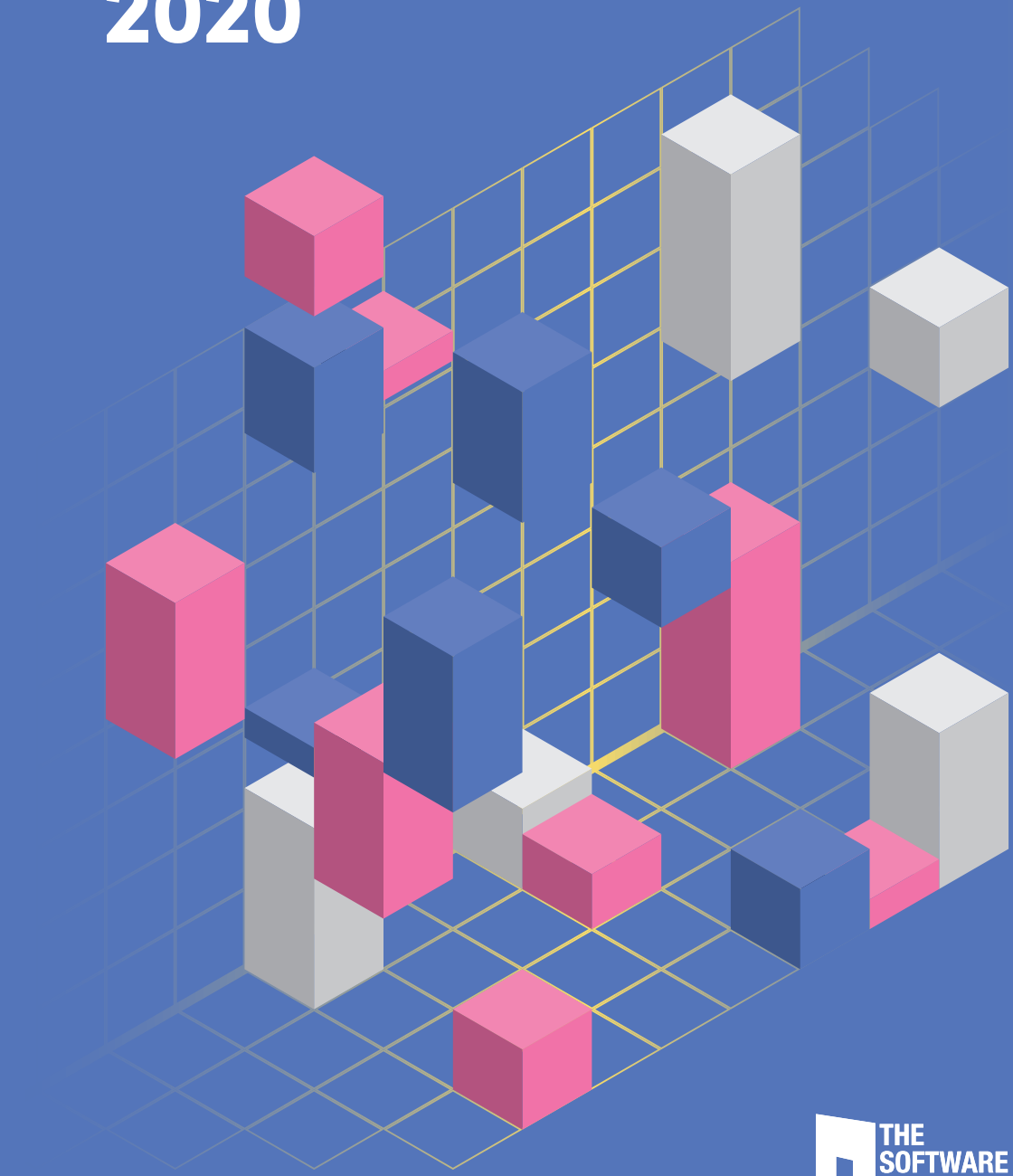
THE NEW STANDARD FOR BUILDING SCALABLE SOFTWARE?

State of Microservices 2020



THE SOFTWARE HOUSE

Authors

Patryk Mamczur
Editor in Chief

Tomasz Czechowski
Mateusz Mól
Design of online version

Mariusz Nowak
Design of print version

Experts

Marek Gajda
CTO of The Software House

Adam Polak
Head of Node.js Team
at The Software House

Yan Cui
Host of Real World
Serverless Podcast

Peter Cooper
The Founder of Cooperpress

Ewelina Wilkosz
IT Consultant at Praqma

Thomas Boltze
CTO of Asto Digital

Sarup Banskota
Head of Growth at ZEIT

Luca Mezzalira
VP of Architecture at DAZN

Richard Rodger
CEO of Voxgig

Published in 2020

Ver. 1.0

_Table of contents

How many IT experts filled in the survey?

Total answers: 669

252
Western Europe

163
Eastern Europe

96
North America

49
South and East Asia

23
Central and South America

24
Middle East

40
Other

22
Australia and New Zealand

_develo-pers

669 microservice experts
from around the world

We wanted to find out how developers around the globe really build their microservices.

Microservice architecture – an architectural style where your application is based on a collection of fine-grained, interconnected services – is definitely a hot thing right now. But what are the pros and cons of this architecture? What's the future of this trend? Should we all jump on the hype train? Well, to find out, we decided to create this report.

The goal of the State of Microservices 2020 research project was as straightforward as it was ambitious. We wanted to find out how developers around the globe really build their microservices. And if you're wondering if we succeeded – just take a look at the map on the previous pages.

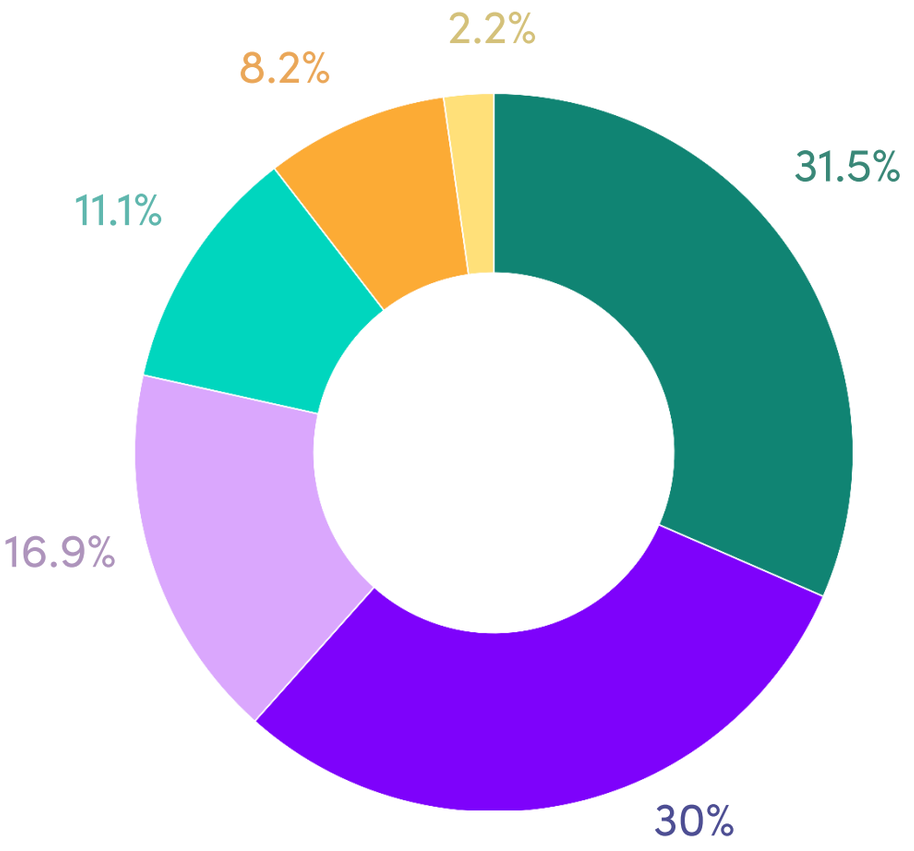
Firstly, over 650 software developers decided to take part in our survey. Secondly, more than a half of these talented folks were CTOs, Lead Developers or Senior Developers, showing us all how experienced the microservice community is. And, last but not least, the respondents really came from all around the world, making the report even more useful and universal than we had wished in the beginning.

However, numbers are just numbers. And this is why we decided to let the voice of the microservice gurus be heard by inviting the awesome people of DAZN, Coop-erpress, ZEIT and more to comment on the results. You'll find their expert opinions on the following pages.

So, without further ado, I present you the complete State of Microservices 2020 report – the most up-to-date source of information on the state of microservice architecture.

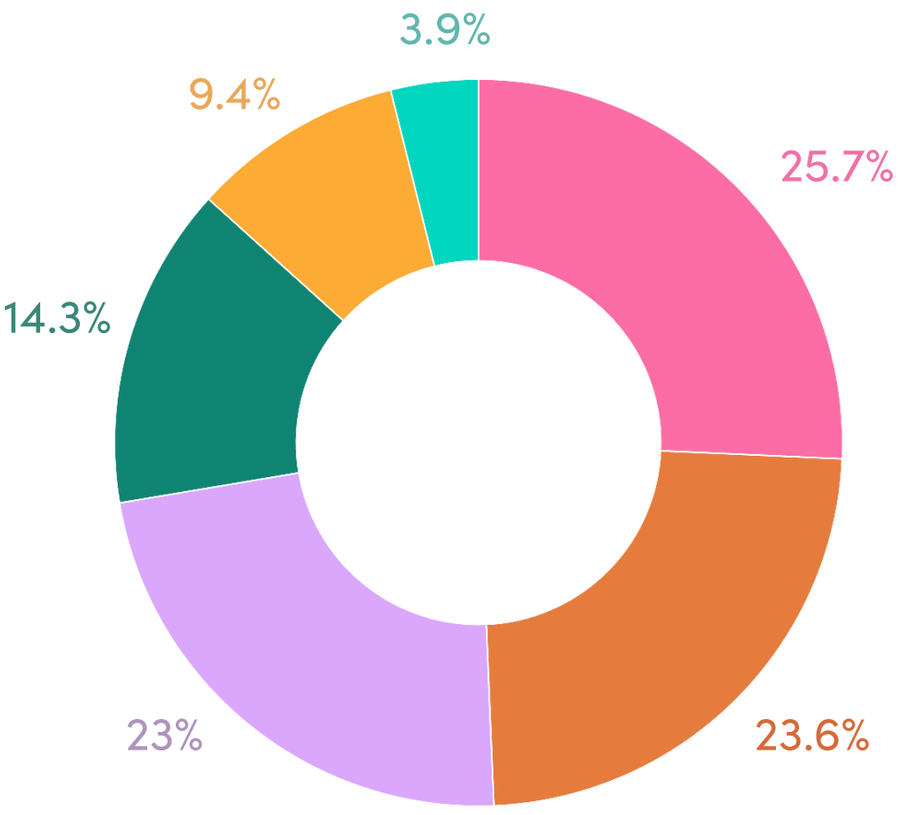
Patryk Mamczur
Report's Editor in Chief

How would you describe your seniority?



- Lead developer, Head of technology (211)
- Chief technology officer (74)
- Senior developer (201)
- Junior developer (55)
- Mid-level developer (113)
- Others (15)

How big is the company you are working in?



- 11-50 employees (172)
- 2-10 employees (96)
- 51-200 employees (158)
- 201-500 employees (63)
- 501+ employees (154)
- I'm the one-person company (26)

02

_maturity

Great architecture for
solving scalability issues

The two most important topics when it comes to microservices are scalability and performance.

I must admit that when we were designing the State of Microservices 2020 survey I had some presumptions. And, when the results arrived, most of my assumptions were confirmed – however, both the positive and the negative ones.

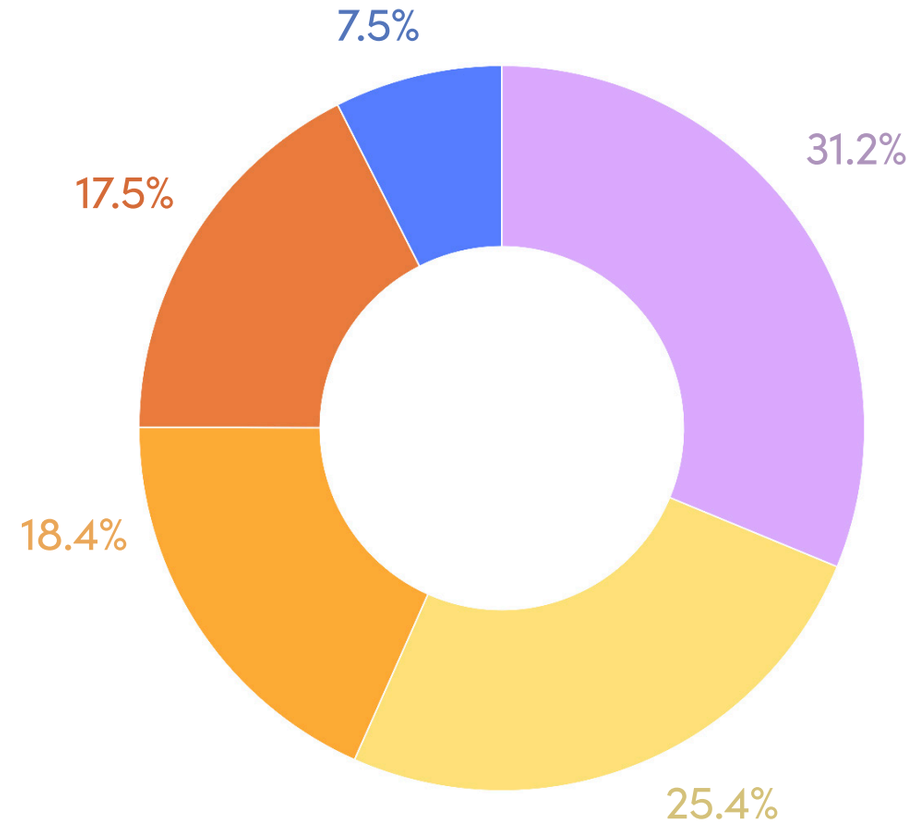
In my opinion, the two most important topics when it comes to microservices are: improving scalability and improving performance. All in all, that's why the idea of microservice architecture was dreamt up in the first place, wasn't it? Fortunately, it seems that software developers around the world are truly happy about building microservices when it comes to solving scalability issues (average rating: 4.3 out of 5) and performance issues (average rating: 3.9). It's even more evident when you look at the responses of CTOs – who should hold the most informed view when it comes to such issues – and who rated the scalability potential to 4.4 and the performance potential to 4.3.

On the other hand, it seems that maintenance and debugging are a bit of a problem for many microservice developers. Speaking from my professional experience at The Software House, I must confirm that these findings are true. For the last 2 or 3 years, more and more clients have approached us and asked us for help because their in-house development teams lost control over microservice-based projects. Usually, after some hussle, everything can be straightened out but my protip for you is: hire an experienced DevOps engineer at the very beginning of your microservice project. These guys can do real magic when it comes to future-oriented thinking and planning out your whole system's architecture.

All in all, microservice architecture is not a cure for all of your software problems. If you think that you can run a short-term microservice project without previous experience, you're probably wrong. However, if your business is based on scalability and you take a minute to plan out the software architecture in the very beginning of a project – you'll definitely see the benefits of microservices.

Marek Gajda
CTO of The Software House

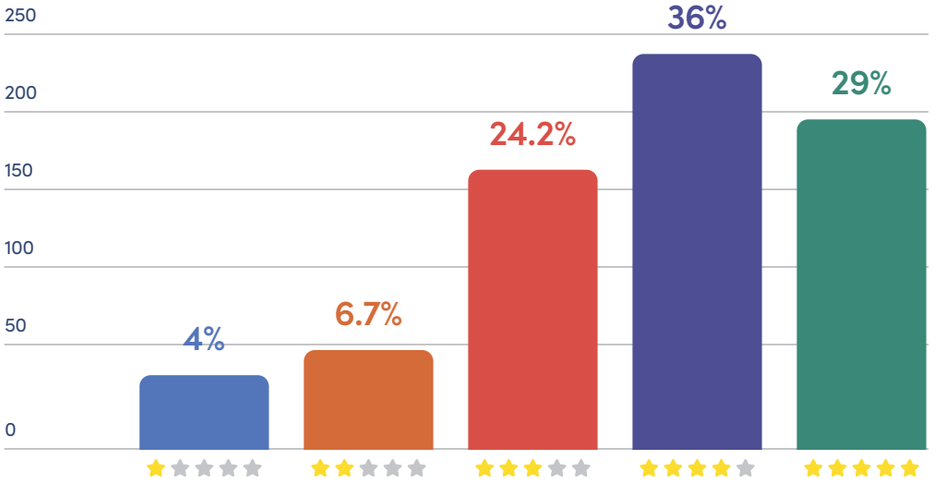
For how long have you been using microservices?



Rate in scale 1–5 (where 1 means the worst, and 5 means the best possible experience) how you enjoy working with microservices when it comes to...

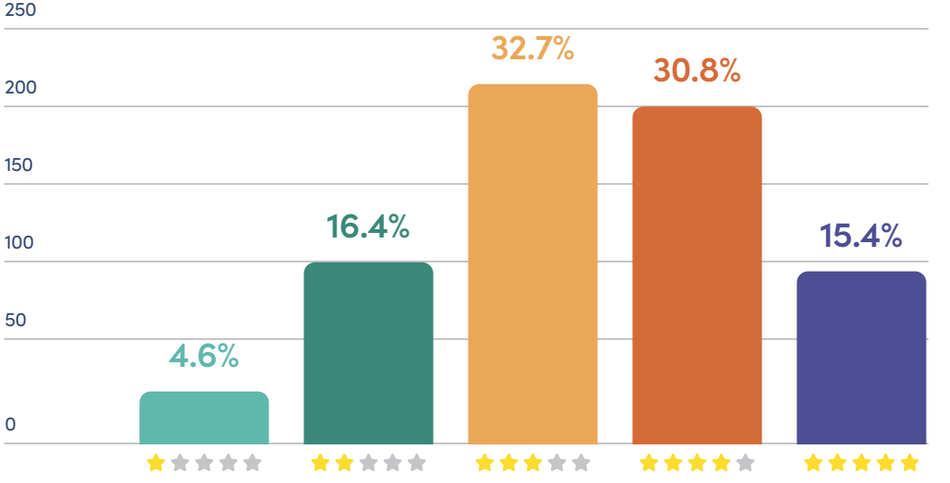
Setting up new project

Avg. 3.8



Maintenance and debugging

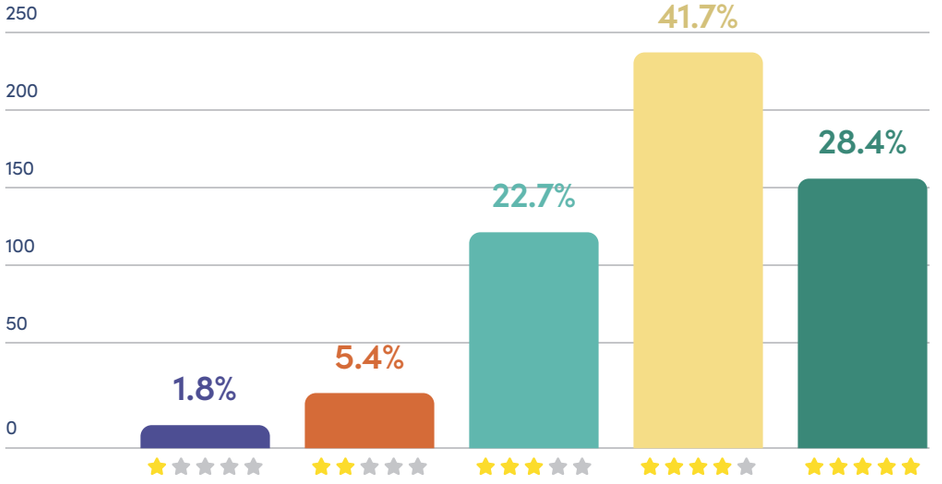
Avg. 3.4



Rate in scale 1–5 (where 1 means the worst, and 5 means the best possible experience) how you enjoy working with microservices when it comes to...

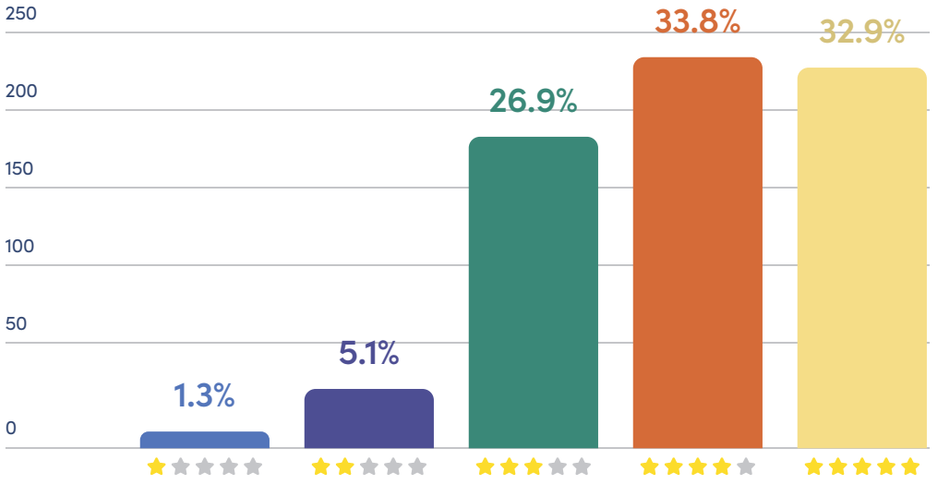
Efficiency of work

Avg. 3.9



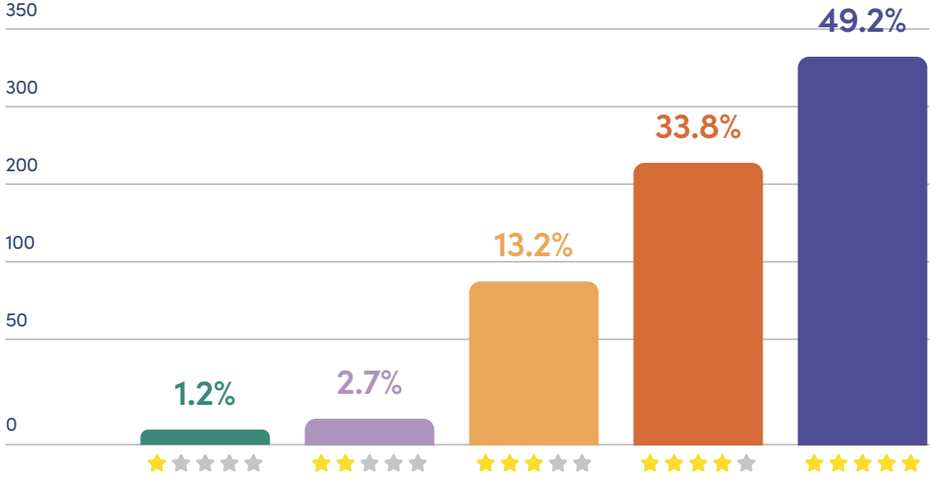
Solving performance issues

Avg. 3.9



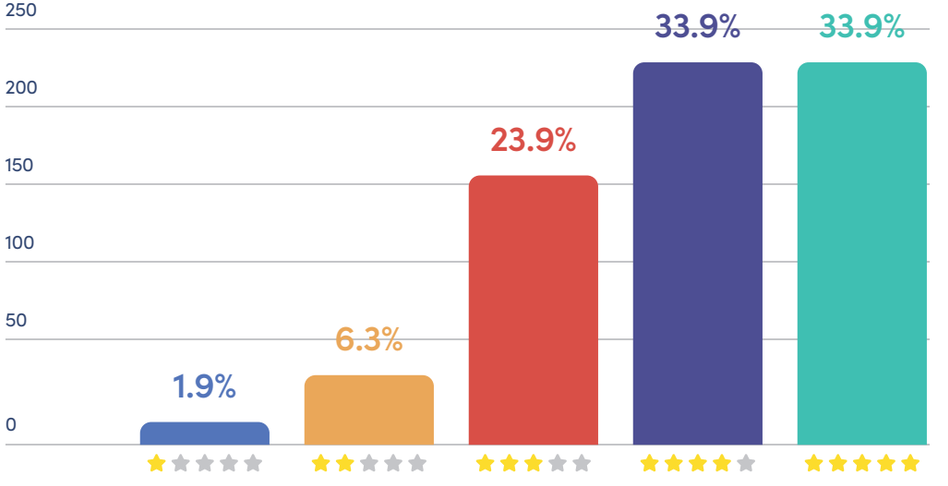
Solving scalability issues

Avg. 4.3



Teamwork

Avg. 3.9



03

_program-ming langu-ages

JavaScript and TypeScript rule all

Microservices and JavaScript/TypeScript go very well together.

To be honest, when I first saw the results of this part of the survey, I was pretty surprised. I knew that JavaScript and TypeScript were getting more and popular – but was it really possible that those were the main programming languages for almost ⅔ of microservice developers? Well, it certainly seems so!

For a pretty long time, microservice architecture has been associated with huge, enterprise solutions. And those were usually built using Java or .Net. However, it seems that things are changing pretty rapidly. Firstly, more and more people are talking about Universal JavaScript and how this language (both with TypeScript) is gaining popularity. Secondly, developers start building microservices not only for enterprise-grade platforms but also for medium-size software projects. Thirdly, the "microservices plus serverless" model is also on the rise and we all know that JavaScript and TypeScript go pretty well with serverless.

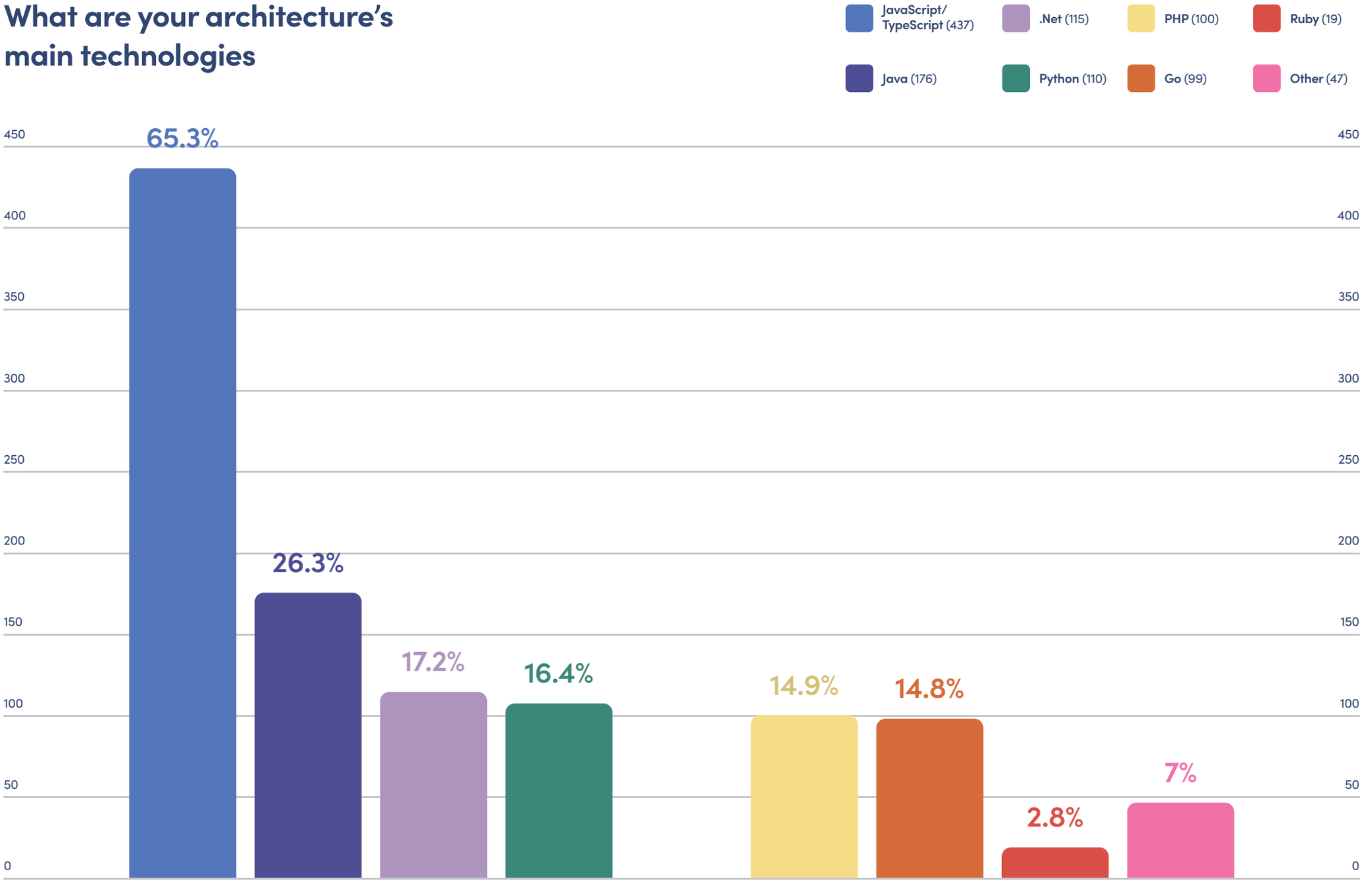
The results of our State of Microservices 2020 survey confirm all of these trends. 437 people (65%) named JavaScript/TypeScript one of their architecture's main technologies. And 171 of them (26%) chose JS/TS as the ONLY programming language for their microservices.

Whether you like this trend or not, one must say that microservices and JavaScript/TypeScript go very well together. Before the era of workers, Node.js architecture was very prone to slowdowns, so Node.js developers simply had to learn how to work with multiple small services.

Now, it's a part of their developer's DNA – and it makes building and maintaining microservice architecture whole lotta easier.

Adam Polak
Head of Node.js Team at The Software House

What are your architecture's main technologies



04

_deploy-ment and serverless

Microservice developers choose AWS

I’m pleased to see that nearly half of the developers are already using serverless technologies.

When I look at the results of the survey, I see that the market of cloud providers and serverless is thriving – there are as many obvious findings as there are surprises.

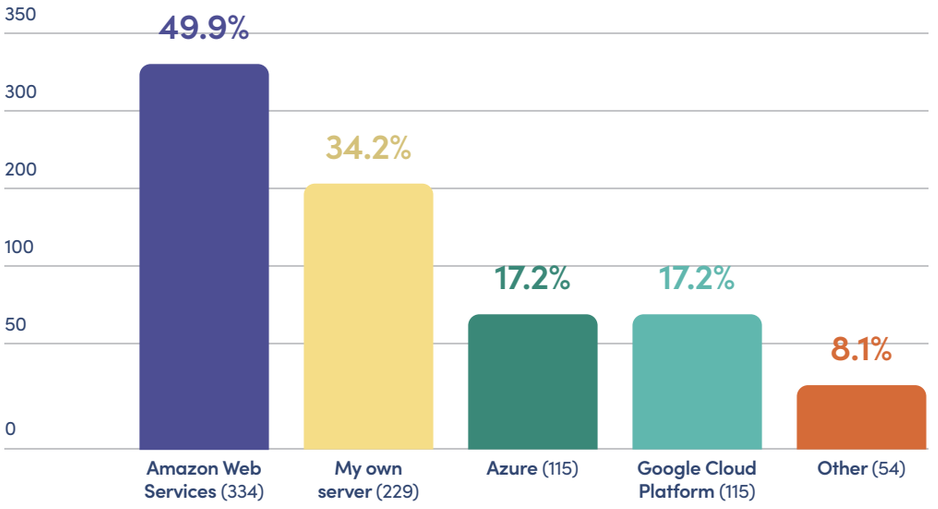
Quite unsurprisingly, AWS is the most popular (49%) deployment target, and most people are using Kubernetes (65%) for service discovery. What is surprising, however, is that 34% of respondents are running on-prem, which is as much as Azure (17%) and GCP (17%) combined! I guess that when you're in the cloud bubble, it's easy to forget that traditional DCs is still a $200B market annually and accounts for as much IT spending as all the cloud providers combined.

I'm pleased to see that nearly half the respondents said they're already using serverless technologies. Here, once again, AWS Lambda is the clear leader with 62% of the responses. I did, however, expect to see Azure functions (13%) faring better than Google Cloud Functions (14%) – given that GCP is still focused on their container services and has largely neglected Google Cloud Functions. Perhaps the numbers have been helped by Firebase, which has a strong user base and does have a good developer story with Firebase functions.

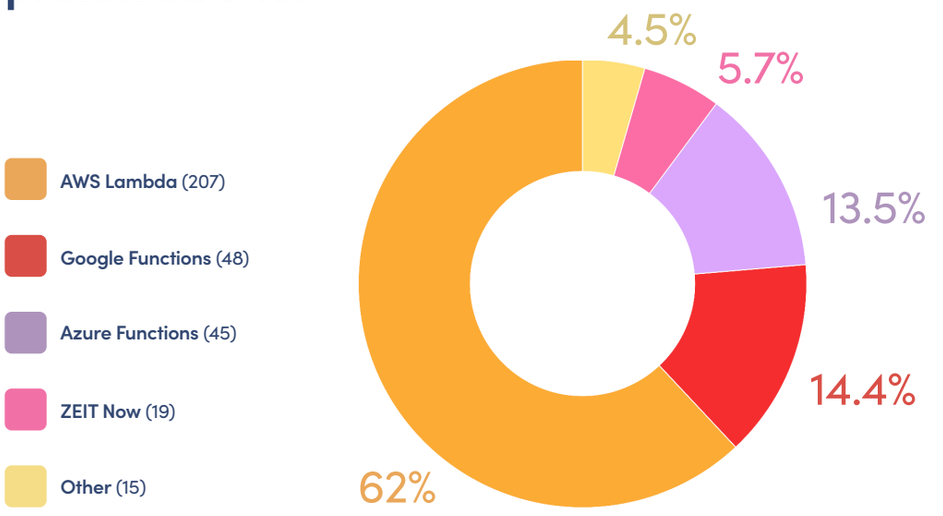
All in all, while we can definitely see that Amazon Web Services are leading when it comes to cloud and serverless, the situation is still far from a monopoly. With other providers reaching for their piece of cake, and with bare-metal servers grabbing a huge market share, one thing is certain – when you’re a microservice developer, there’s plenty of options for you to choose from.

Yan Cui
Host of Real World Serverless Podcast

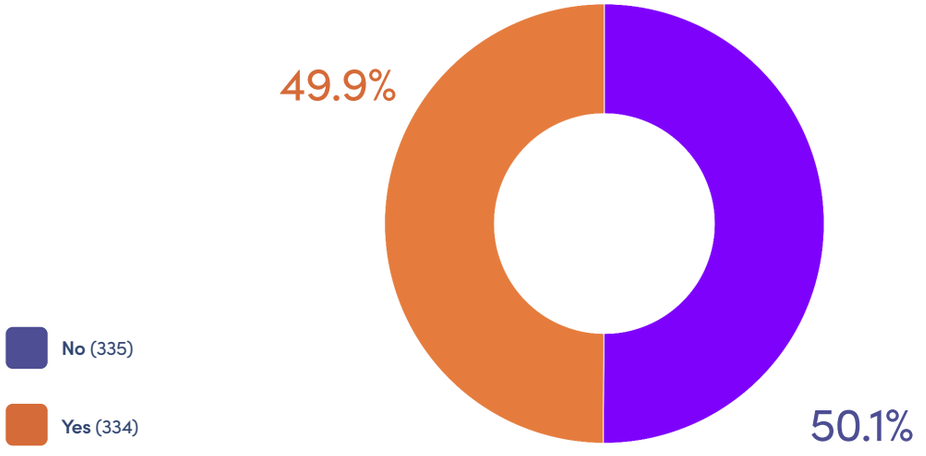
Where do you usually deploy your microservices to?



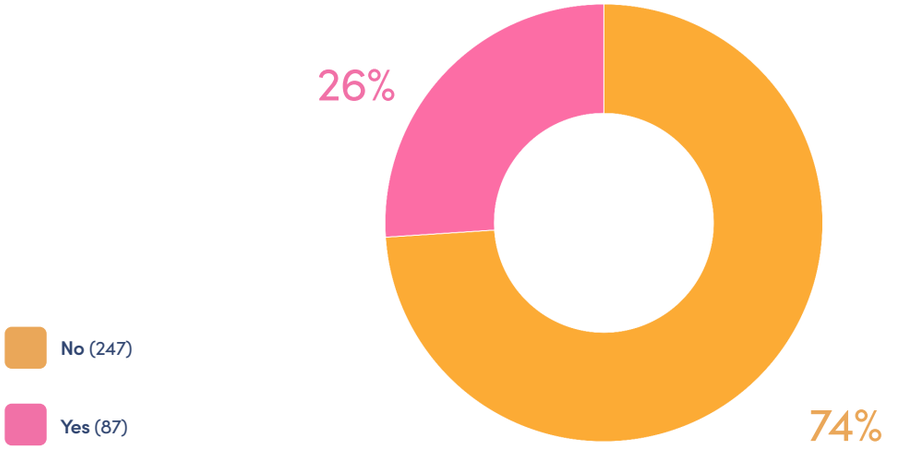
Which serverless solution is your preferred one?



Do you use serverless technology?



Do you use AWS Serverless Application Model?



05

_reposito-ries

Monorepo or multirepo?

I expect to see significant growth for monorepo use over the next couple of years.

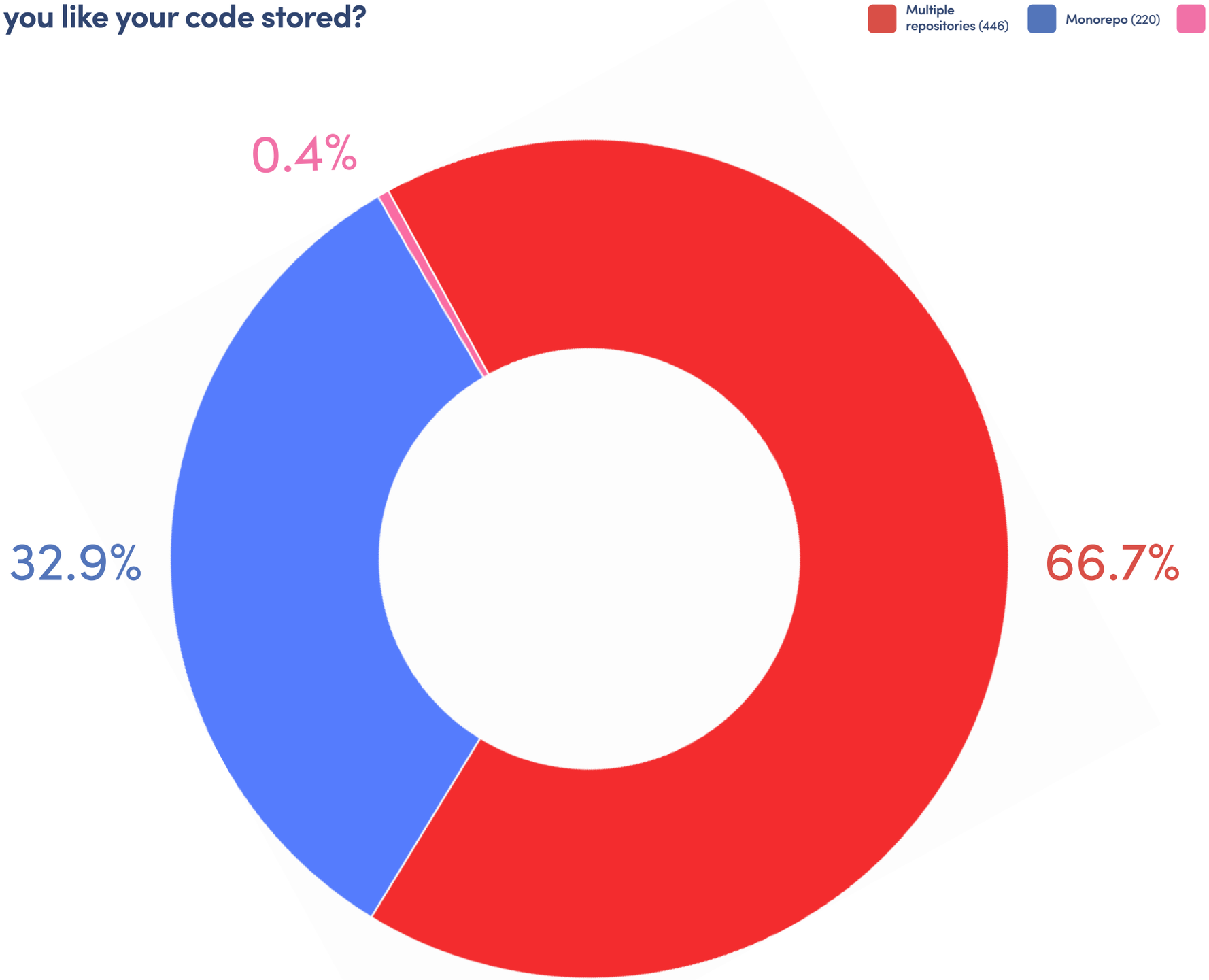
It's not a big surprise that the majority of respondents say that they prefer using multiple repositories for their projects – that's been the status quo for years now. It's more interesting, however, that as many as 32.9% said they DO prefer monorepos. And this number is sure to grow.

The monorepo approach to software development involves storing the files for numerous projects within a single repository (that's internally segregated in some way, usually through folder structure). One company well known for adopting this approach is Google. With the majority of their code residing within a single monolithic repository, they can take advantage of easier code reuse across the company and easier integration with their company-wide Continuous Integration systems. Other companies using monorepos, at least to some extent, are Microsoft, Facebook, Digital Ocean, Twitter, and Uber. On the other hand, the monorepo approach is still broadly considered cutting-edge or experimental in smaller companies and in single developer cases. To be honest, it's not that surprising, as the approach's main advantages are around teamwork and integration.

However, much like the almost universal growth of CI (after initially being more popular within larger companies and teams), I'd expect to see significant growth for monorepo use outside of its core user base over the next couple of years too. Especially, as more tools emerge that target smaller use cases.

Peter Cooper
The Founder of Cooperpress

How do you like your code stored?

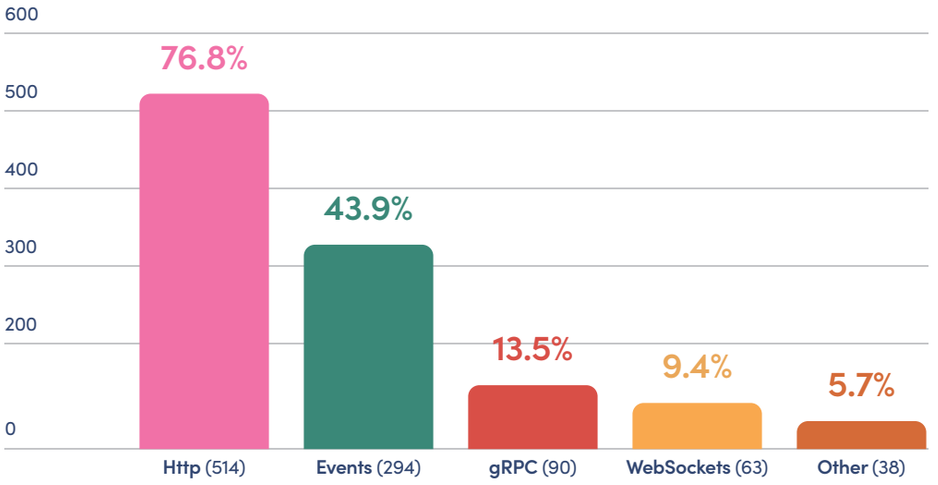


06

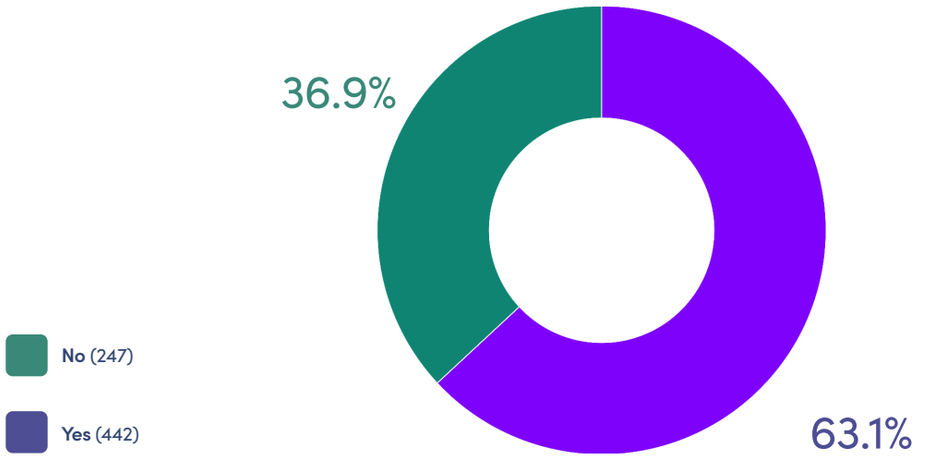
_varia

Communication, authorisation, message brokers

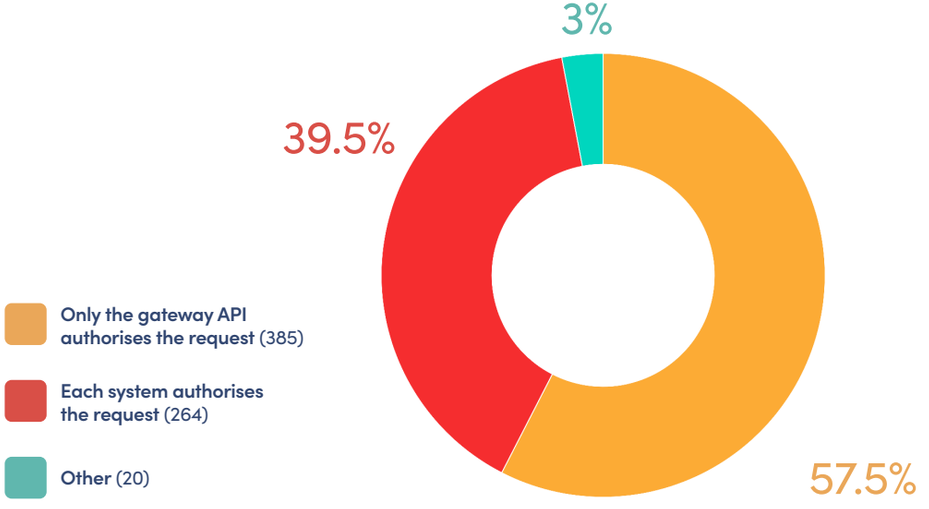
How do your microservices communicate with each other?



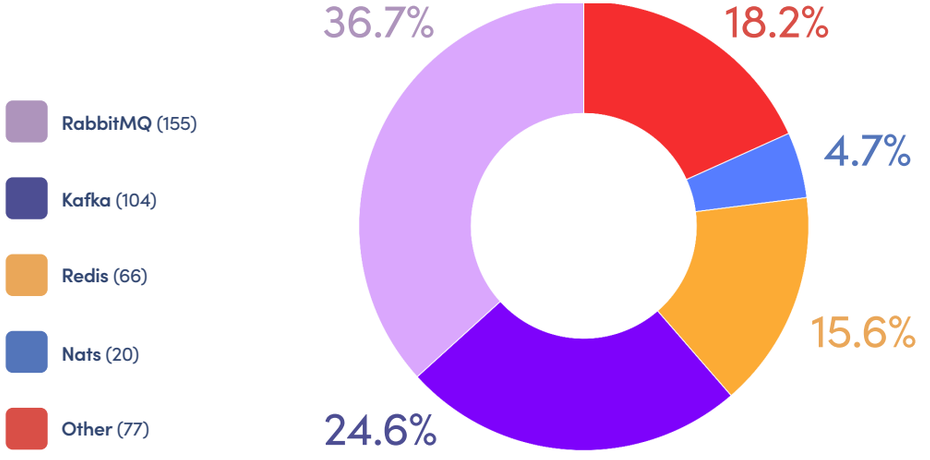
Do you use message brokers?



How do you take care of authorisation?



Which message-broker software do you prefer?



07

_continuous integration

Microservices + CI = <3

It’s fair to say that Continuous Integration is quickly becoming a standard – at least among the microservice developers.

It’s fantastic to see that almost 87% respondents use Continuous Integration. It’s fair to say that CI is quickly becoming a standard – at least among the developers who build microservices. However, I can’t stop wondering: what is the rest doing? 13% is a pretty significant number! So, the need to educate and help developers to understand the topic is still there.

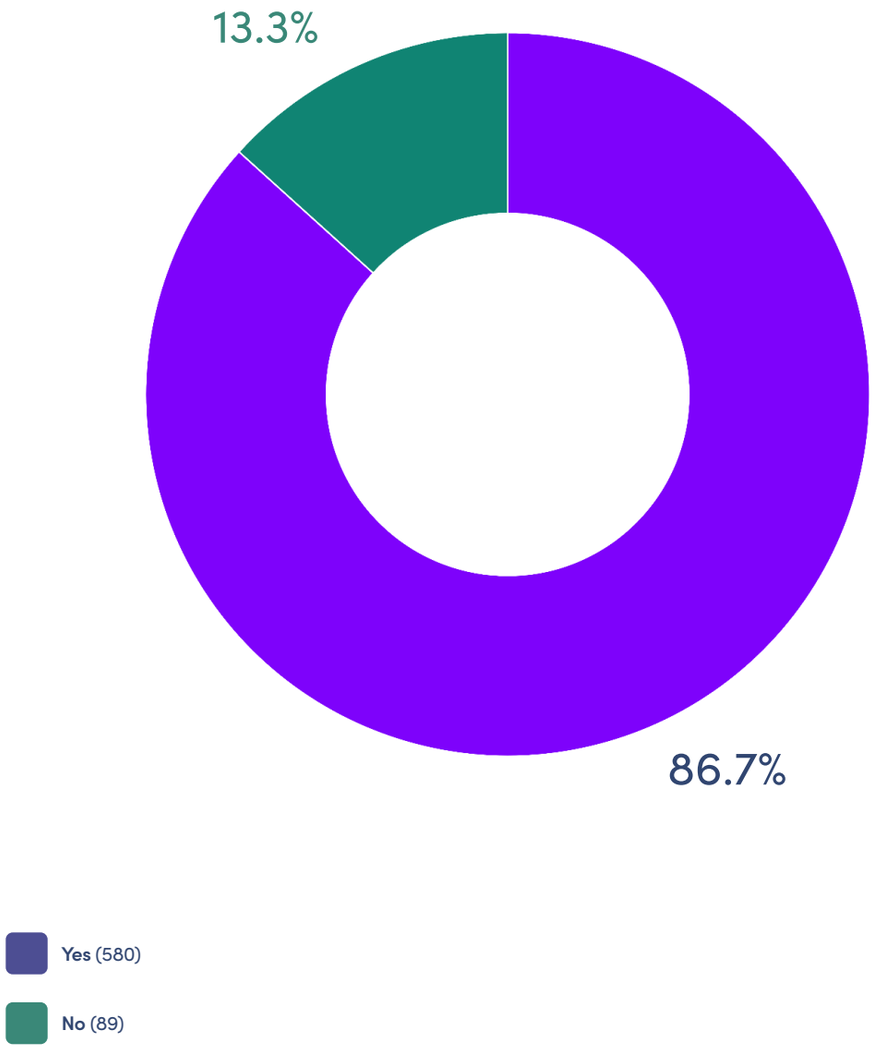
What makes me genuinely happy, are the results regarding the most popular CI solutions. Frankly, I did expect Jenkins to have a bit bigger share, but the fact is that the industry is changing and it is great to see that there is a diversity in the area. Especially, as many of these CI solutions are available “out-of-the box”, provided to you alongside a repository or a cloud solution. In practice, it means that running pipelines is now easier than ever before – and, judging by the numbers, it is working really well.

Having so many options to choose from might also impact the way the CI pipelines are built. Nowadays, it makes a lot of sense to put an effort into writing a pipeline in a smart way, so it is easy to migrate to another solution if need be. Of course, that requires a bit more attention at the beginning of the process, but it gives you bigger freedom and might be very rewarding in the future.

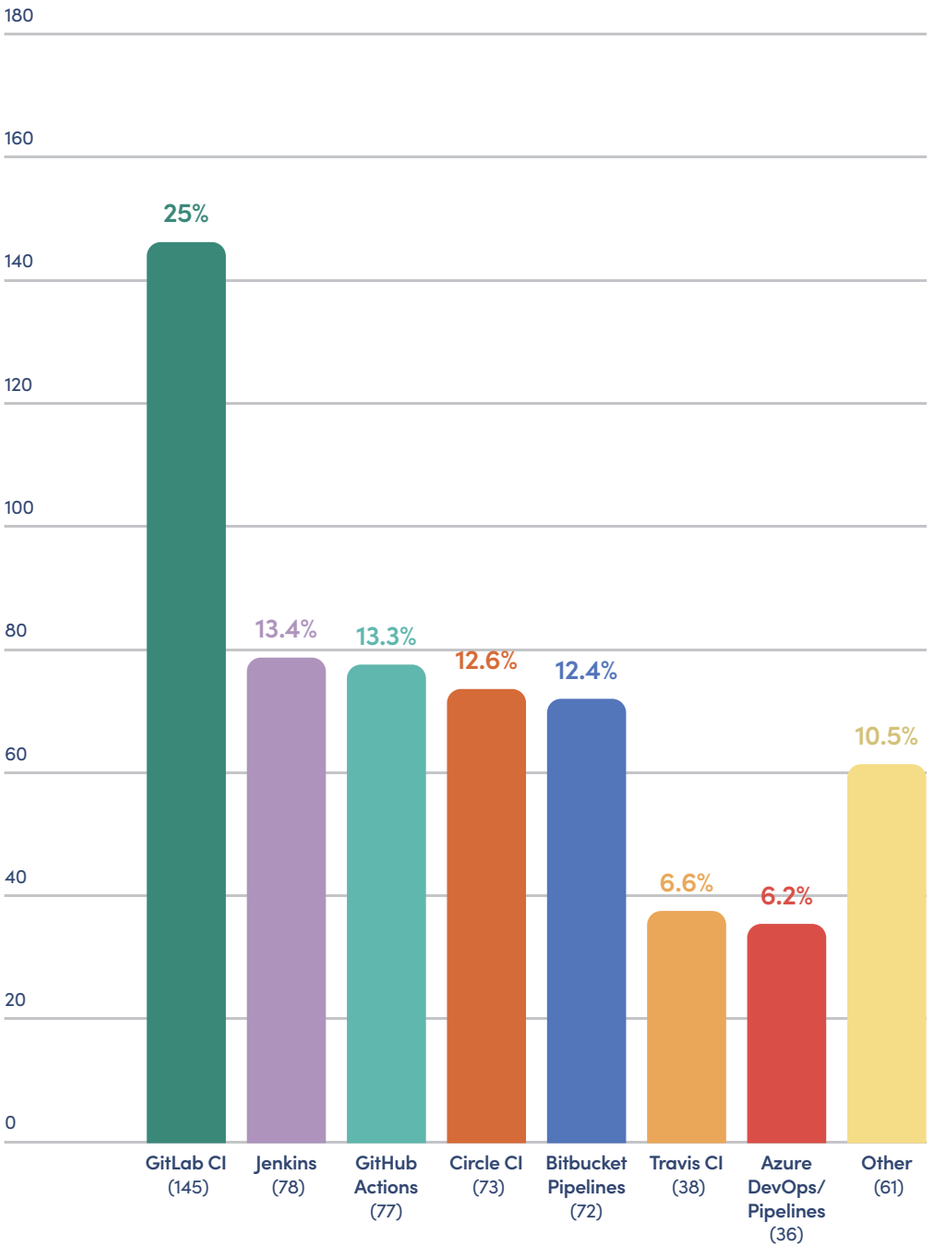
CI/CD is an important aspect of the software development process and it can bring many benefits to those who use it. Having a variety of well-working solutions and good practices to choose from is a luxurious situation for all of us.

Ewelina Wilkosz
IT Consultant at Praqma

Do you use Continuous Integration?



Which CI solution do you prefer?



08

_debugging

Are logs enough for your project?

Development teams struggle with predicting the consequences of going all-in with microservices.

For me, as the Chief Technology Officer, the survey results regarding debugging solutions were particularly interesting. The thing that immediately captured my attention was that the most popular debugging solution, with as much as 86% of developers choosing this answer, were... logs.

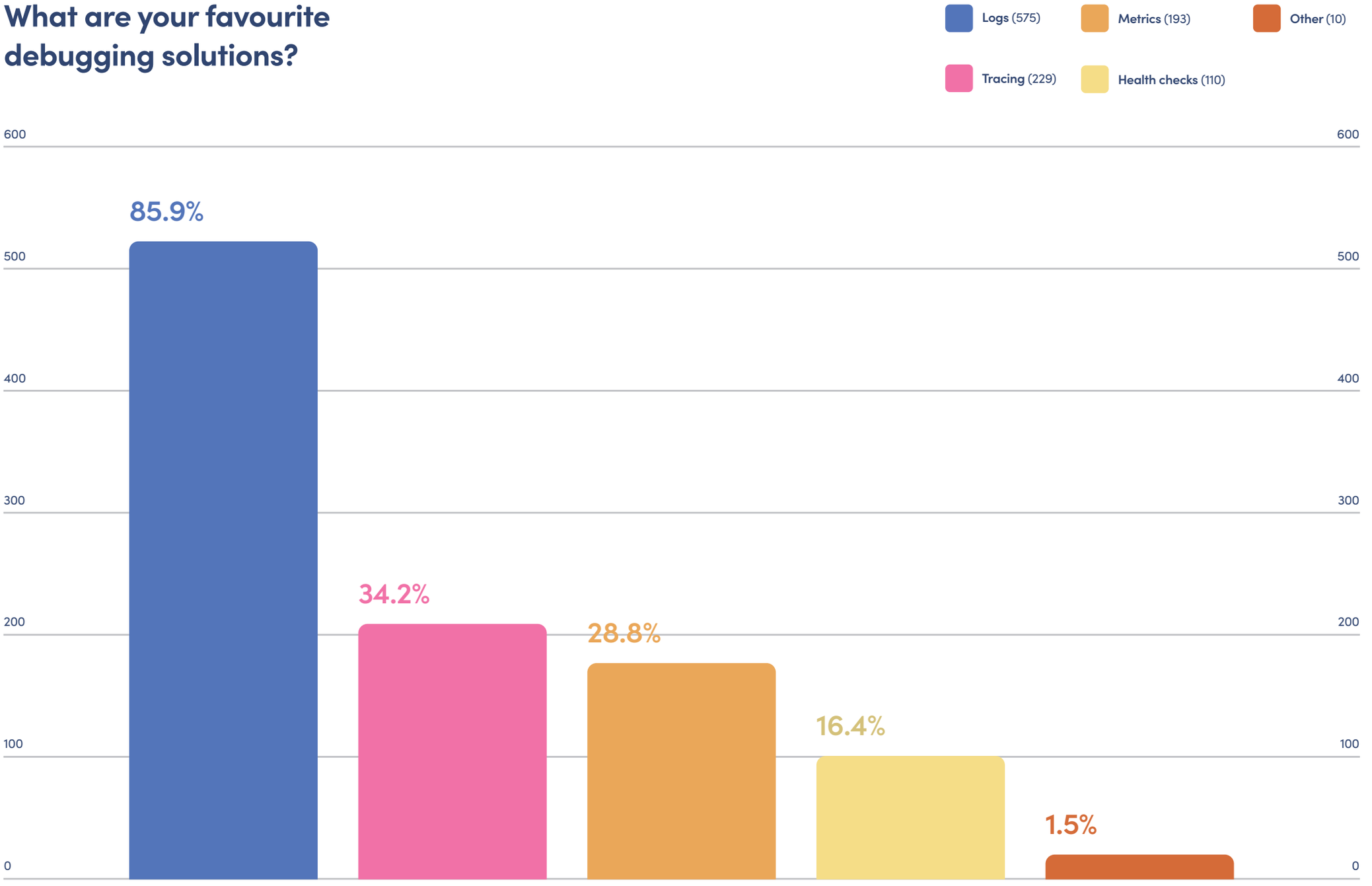
It shouldn't be that alarming, as it was a multiple-choice question. However, when getting deeper, we can see that as much as 179 respondents (27%) use ONLY logs. Knowing that logs definitely don't show you everything, it poses quite a problem. In yet another question – where we asked people how would they rate building microservices when it comes to various areas of software development (see: Chapter 2. Maturity) – maintenance and debugging were voted the most problematic areas. These two pieces of information correspond very well.

Sadly, it's also consistent with my personal observations. In general, I find that development teams often struggle with two things: predicting the consequences of going all-in with the microservices, and getting the scope of a service right. Firstly, people start buuilding services, but forget about fault tolerance, permissions, monitoring, etc. Secondly, they often tend to go overboard, making the services super small before they have anything in place to manage that effectively – and then wonder why failure modes are taking over, try to do distributed transactions, and generally end up in some form of misery.

Microservice architecture is a great invention and I must say we benefit from it a lot at Asto Digital. However, before developing microservices-based software, you must think about the future and prepare for maintenance beforehand. Starting to care about debugging in the middle of the development process – when things finally "go south" – is simply too late.

Thomas Boltze
CTO of Asto Digital

What are your favourite debugging solutions?



09

_con-sumption of APIs

Is static the future?

Consumers today are more impatient than ever, demanding top-notch performance from the applications they use.

The findings in the report regarding the consumption of APIs are consistent with the industry trends we've been noticing here at ZEIT. Consumers today are more impatient than ever, demanding top-notch performance from the applications they use.

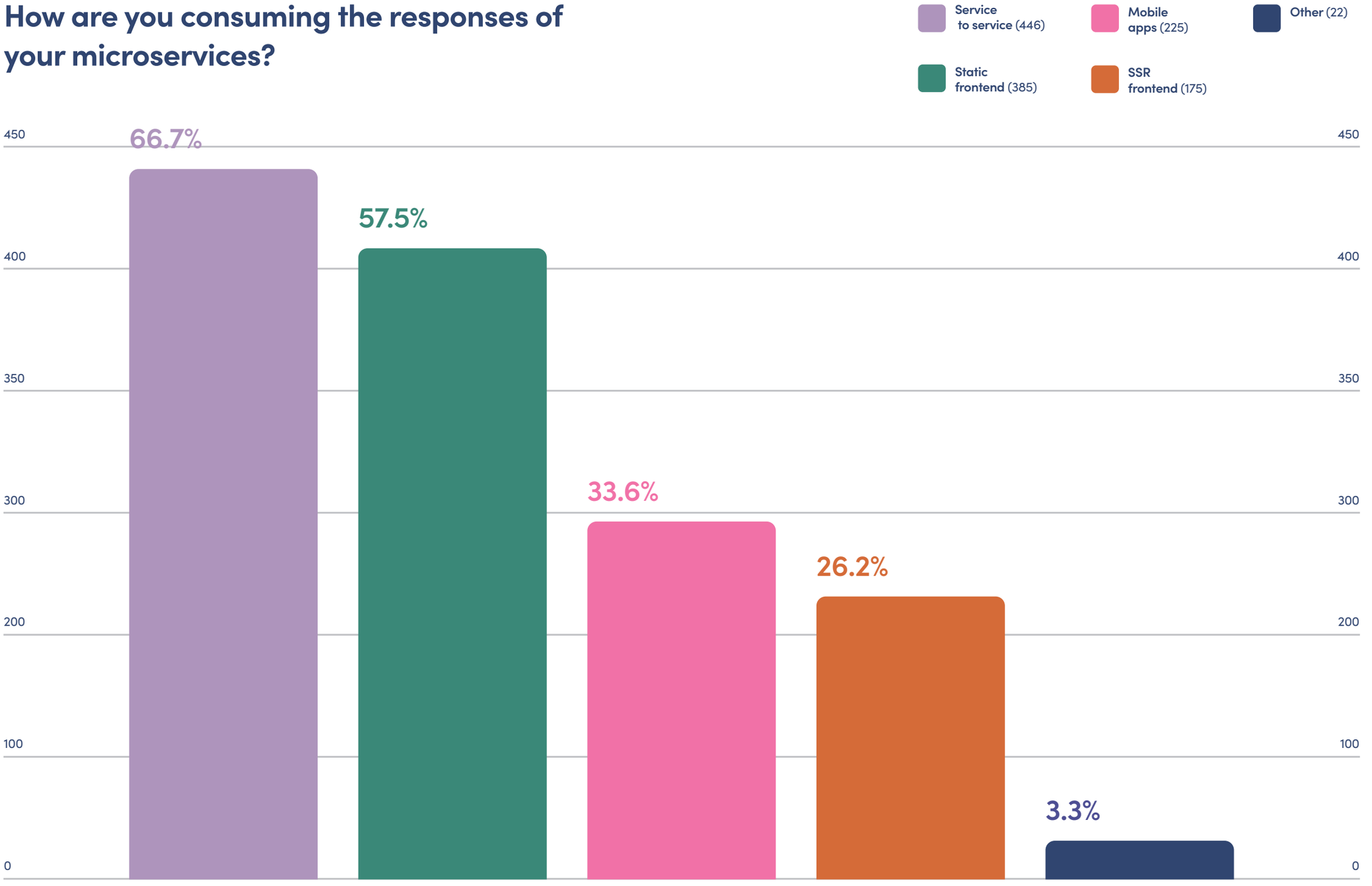
Personally, I'm particularly interested in JAMstack-powered (static) sites, which 57.5% of the survey respondents are developing. Static sites are a great choice for modern web apps. They can be aggressively cached, and served with minimal latency via Edge networks. Thanks to the proliferation of API-based solutions spanning every aspect of development, businesses can focus on building core features, testing variations, and ultimately serving their customers better.

Going static allows rapid iteration, top-notch client performance, vastly reduced development and hosting costs, zero-downtime, faster builds – there is little room to complain! With a technology stack like Next.js and ZEIT, developers are able to elevate their Git-based workflow to a Deploy–Preview–Ship workflow, unlocking all the benefits of working with static in one neat package.

Thanks to integrations with GitHub, BitBucket, GitLab, and Slack – the adoption is only increasing among remote teams. We're incredibly excited for the future.

Sarup Banskota
Head of Growth at ZEIT

How are you consuming the responses of your microservices?



10

_micro frontends

Time for the microservice revolution on frontend?

This paradigm allows you to scale up by breaking interface into separate features developed by separate teams.

I don't expect the percentage of developers using micro frontends to grow far more than the 24% that we see in the results of the survey. However, it doesn't mean that there's no place for the microservice revolution on frontend. Quite the contrary.

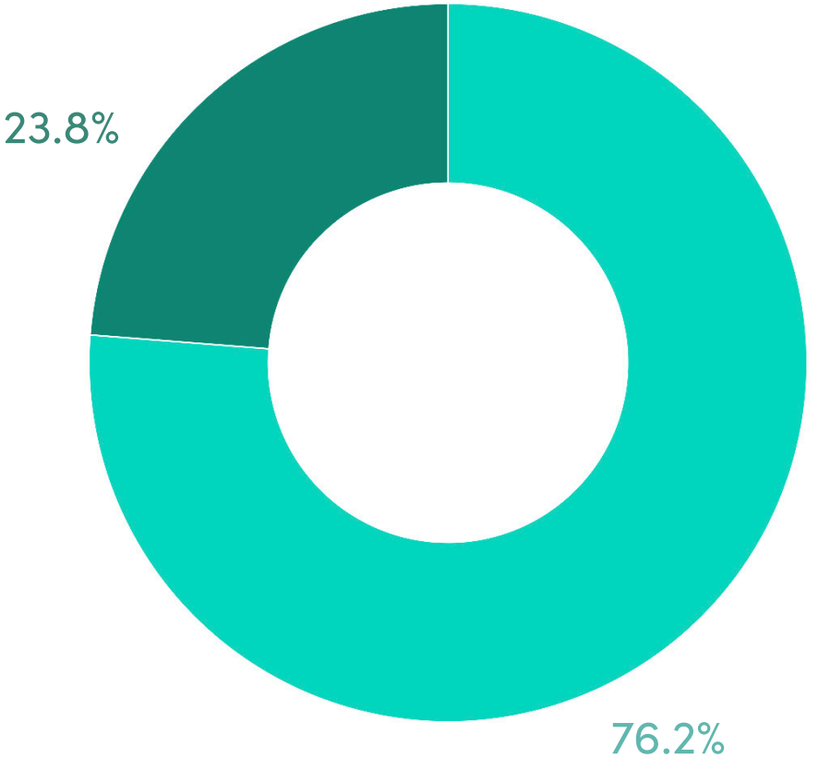
The main problem when it comes to micro frontends is that this technology is just starting to get traction and people aren't really familiar with it – which gives rise to lots of misconceptions. For example, people believe that assembling multiple micro frontends in the same view, using a few different frameworks, may lead to an app that weighs 10 MB instead of 100 KB. Well, yeah, you can do it – just as you could do, technically speaking, with a single page application – but obviously it won't work well.

That's why, some time ago, I've decided to debunk these myths and spread the knowledge regarding micro frontends. The fact is that applications tend to get more complicated on frontend, so you can't always use the same pattern. So far, we've usually built either a single page application (SPA) or one based on server-side rendering. Now, there's the need for a third option – a paradigm that allows you to scale up by breaking your interface into separate features developed by separate teams. That's the micro frontend pattern.

I discourage people from using micro frontends in new projects without understanding the business and organizational challenges which are there to solve. That's because when deciding to use micro frontends, you need to invest resources in creating the automation pipeline, in designing proper communication, in taking care of governance, etc. However, if you believe that the frontend of your app needs to be ready for scaling up, micro frontends are definitely something you should get to know better.

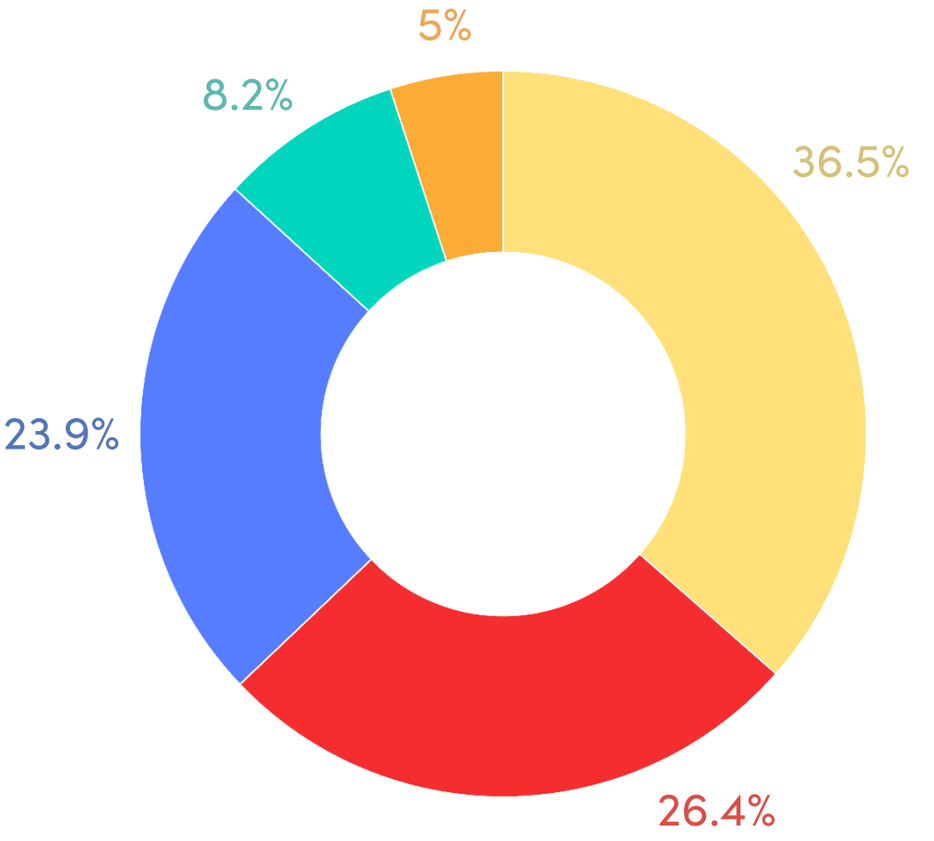
Luca Mezzalira
VP of Architecture at DAZN
The author of "Building Micro-Frontends"

Have you used micro frontends?

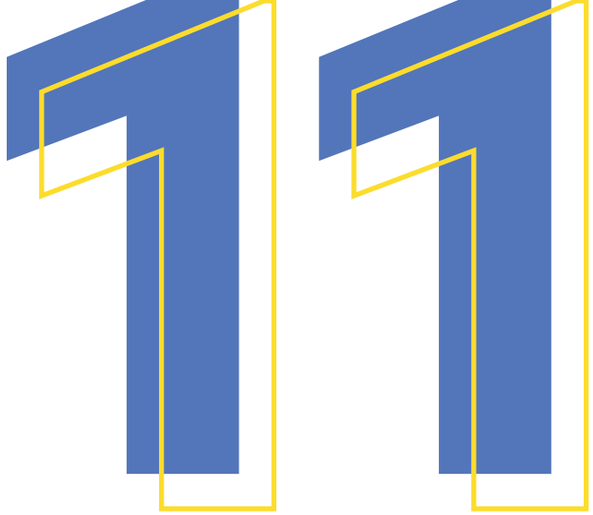


- No (510)
- Yes (159)

How do you compose your micro frontends?



- Web components (58)
- iFrame (13)
- Server-side rendering (42)
- Other (8)
- Micro frontends as npm packages (38)



_future

Microservices – the new backend architecture standard

We will see the term “microservices” virtually evaporate.

The great benefit of the microservices architecture, and the reason why it will dominate the future of software development, is that it provides a practical component architecture. In my own work building such systems, two core principles keep coming up, and their effectiveness in practice remains the reason why I believe microservices are the future: the basic principle of independent components exchanging messages, and the dynamic routing of those messages.

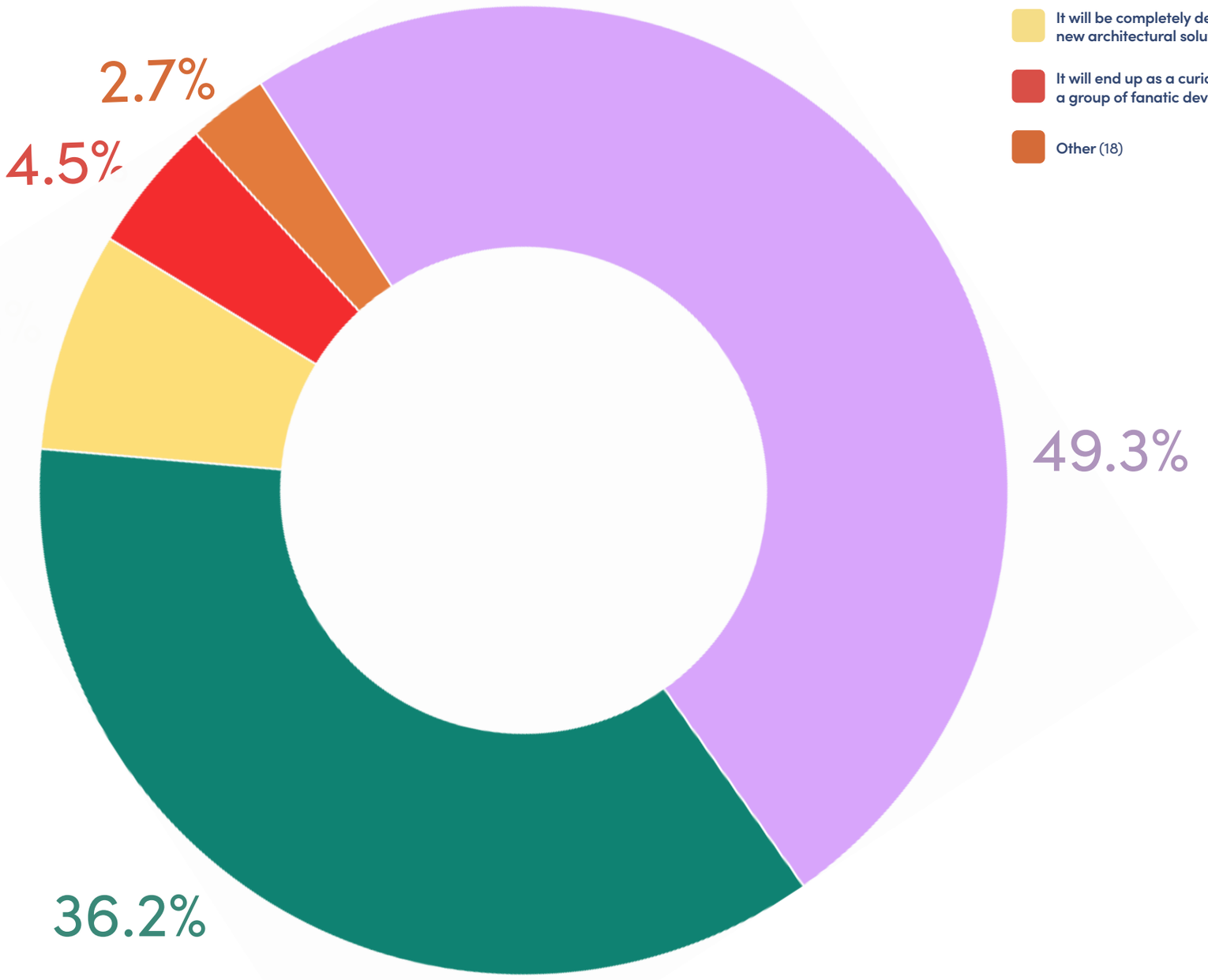
The first, transport independence, means simply that how messages are transferred, is quite irrelevant. I mean this up to the level of the messaging model – synchronous or asynchronous, it really doesn’t matter. What does matter, is that messages are the only interface. This reduces the component integration surface drastically, and enables composition – the most important attribute of a component model.

And the second principle is zero identity. Microservices and components must not know about each other. Messages are simply sent and received with no thought for destination. This approach provides the dynamic ongoing modification of live systems. Sadly, many implementations that I see in the wild still rely on concepts of identity embedded within services – this is the single greatest mistake that leads to most of the microservice horror stories.

However, with these two principles – which will become almost invisible features of the microservice substrate – we will see the term “microservices” virtually evaporate, and that means they will truly have become the primary architecture of software development.

Richard Rodger
CEO of Voxgig
Author of “The Tao of Microservices”

What do you think about the future of microservice architecture?



Software development has changed for good

We help tech managers who understand this change and want to make the most of it

Tech stack migration

Microservice architecture

Cloud migration, serverless

Modern real-time frontend

Find out how we can help you:

THE SOFTWARE HOUSE

www.tsh.io