with Java shared the 23% mark. Lastly, respondents with over ten years of experience reported in at 17%.

What does that mean for the survey data? Mostly that the survey responses reflect the gamut of Java development and that some of the respondents have been working with Java from around its inception.

How many years of experience do you have with Java?

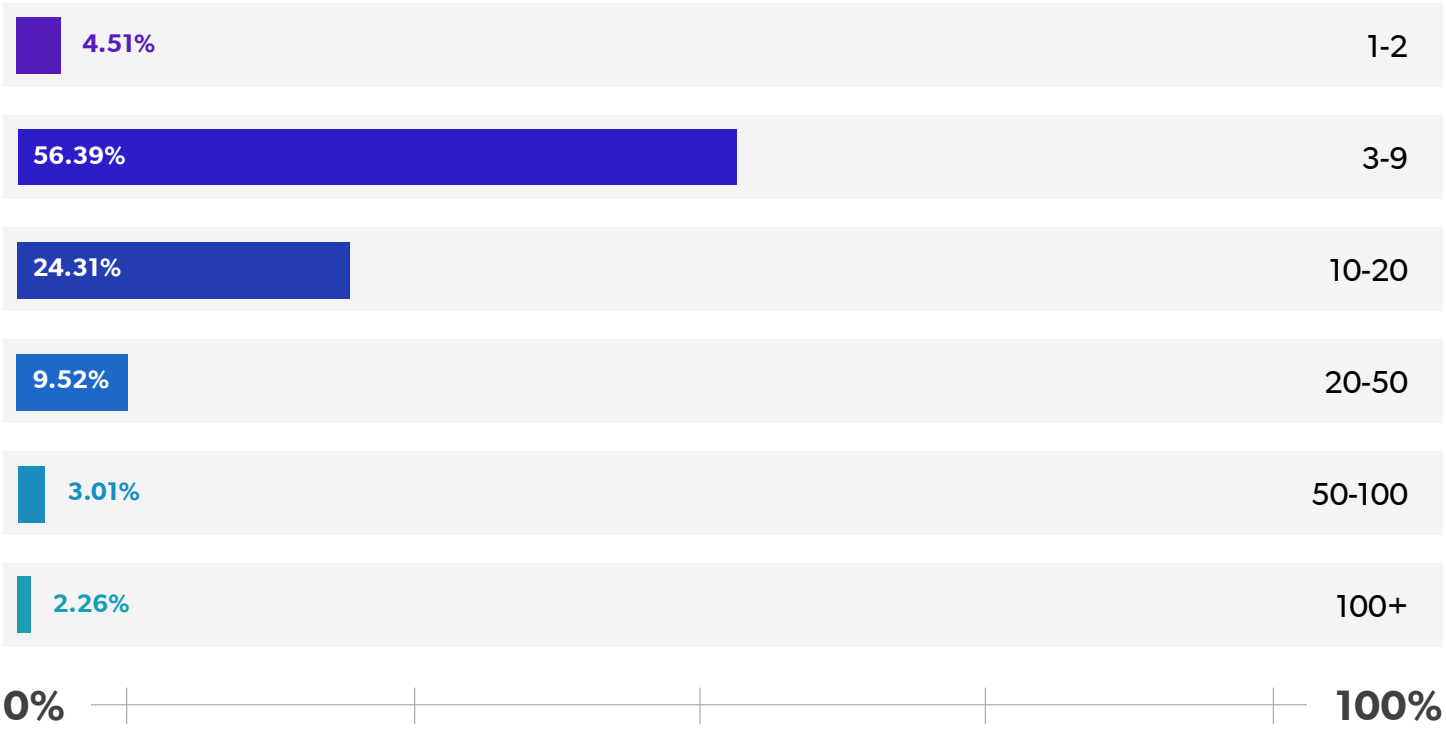


Team Size

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. We also wanted to get a grasp of the changes occurring in Java communities when it comes to to smaller, agile development teams.

The results showed respondent team size largely following previous survey results, with 56% 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 10%, respectively. Teams of one to two people made up 5% of respondents, with teams of 50-100 and 100+ people comprising 3% and 2% respectively.

What is the size of your team?



Company Size

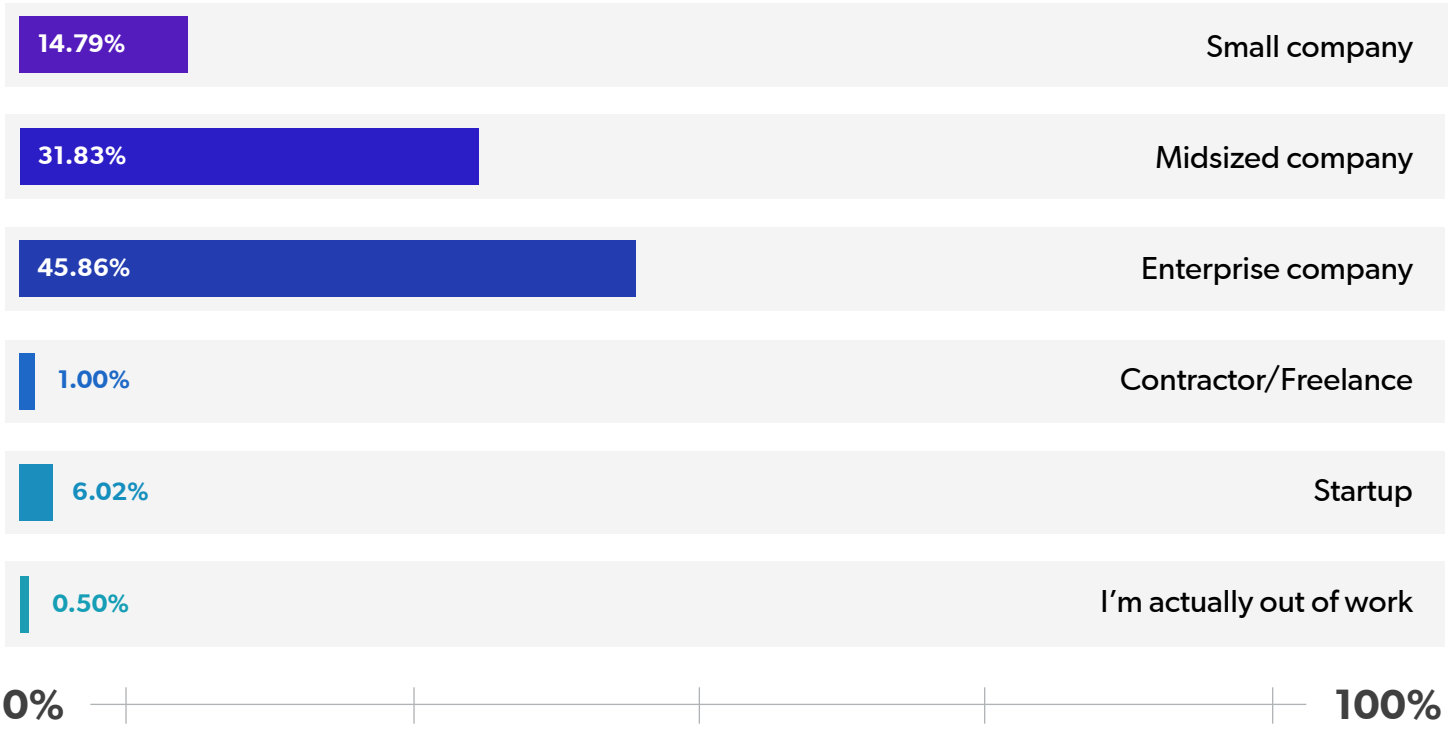
Company size can give valuable information on how to frame questions regarding Java technology choices. We asked respondents to classify the company they work for into six categories:

- Small company
- Midsized company
- Enterprise company
- Contractor/freelance
- Startup
- Unemployed

Almost half of respondents (46%) reported working in an enterprise company. Respondents reported working in midsized and small companies at 32% and 15% respectively. The remaining respondents fell primarily in startup companies at 6% and contractor/freelance at 1%.

Lastly, and in a bit of good news, a statistically insignificant percentage of respondents reported being out of work.

What is the size of your company?



Java Developer Productivity Report: Technologies

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

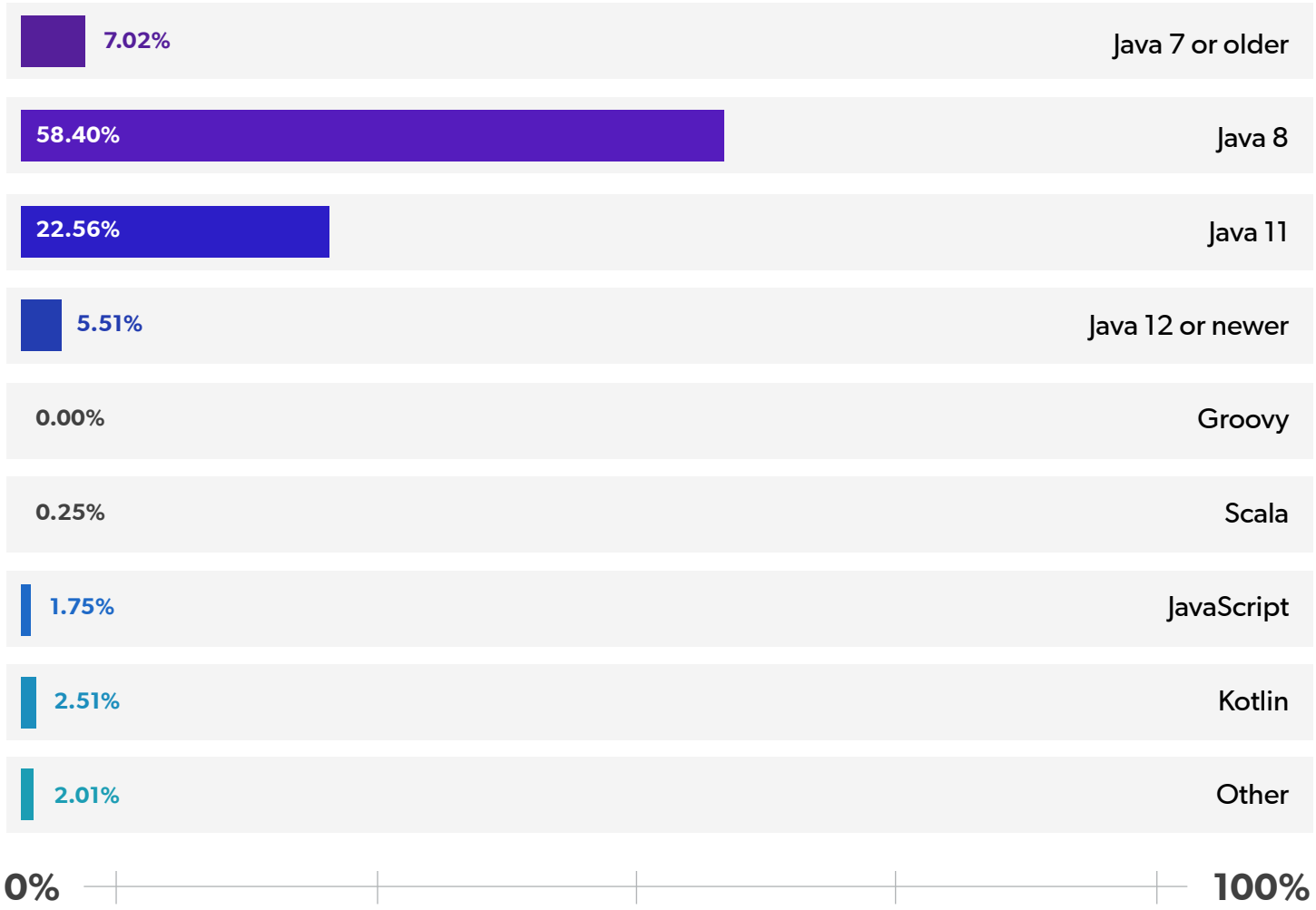
While we understand that many of the people surveyed are relatively close to our products, (i.e. they follow us on social media or are part of one of our email lists, or responded to an advertisement) we do feel that the tool and technology usage represented in this survey report are fairly representative of the Java development community.

Language

At 58%, the majority of respondents reported using Java 8 as the programming language of choice in their main application. Java 11 was the next highest at 23% of respondents. 7% of survey respondents reported using Java 7 or older, with another 6% of respondents reporting using Java 12 or newer. Kotlin and JavaScript were the least-used at 3% and 2% respectively.

For us, it wasn't a big surprise to see Java 8 as the dominate programming language. It was surprising, however, to see that only 23% of respondents were using Java 11. We think this has to do with the lack of truly impactful updates to Java since Java 8 and the cost to update the JVM. For respondents still working with Java 7 and older, we expect that number to dwindle as more applications continue to migrate to Java 14 or whatever number is currently available.

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



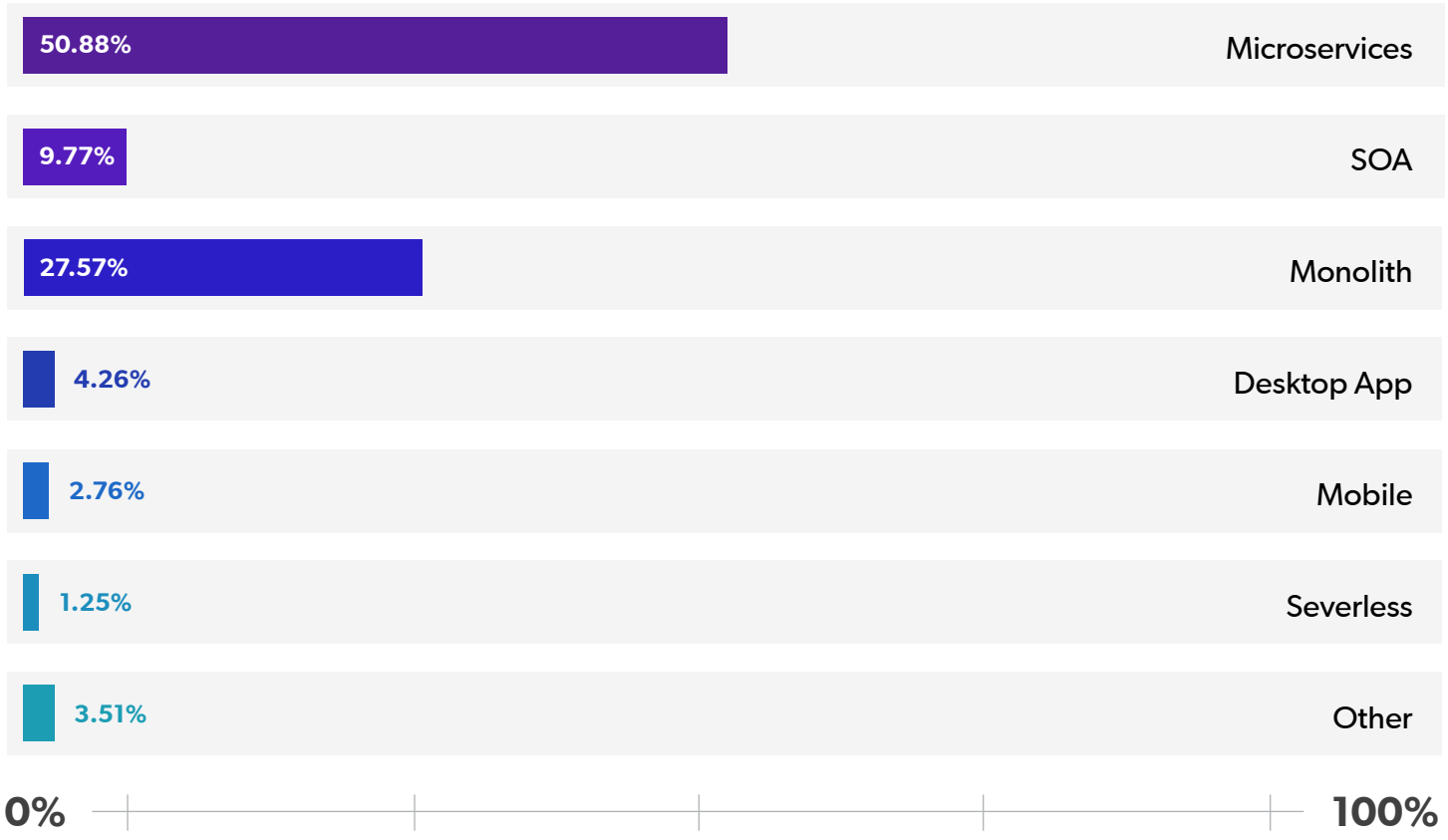
Application Architecture

Application architecture is one of the most determinant factors for which technologies developers use during development. And, with the majority of respondents now working on microservices-based applications, you can see that effect throughout our survey results.

But it's also important to note that the microservices revolution isn't all the way complete. While over 50% of respondents use microservices architecture for their main application, 28% are still working within a monolithic architecture, and 10% are working on SOA-based applications.

Desktop, mobile, and serverless applications made up the rest, with 4%, 3%, and 1% of survey respondents reporting usage, respectively.

What is the architecture of the main application you develop?



Application Server

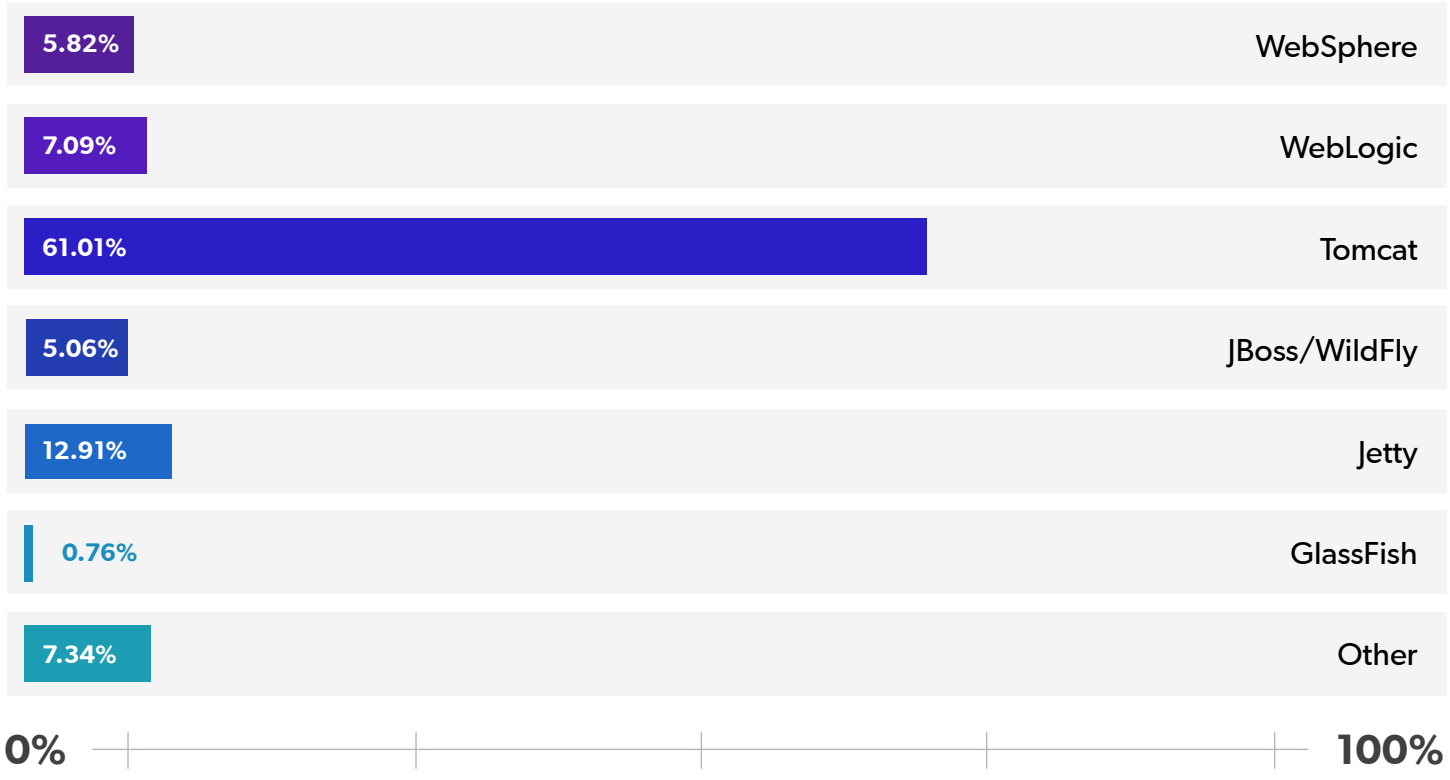
Is application server variety dying? Our survey results show that developers are using Tomcat at nearly five times the rate of the closest competitor.

With 61% percent of respondents using Tomcat on their main application, it's clear that Tomcat is the dominant application server. Indeed, the next highest application server was Jetty at 13%, with WebLogic and WebSphere at 7% and 6% usage respectively. Lastly, JBoss/Wildfly usage came in at 5%.

It's clear that Tomcat is doing something right to garner such a large share of the market. For us, at least part of that popularity is due to Tomcat making itself highly compatible with other platforms, like Hybris, Spring Boot, Docker, AWS, and others. Another major aspect is its cost, it's free.

Previously, the lack of support surrounding the application server was one of the core reasons why people sought out application servers like Websphere, Weblogic, and JBoss. It appears that what has happened is Java architecture companies have decided the best way to support application servers is within their architecture as opposed to exclusively in the application server.

What application server do you use on your main application?



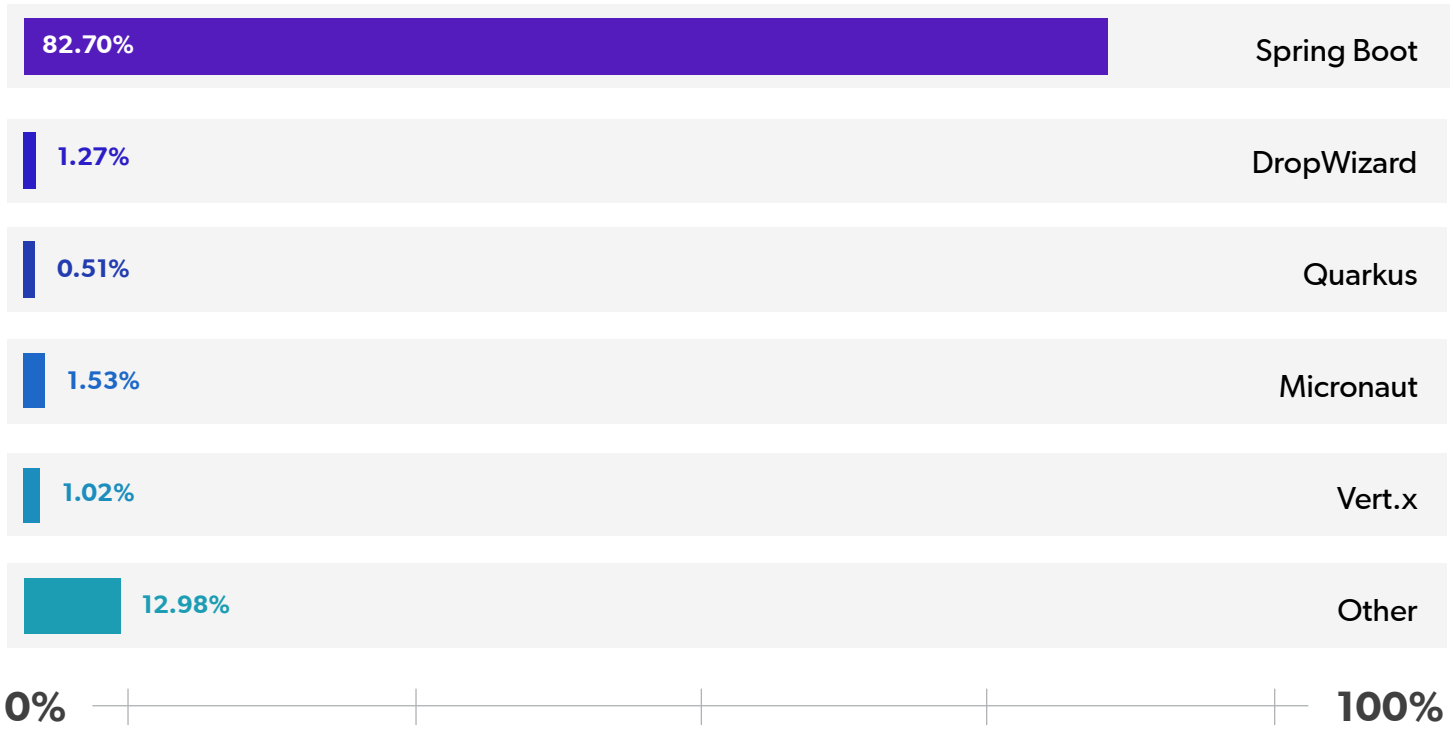
Runtime Platform

The far and away favorite selection for runtime platform is Spring Boot at 83%. Spring Boot, of course, has been the preferred Java framework for several years now. That's due, at least in part, to the increased adoption of microservices for Java applications in recent years.

At 2% of total respondents, Spring was tied for the second most popular choice, with respondents not using a runtime platform also reporting in at 2% of the total.

Respondents using Dropwizard, Micronaut, Vert.x, or custom platforms each came in at 1% a piece.

What runtime platform does your application use?



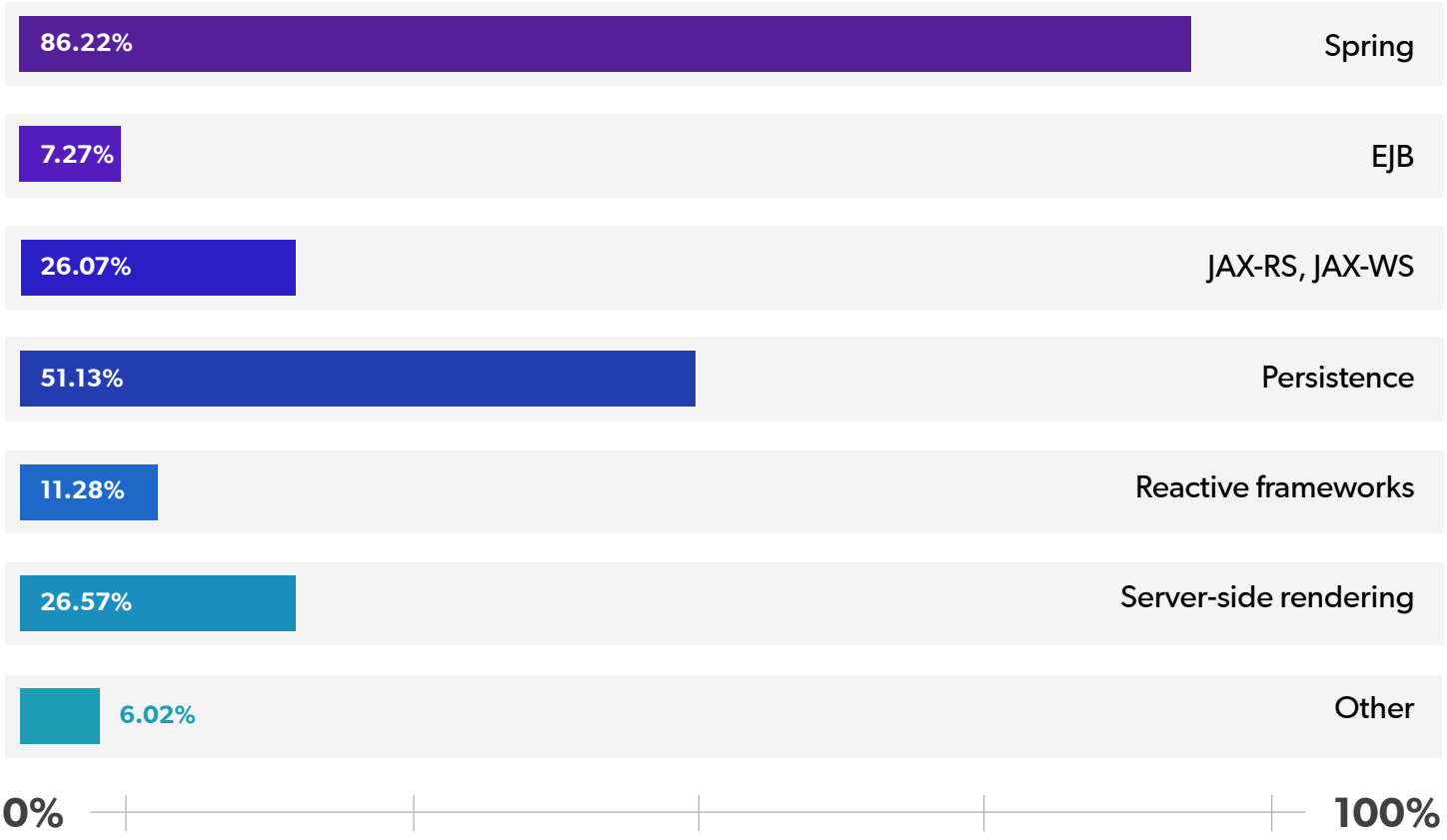
Framework Technologies

This question asked participants to select the application frameworks and technologies being used on their main project. This question allowed for multiple response as most applications use a variety of different framework technologies.

At 86%, most respondents were working with Spring. 51% of respondents reported working with persistence technologies like Hibernate, OpenJPA, or EclipseLink. 27% reported using server-side rendering technologies like JSP, JSF, Thymeleaf, FreeMarker, or GWT. 26% reported using JAX-RS or JAX-WS technologies like Jersey, RESTEasy, CXF, or Axis.

For reactive frameworks, 11% of respondents reported using technologies like Vert.x, Akka, RxJava, or Project Reactor. Lastly, 7% of respondents reported using enterprise JavaBeans in their main project.

What Java application frameworks and technologies are you using in your main project?

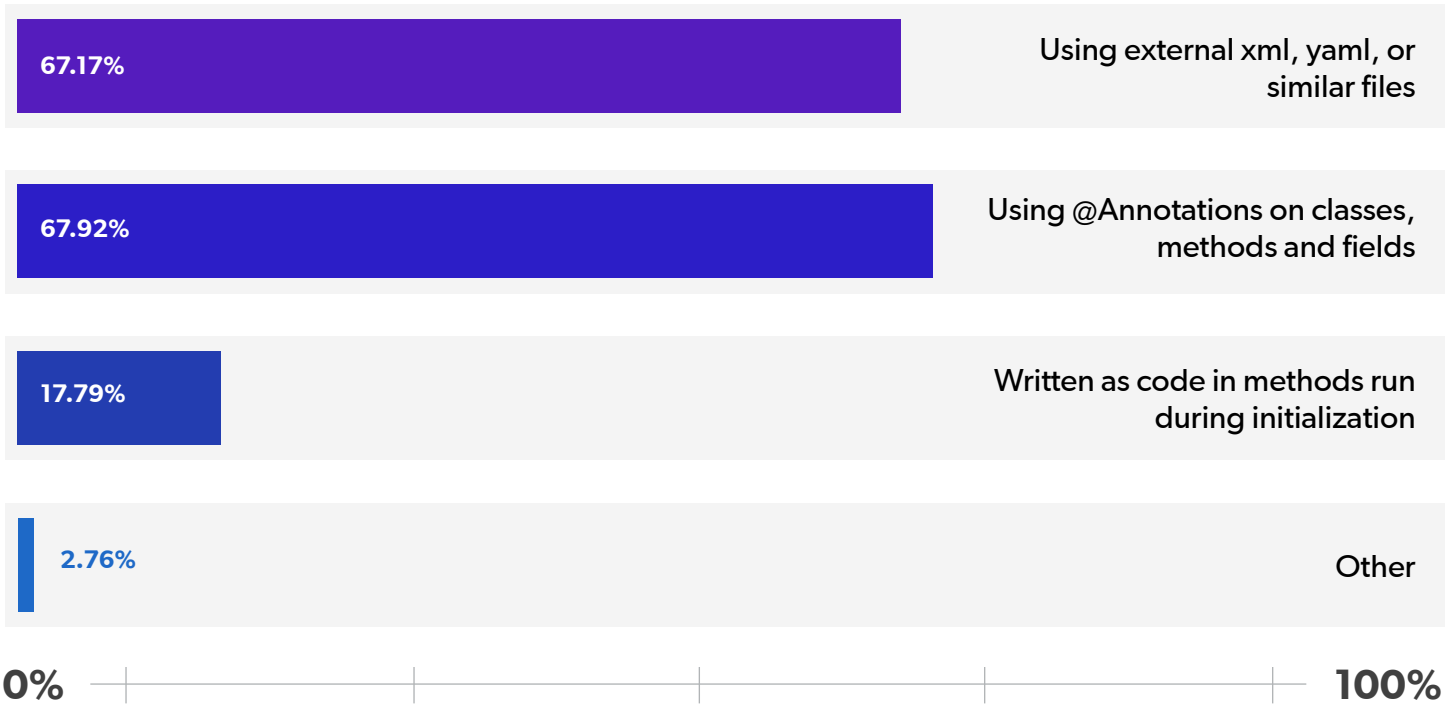


Framework Configuration

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

68% of users reported using @Annotations, with 67% of users using external xml, yaml, or similar files. 18% of respondents were configuring with code added to methods that run during initialization.

How do you configure most of your frameworks?



IDE

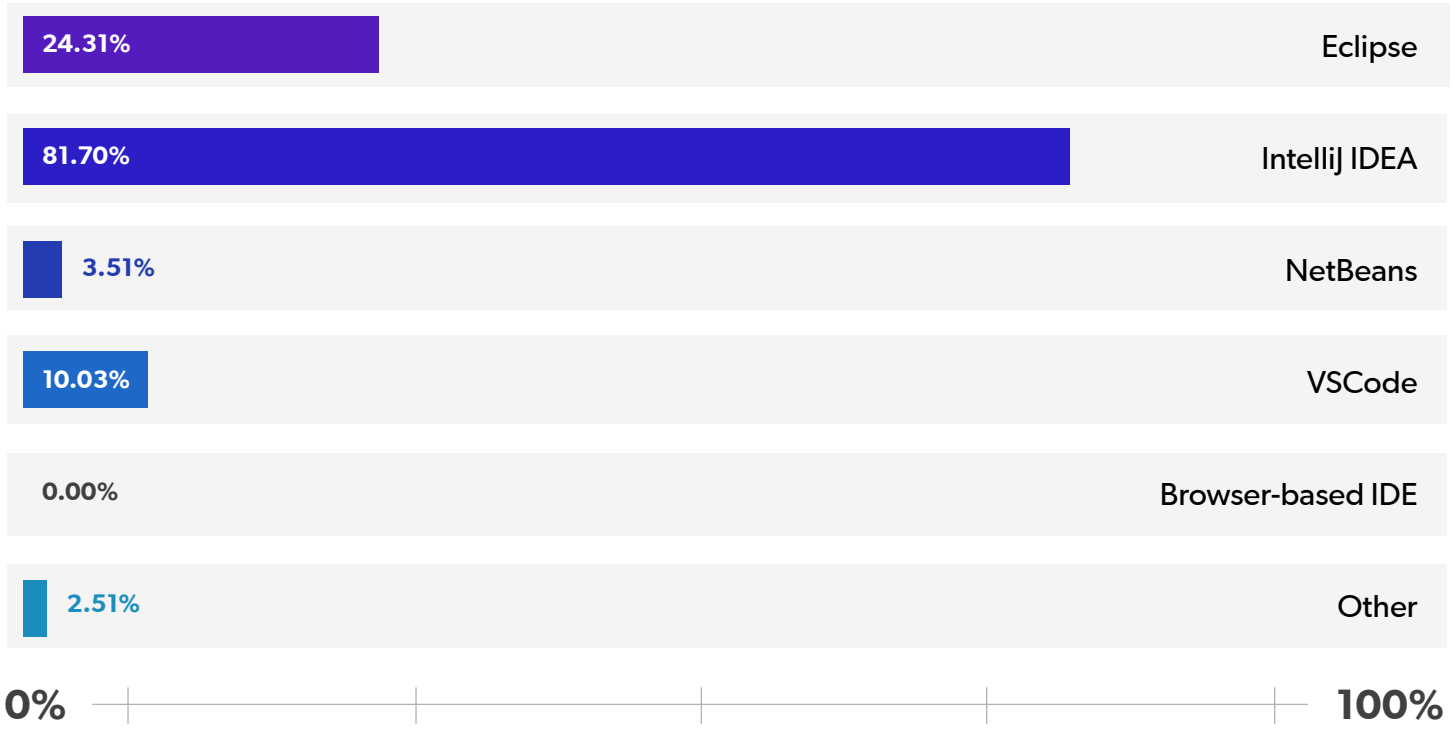
In this question, we asked developers to list the Integrated Development Environment (IDE) they use professionally.

As recently as a few years ago, developers would have questioned the value of spending money on an IDE.

But, with 82% of respondents using IntelliJ IDEA, it's clear that this paid IDE is well worth the price of admission.

The next most used IDE, Eclipse, comes in at 24% with VSCode coming in at 10%. Lastly, 4% of respondents reported using NetBeans in a professional capacity. Netbeans has really lost all its interest through the years with even VSCode passing it in this survey.

What developer IDE do you use professionally?



JRE/JDK Distribution

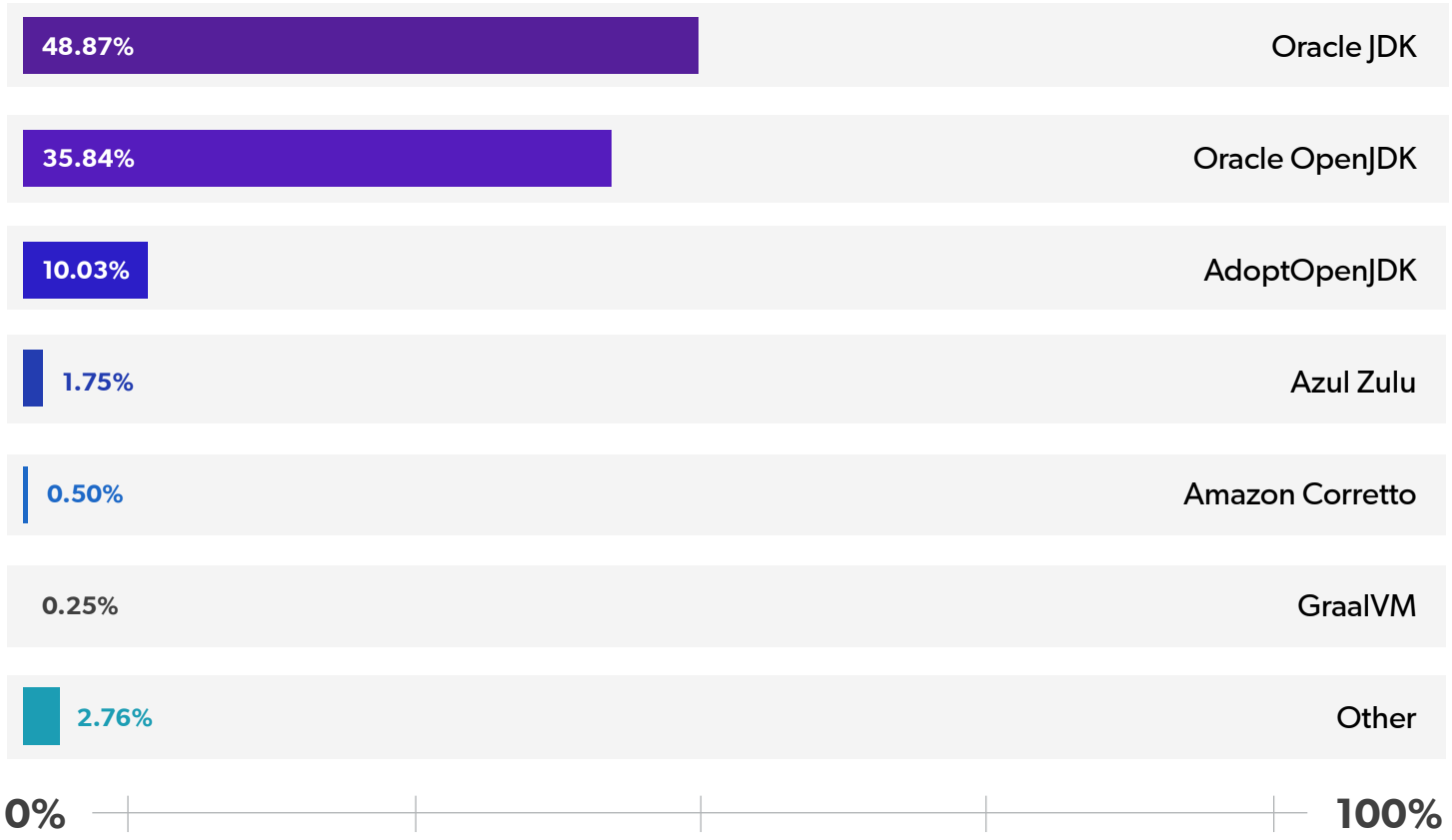
In this survey question, we asked respondents to select the JRE/JDK distribution they use. And, in one of the more surprising results from this survey, we found that nearly 50 percent of respondents were using the paid Oracle JDK. That's 13% more than respondents using Oracle OpenJDK — which reported in at 36%.

We think that may be due to two reasons:

1. The large Java 8 developer demographic represented in our survey. We think that the number of developers using Java 8 suggests that the project hasn't been updated in a long period of time which could be what has kept the applications in the Oracle distribution.
2. Organizations, particularly Enterprise-sized, are still finding value from the support provided by Oracle for OracleJDK.

Aside from Oracle OpenJDK and Oracle JDK, 10% respondents reported using AdoptOpenJDK, while another 2% reported using Azul Zulu.

What JRE/JDK distribution do you use?

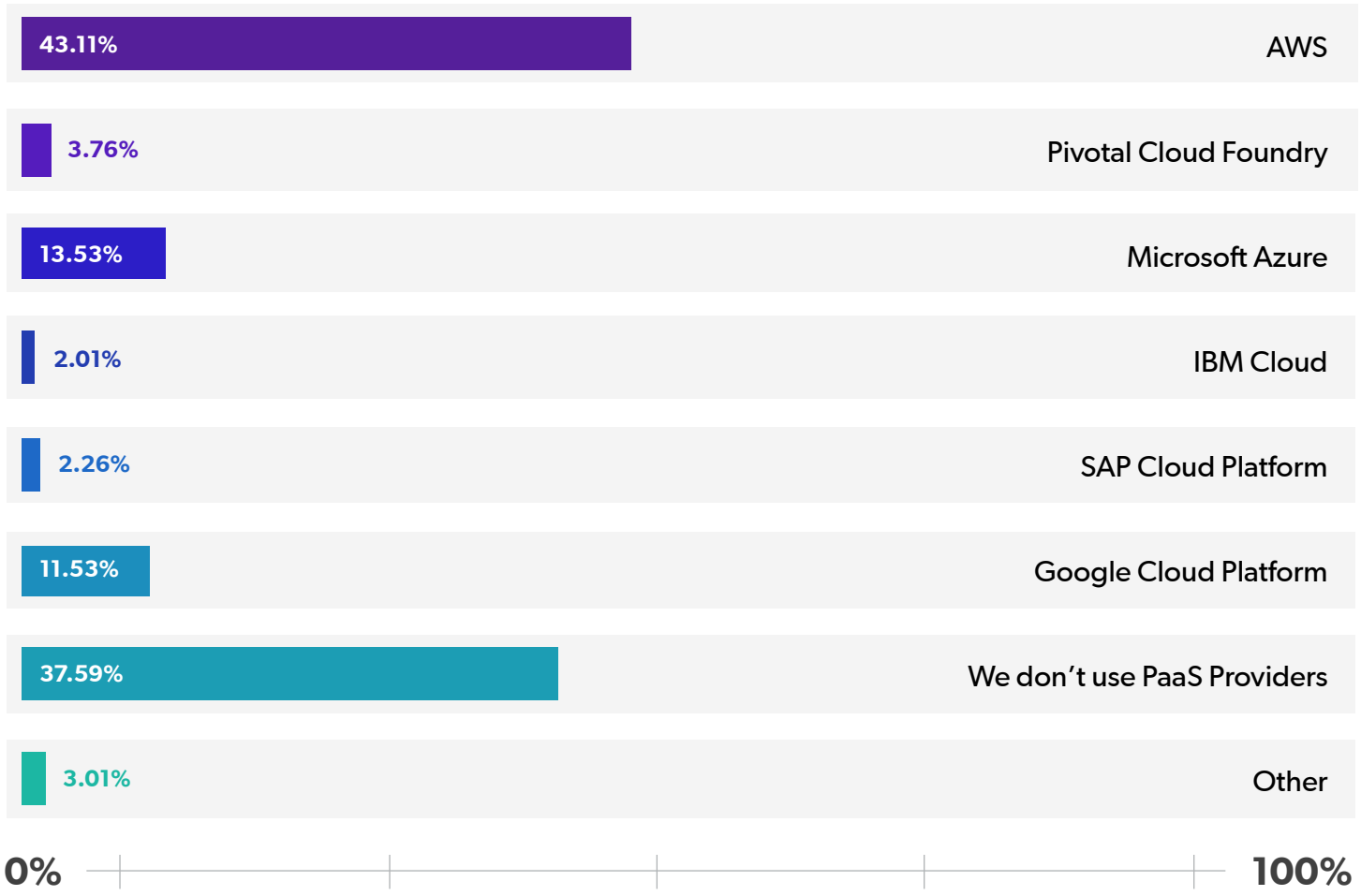


PaaS Provider

In this question, we asked respondents to select which PaaS provider they used. Of the 60% of respondents using a PaaS provider, 43% use AWS, 14% use Microsoft Azure, and 12% use Google Cloud Platform.

Lesser-used providers included Pivotal Cloud Foundry at 4%, with SAP Cloud Platform and IBM Cloud at 2% each.

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



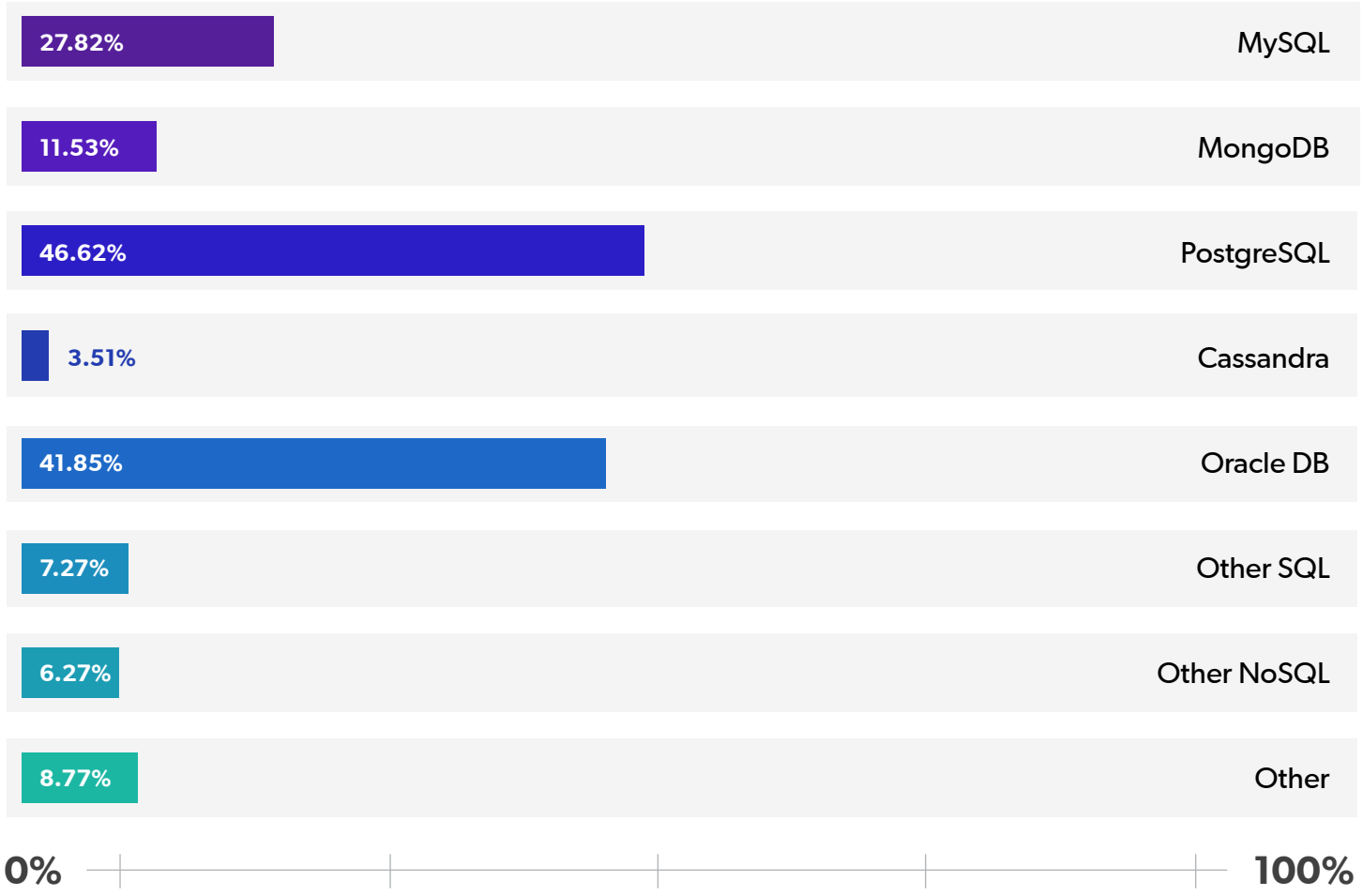
Database

In this question, we asked respondents to select the database(s) they're currently using. The most popular was PostgreSQL at 47%, with Oracle DB a close second at 42%.

Next up was MySQL, with 28% of respondents reporting usage. MongoDB was in fourth place, with 12% of respondents.

The least-used databases included SQL, NoSQL, and Cassandra with 7%, 6%, and 4% usage, respectively.

What database are you using?

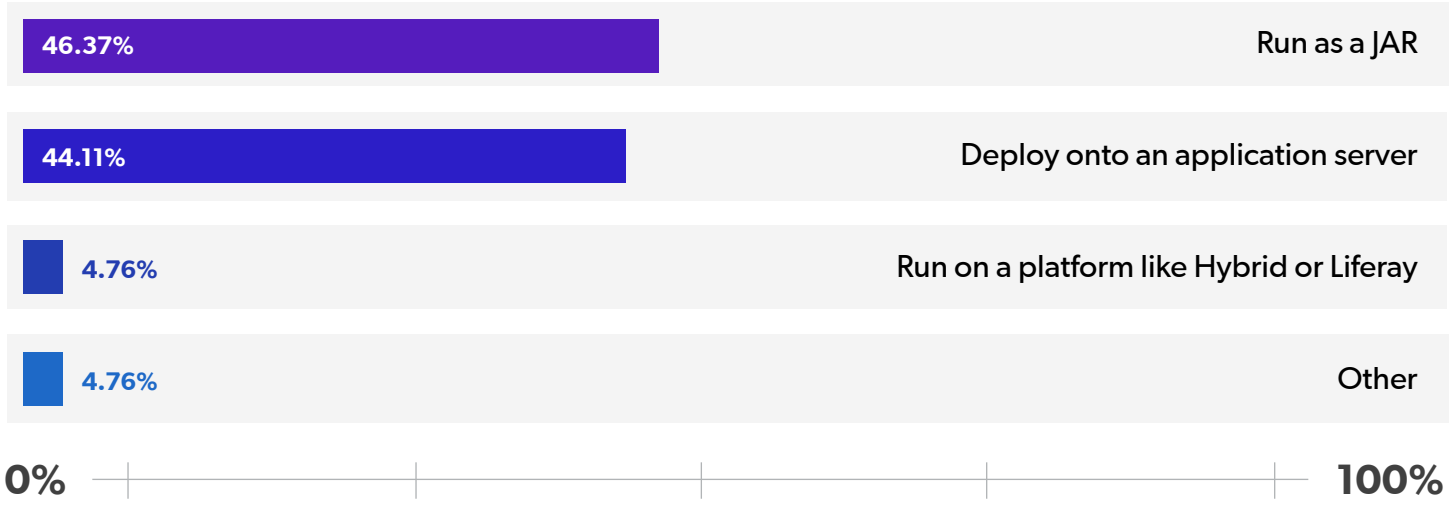


Deployment Model

In this question, we asked respondents to tell us which deployment model they use. The most popular deployment model, at 46%, was to run the application as a JAR.

44% of respondents reported deploying onto an application server, while 5% of respondents reported using a platform like Hybris or Liferay to deploy their application.

Which deployment model are you using?



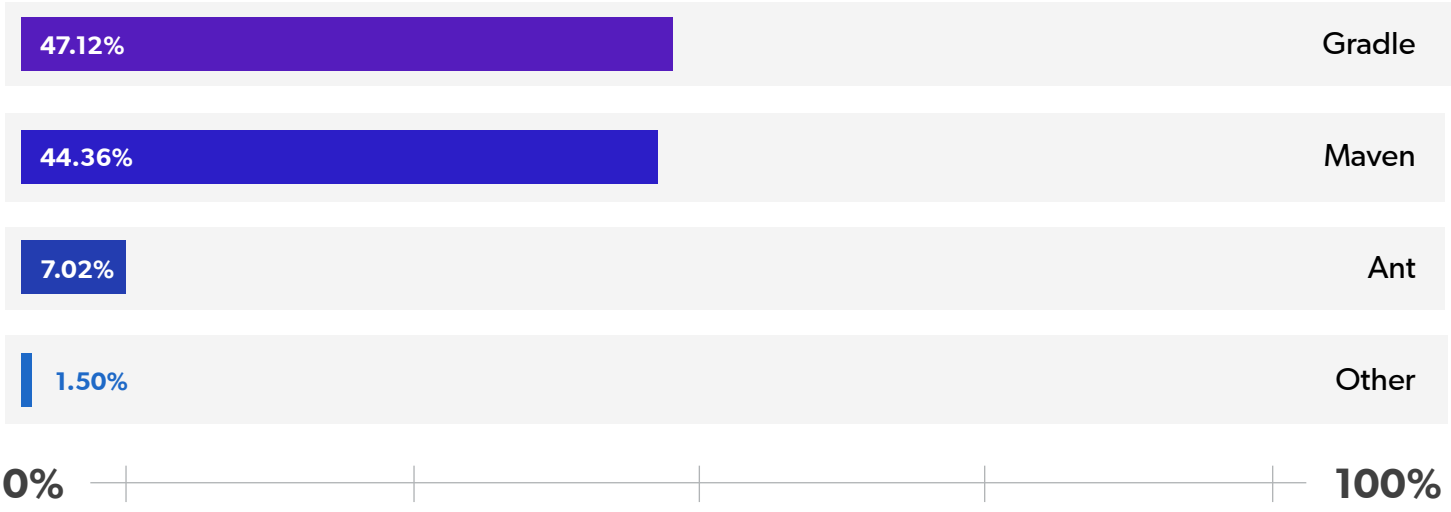
Build Tools

For build tools, we asked developers to pick the tool they use on their main application. Expectedly, Maven and Gradle were the most-used tools. But, perhaps reflecting the amount of respondents working in mobile development, Gradle was the more popular of the two.

We don't think that this reflects the reality of Maven vs Gradle adoption — but it's interesting to see, regardless.

Our survey found that 47% of respondents were using Gradle as a build tool for their main application, while 44% were using Maven. Ant usage trailed both at 7% of respondents.

What build tool do you use in your main application?



Virtualization Tools

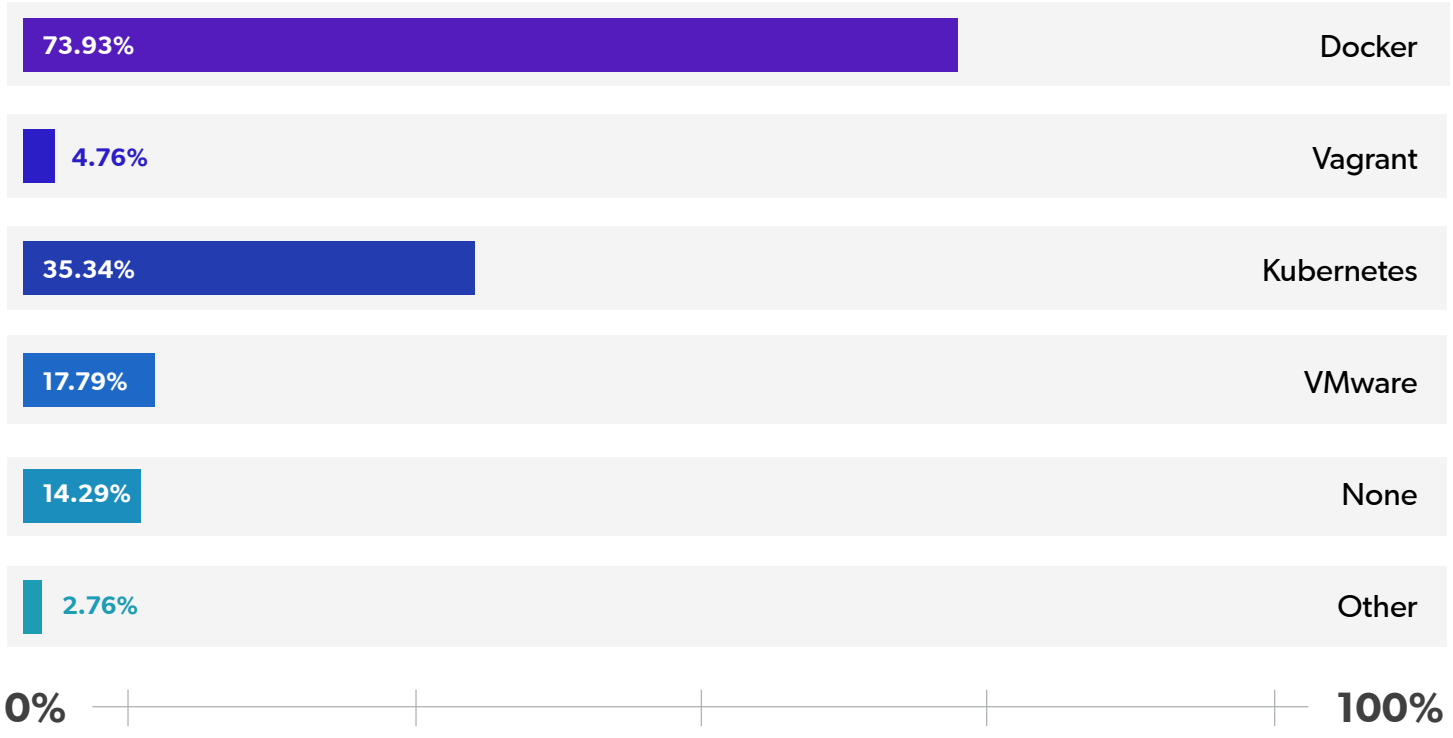
In this question, we asked developers which virtualization tools they use. Far and away the most commonly-used tool was Docker, at 74%. Considering the percentage of respondents using microservices, this wasn't a big surprise.

Kubernetes was the second most popular virtualization tool at 35%, which shows the continued growth Kubernetes has experienced over the last couple years. With the way Kubernetes has grown it wouldn't be surprising to see it play a dominating role in the market in coming years.

VMware and Vagrant made up the last two significantly used tools, with 18% and 5% usage respectively.

Lastly, 14% of respondents reported not using a virtualization tool.

Which virtualization tools do you use?



Java Developer Productivity Report:
Application Performance

Application performance is one of the primary focuses for us as a company — after all, we want our application performance tool XRebel to solve performance issues for developers.

And, just like Java technology usage rates, these performance issues grow and shift as new ideas work their way into the Java ecosystem.

Take microservices as the glaring example, and one we'll talk about at length in the next chapter: they rely on independently functioning services that can communicate and interact with other services within an application.

But, because of the rapid advances in how these individual containers can interact, and the seemingly infinite combinations of ways that they can cause performance issues as the application grows more complex, microservices have proven to be a veritable Pandora's box of new performance issues.

In the following question responses, we see the shifting nature of these application performance issues, and the increasing call for developers to address them during application development.

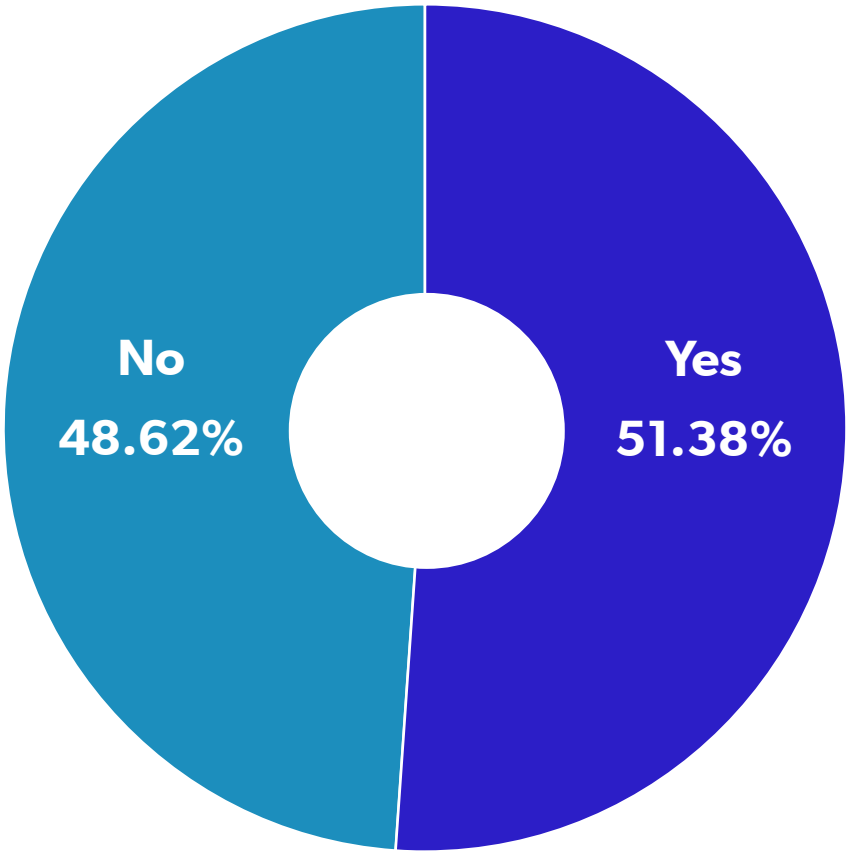
Non-Functional Requirements for Performance

The days of sending potentially non-performant code to deployment are over — at least for half of our respondents.

51% of respondents reported that they have non-functional requirements defined for performance in their development project, with 49% reporting the opposite.

Is this due to the necessities of microservices development? Or the shifting development responsibilities with DevOps? Or is this just an emerging focus for developers? Regardless, more developers have non-functional requirements for application performance than those who do not.

Do you have non-functional requirements defined for performance in your project?



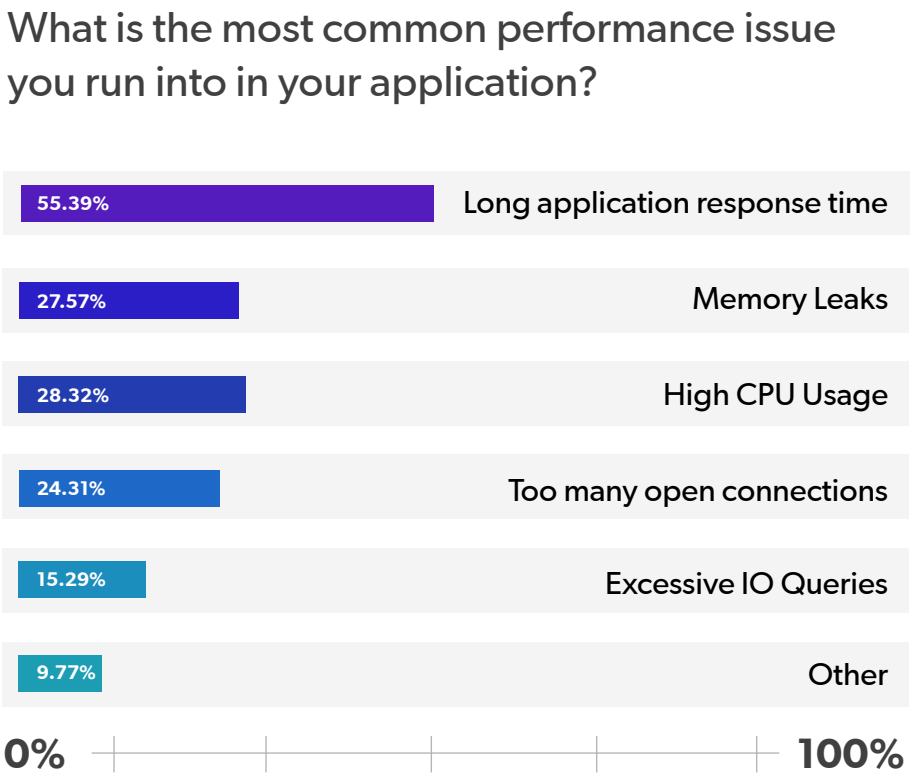
Common Performance Issues

One of our favorite questions to ask developers is about where they're experiencing performance issues. Understanding this gives us a grasp of what development teams experience when it comes to performance issues.

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

- Long application response time
- Memory leak
- 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.



55% of respondents reported the most common performance issue was long application response time. While this isn't a surprise, it does mark a continuing trend for developers. The trend also aligns with microservices adoption, which is notable for developers considering a transition to the architecture.

The next highest was memory leaks at 28% of respondents. High CPU usage marked the third most common issue at 28% of respondents, with excessive open connections and IO queries coming in at 24% and 15% respectively.

10% of respondents chose other, but there were no statistically notable answers from those who wrote in.

Verifying Application Performance Requirements

There are many ways in which Java development teams verify application performance requirements. But, for the purpose of this survey, we limited it to five different responses:

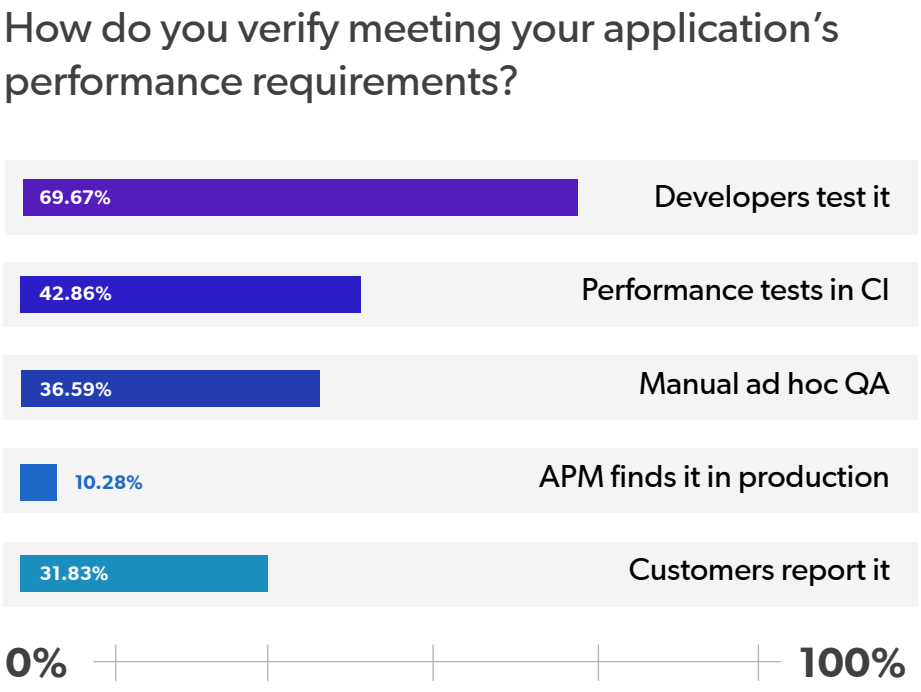
- Developers test it
- Performance tests in CI
- Manual ad hoc QA
- APM finds it in production
- Customers report it

Responses for this question went well over one per person, showing us that performance requirements are commonly addressed throughout the application lifecycle.

70% of respondents reported that developers tested for performance, while 43% reported testing for performance during continuous integration.

37% reported that they verify performance via manual ad hoc quality assurance, and 32% reported that they use customer reports to inform their performance success.

Lastly, 10% of respondents reported that they verify performance with application performance management technology.



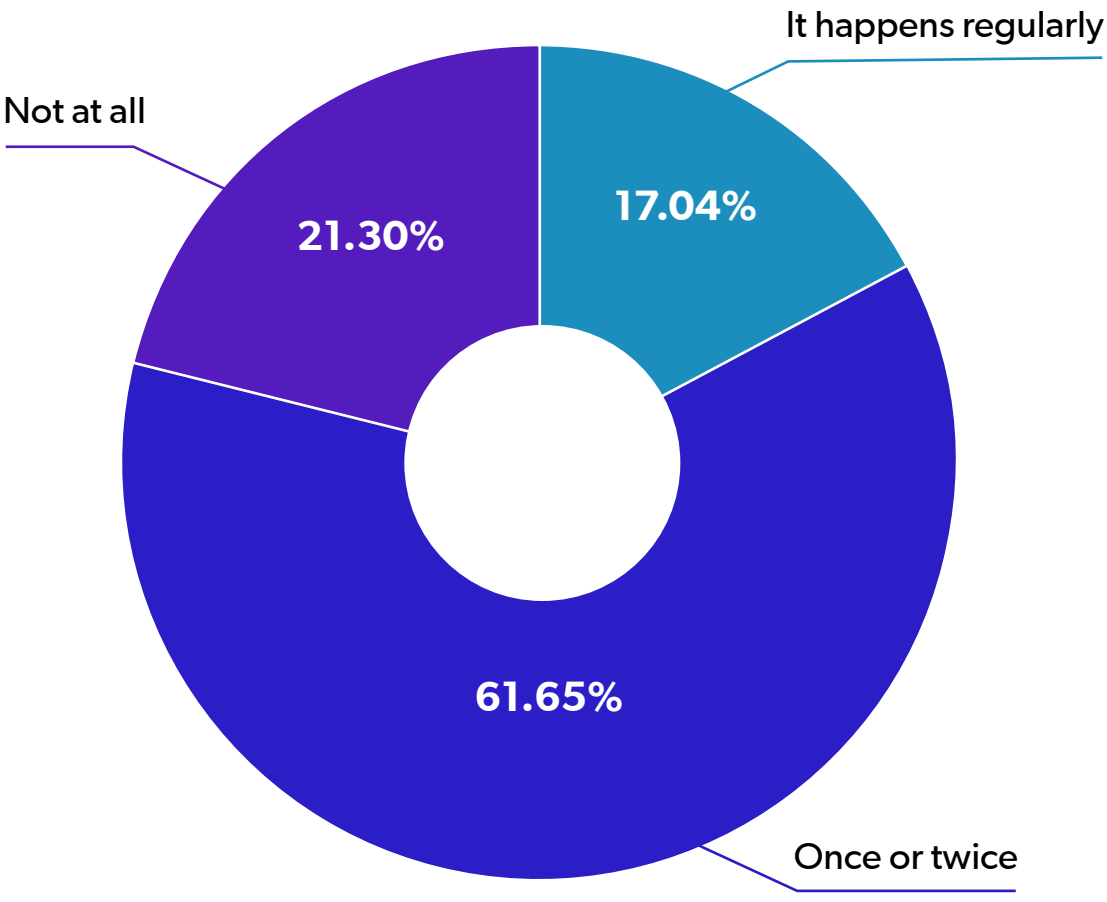
Performance Issues Reaching Production

Despite the increasing responsibilities for developers to test performance during development, performance issues still reach production. Our survey found that over 78% of teams had issues that reached production in the last 12 months.

While it wasn't happening regularly for most respondents, (only 17% reported experiencing issues regularly) 62% of developers reported performance issues experienced during production.

The lucky minority who didn't experience issues during production comprised 21% of overall respondents.

Have you had performance problems reaching production in the last 12 months?



Java Developer Productivity Report:
Impact of Reload and Redeploy Times

It wouldn't be a JRebel survey without asking about how the build process affects developers. After all, our flagship product, JRebel, is built on helping developers to skip rebuilds and redeploys.

But learning how the build process affects developers can give us valuable insights into the reasons why people use and don't use JRebel.

Of the 231 who answered the question, we found a common theme — slow build processes negatively impact the developer and the code.

NOTABLE QUOTES

Here are a few of our favorite quotes from the survey:

> "By automating the build process, it lets me focus more on the development aspect and not spend time in the build/deploy flow."

> "It becomes boilerplate at times that for even a small change in the code, it needs to be re-built and deployed, which is repetitive, time-consuming, and cumbersome."

> "Build and deploy takes three to four minutes total, it sucks."

> "If the build process is heavy/complex/requires rebuilding the entire app/restarting the server after each change then the quality of the work may suffer (my ability stays the same, but time consumption increases drastically)."

> "Quick build = faster deployment, less context switching = more thought-out code = better overall quality."

Average Redeploy Times

Redeploys take too long, regardless of how long they take. But for some developers, redeploys can go from annoyance to development roadblock.

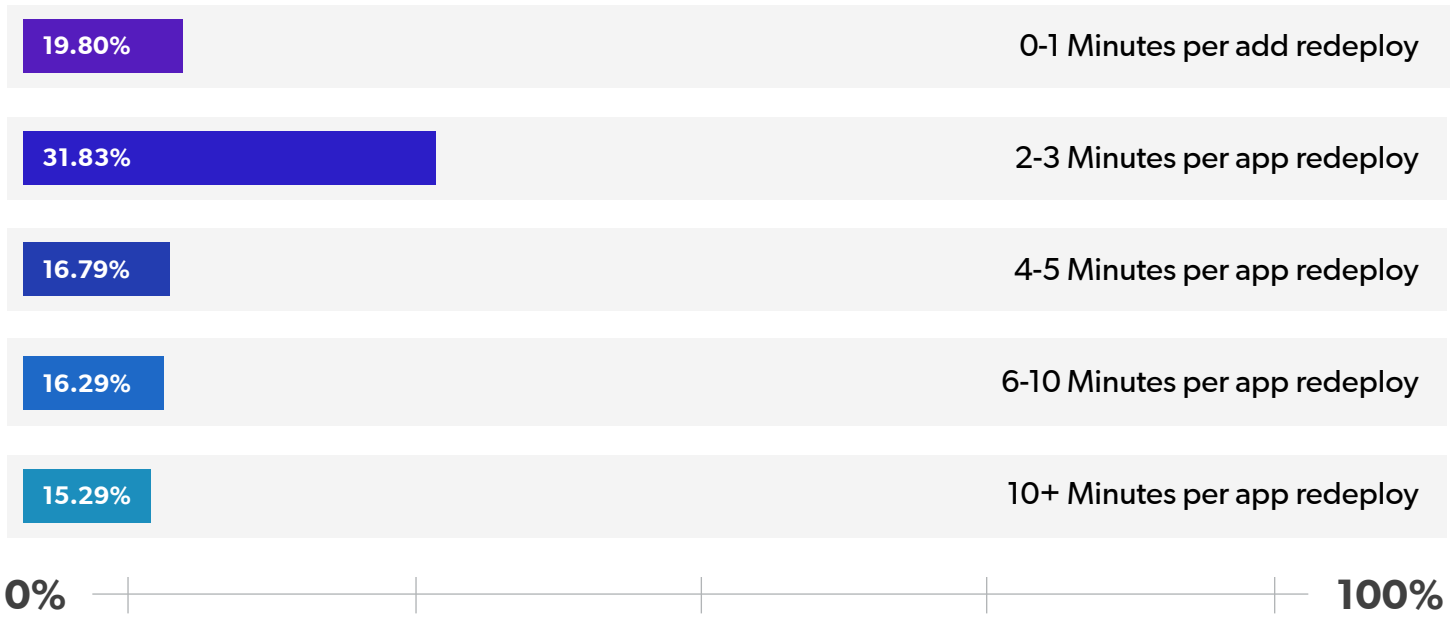
In this survey question, we asked developers to place their redeployment time onto a scale of zero minutes to ten minutes plus.

For 32% of respondents, the most common redeploy time was between two to three minutes. The next most common redeploy time, at 20% of respondents, was up to one minute per redeploy.

The surprising thing for us was that, despite the widespread adoption of microservices, over 48% of developers experienced reload times of over four minutes.

Perhaps even more surprising was that over 15% of our respondents reported reload times of over ten minutes!

After a code change in your application, how long does it take to compile, package, and redeploy your application to a visibly-changed state of runtime? (in minutes)



Java Developer Productivity Report: Microservices

When is the last time you went a day without hearing the word microservices? This popular application architecture is the current architecture of choice for most Java application developers.

But there's a reason why everybody is talking about (and using) microservices --- microservices offer increased flexibility for testing and development. Completing the transition from the monolith to microservices can have a profound, positive impact on applications and empower development teams to achieve new innovations faster.

But, as we'll explore in the next few pages, there is no shortage of issues facing developers working with microservices-based Java applications and those issues may become more significant as applications mature.

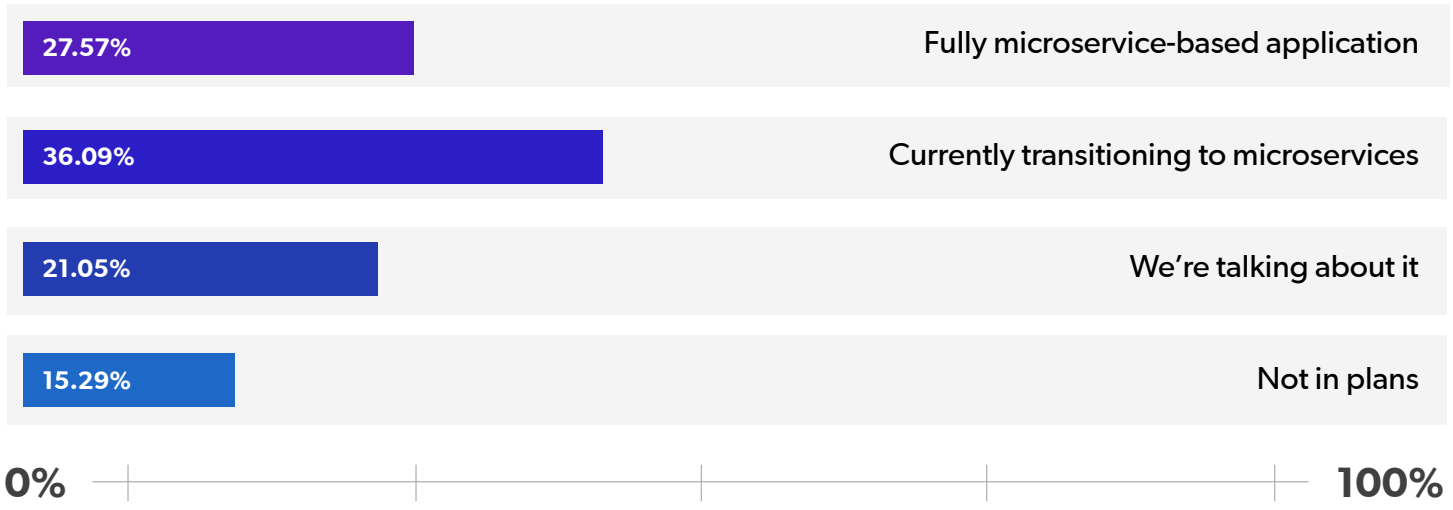
Microservices Adoption

In our survey, we found that over 63% of our survey respondents were either working in, or actively transitioning to, microservices. An additional 21% were actively talking about adopting microservices.

Combined, that means 85% of survey respondents are either in microservices, or considering adoption, with only 15% not considering microservices for their application.

But maybe the most surprising statistic of our survey is that 28% of respondents were working with fully microservice-based applications.

What is your status for microservices adoption?



Challenges With Microservices

Microservices offer solutions to many problems present in monolithic applications – but these new solutions often present new challenges. Microservices performance, for instance, can be difficult to improve or even maintain as microservices applications grow more complex. In fact, our survey found that performance issues, both inter-service and overall, were the number one reported issue for developers.

Combined, performance issues impacted developers the most — with over 62% of respondents reporting issues in either troubleshooting inter-service performance (29%) or performance of the overall distributed system (33%).

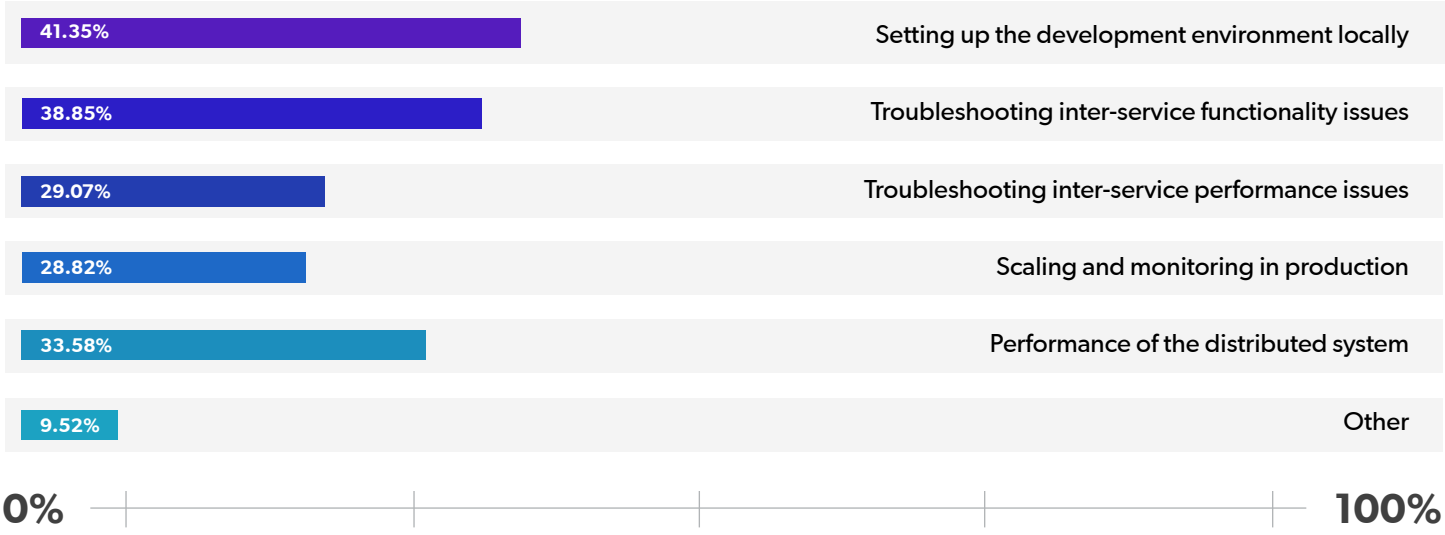
The next biggest issue was setting up the development environment locally, which garnered a 41% share of respondents. This isn't a big surprise, as local deployment of distributed applications can be an intricate, difficult, and time intensive process.

Next, 38.85% of respondents found troubleshooting inter-service functionality issues to be their biggest issue.

Another challenge for developers was in scaling and monitoring during production. 29% of respondents listed these issues as their biggest challenge.

While other respondents weighed in with individual complaints ranging from integration testing to executive buy-in, there were no statistically-significant complaints.

What's the biggest challenge while developing microservices?



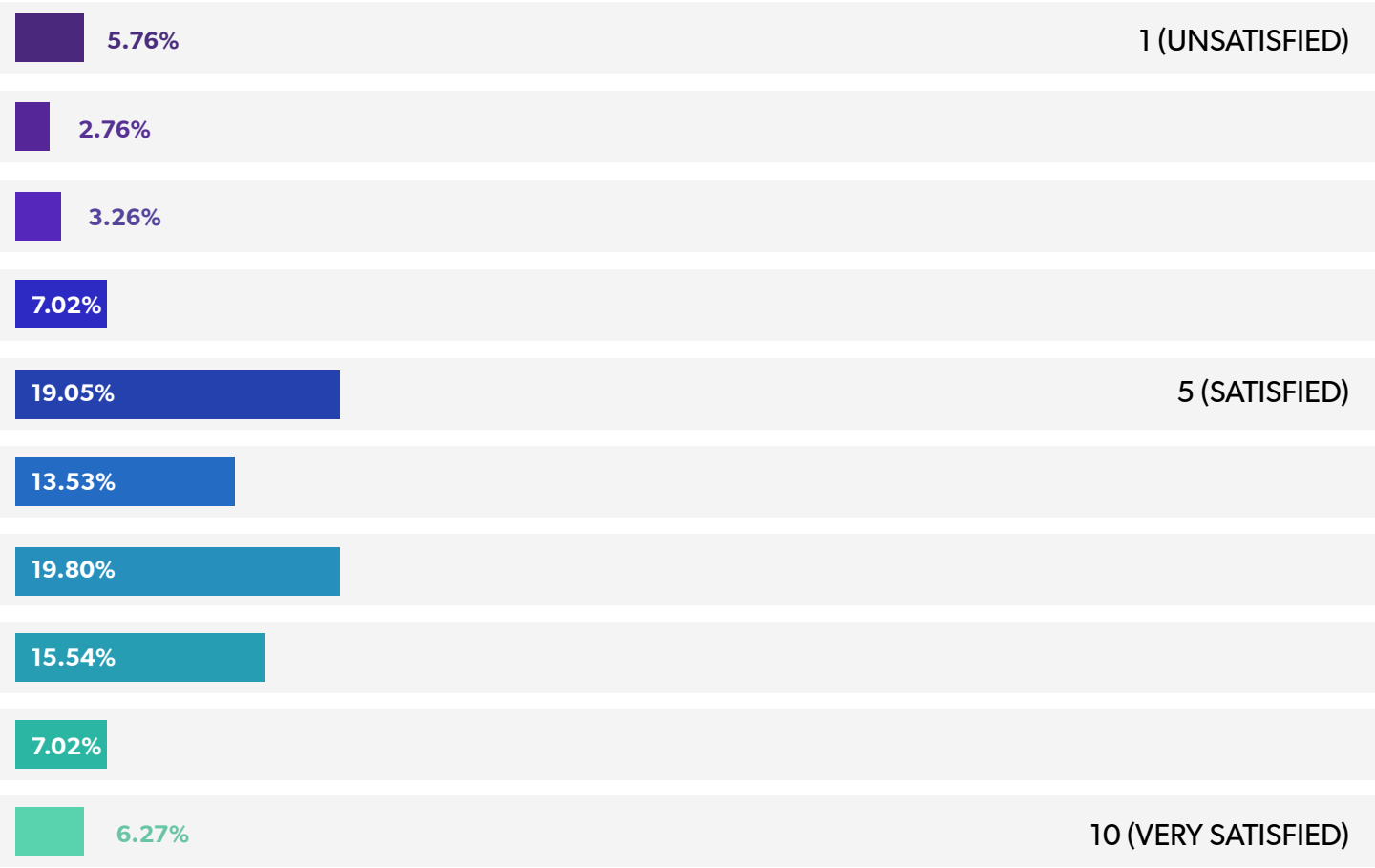
Microservices Visibility

For microservices developers, visibility can be a big issue. And, as we saw in the previous section, not knowing how code in one service can affect code in another service can prove a nuisance for developers.

In this question, we asked respondents to rate their satisfaction with their insight into how their microservices work together as a system.

Not surprisingly, most respondents weren't completely satisfied with their level of insight into how their microservices work together as a system. On average, respondents reported satisfaction at six out of ten stars, with only 6% reporting that they were very satisfied with their visibility into microservices interaction.

How satisfied are you with the level of insight you have into how the microservices work together as a system?



Troubleshooting Microservices

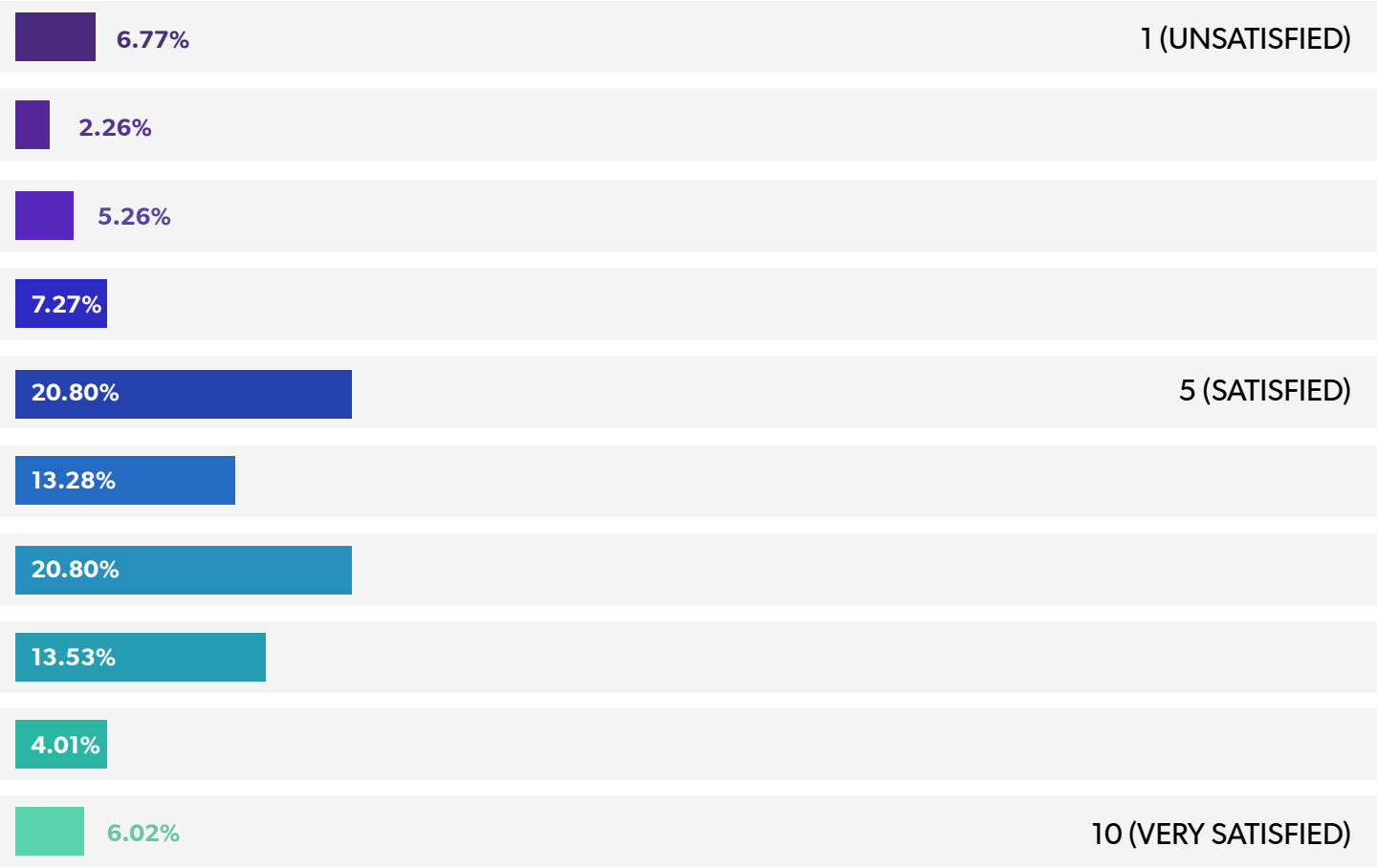
Our next question focused on how developers felt about the tools available to troubleshoot their microservices-based applications.

Perhaps showcasing the lack of polished microservices troubleshooting tools, only 6% of developers reported being very satisfied with available solutions — with the average developer rating their overall satisfaction of six out of ten stars.

In fact, more developers felt completely unsatisfied than very satisfied, with 7% reporting no satisfaction with available troubleshooting methods.

As available solutions (like XRebel, our microservices-ready performance tool for Java development) become more integrated into development teams and more suited to microservices, we expect satisfaction to increase.

How satisfied are you with the means to troubleshoot your microservices-based application?



Java Developer Productivity Report: Summary

In this report, we explored the tools and technologies used by Java professionals, how they approach performance in application development, the effect of redeploys on developer efficiency, and the effect that microservices are having on development teams.

TOOLS AND TECHNOLOGIES

The effects of microservices were everywhere in our tools and technologies survey results, with many developers leaning toward microservice favorites like Spring Boot, Tomcat, and Docker.

Language choice seems to be stuck in the past, with most developers using Java 8, but given the lack of (debatably) significant changes to the language, it's understandable.

APPLICATION PERFORMANCE

Our survey showed a continuing trend — developers are expected to play a bigger role in application performance. In fact, over half of our respondents reported having non-functional performance goals for their current project.

This also means that developers are being asked to test application performance during development, often without full insight into how their code is impacting the entirety of the application.

DEVELOPER PRODUCTIVITY

For Java developers, one of the biggest issues facing productivity (aside from lack of caffeination) is redeploys.

And, as our report showed, microservices aren't necessarily making these issues go away. In fact, almost half of our survey respondents reported redeploy times of over four minutes per redeploy.

MICROSERVICES

The effects of microservices adoption Java applications and developers has been monumental — with changes to everything from team composition to emerging technical problems and technological solutions.

And, with the wholesale adoption of microservices as the architecture of choice for most Java applications, these issues will continue to develop in complexity as new technologies interact with one another.

As these problems grow more complex for microservices developers, tools and technologies that solve them will become more necessary.

Improving Microservices Development

Our two Java development solutions, JRebel and XRebel, can combine to help Java developers seamlessly find, fix, and instantly check code changes in microservices-based applications.

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

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