

GUIDE TO CLOUD NATIVE MICROSERVICES

The New Stack

Guide to Cloud Native Microservices

Alex Williams, Founder & Editor-in-Chief

Core Team:

Bailey Math, AV Engineer

Benjamin Ball, Marketing Director

Gabriel H. Dinh, Executive Producer

Joab Jackson, Managing Editor

Judy Williams, Copy Editor

Kiran Oliver, Podcast Producer

Lawrence Hecht, Research Director

Libby Clark, Editorial Director

Michelle Maher, Editorial Assistant

© 2018 The New Stack. All rights reserved.

20180828

Table of Contents

Introduction 4

Sponsors 7

SECTION 01 - CONSIDERATIONS FOR A MICROSERVICES TRANSITION

01 - Introduction to Cloud Native Microservices..... 10

02 - Business and Process Decisions for a Microservices Transition..... 22

KubeCon + CloudNativeCon: Redefining Cloud Native to Focus on Business Value ... 35

SECTION 02 - DEPLOYING MICROSERVICES

03 - Migration Strategies for Microservices..... 39

04 - A Case Study of Questback’s Phased Approach to a Microservices Transition 50

05 - Microservices Security Strategy 56

06 - Deploying Microservices..... 65

07 - DevOps Practices for Microservices 73

Twistlock: Automation Makes Microservices Security Practical to Deliver..... 80

SECTION 03 - MANAGING MICROSERVICES IN PRODUCTION

08 - Microservices Monitoring..... 84

09 - Microservices Pricing 92

10 - Disaster Recovery for Microservices..... 97

11 - A Case Study of How WeatherBug Uses Microservices Without Containers..... 105

Dynatrace: When Breaking Up a Monolith, Consider Data as Much as Code..... 110

SECTION 04 - BIBLIOGRAPHY

Bibliography 113

Closing..... 123

Disclosure 124

Introduction

Application architectures that scale on sophisticated and distributed resources reflect an organization's business objectives. How the business achieves its objectives is largely dependent on the developer teams building the services. Their workflows and the technologies they employ create what's in vogue to call microservices.

Building microservices provides scale and automated workflows that get implemented through small teams that each work on specific services. The New Stack's "Guide to Cloud Native Microservices" explores how teams build, deploy and manage these scaled-out application architectures with technologies that fit the organization's objectives.

To be most effective, microservices must be built by organizations with clear business objectives. They will have teams led by experienced, full-stack developers who understand the organization's goals. These technologists are often making recommendations to senior management who must align on strategy. It is through the experiences of the teams led by full-stack developers that workflows evolve and the services become more meaningful and important in the overall deployment and management of the technologies that support the organization and its goals.

Organizations that do find success with microservices gain an approach and workflow that optimizes their compute, storage and networking resources. This allows developer teams to work independently toward a common goal across the organization. By scaling the development across individual teams, production increases. Work is completed in parallel, which may cause challenges in itself. The DevOps team must consider the compute, networking and storage requirements of all the combined developer teams. Optimizing the architecture for performance allows developers to have more capabilities

and, at the same time, allows DevOps teams to use feedback loops for continuous efficiencies and improvements.

Organizations that take the time to analyze an approach to microservices have two roads to follow.

They may choose a route to adopt microservices with consideration for the choices made by generations of teams before them. It means having a clear understanding of what microservices offer, but also facing the inherent risks and disruptions that inevitably will come when decoupling monolithic architectures. There is no return once the microservices journey begins. The decision is clear. It is assumed a microservices approach will lead to management challenges — that is without question. Senior teams with experience know there will be changes to team structure and workflows that will take time to adapt into the organization. That's fine. They have accepted that going back to the monolith has no business merit and would be unhealthy for the organization.

The road to microservices will be one many organizations will decide not to follow. These organizations have ultimately decided to optimize, as much as possible, the monolithic technology stacks that serve as core to the overall enterprise. An investment in developer-oriented approaches may be a matter to revisit in another analysis, especially as the technologies put more emphasis on the developer experience.

The work presented here by The New Stack is based upon research, reporting and discussions with senior technologists and the people using these technologies. It's a dynamic space but, ironically, still relatively unknown for most people. The community is growing fast, but also still has a sense of openness and excitement of a culture that is still developing.

How communities develop over time is a consideration for all of us. We need healthy open source communities that are inclusive and reflective of the many

backgrounds that developers can come from. Application architectures are developing fast, but there needs to be an emphasis on who is actually building the technologies so the end-user has an experience that is reflective of their own workflows and behaviors.

The New Stack's goal is to provide a comprehensive guide and resource that explains and analyzes how organizations build, deploy and manage scaled-out architectures. It's a human approach intended to help understand the dynamics of DevOps cultures, the engineers who manage them and the technologies they use. We hope you find the ebook useful and a way to think through the complexities that come when organizations, teams, workflows and technologies intersect.

Thanks for reading!

Alex Williams

Founder and Editor-in-Chief, The New Stack

Libby Clark

Editorial Director, The New Stack

Sponsors

We are grateful for the support of our ebook sponsors:



Dynatrace is the leader in Software Intelligence, purpose built for the enterprise cloud. It's the only AI-assisted, full stack and completely automated intelligence platform that provides deep insight into dynamic, web-scale, hybrid cloud ecosystems. That's why the world's leading brands trust Dynatrace to deliver perfect user experiences.



KubeCon + CloudNativeCon conferences gather adopters and technologists to further the education and advancement of cloud native computing. The vendor-neutral events feature domain experts and key maintainers behind popular projects like Kubernetes, Prometheus, gRPC, Envoy, OpenTracing and more.



Trusted by 25% of the Fortune 100, Twistlock is the most complete, automated and scalable cloud native cybersecurity platform. Purpose built for containers, serverless, and other leading technologies — Twistlock gives developers the speed they want, and CISOs the control they need.

SECTION 1

CONSIDERATIONS FOR A MICROSERVICES TRANSITION

Breaking up the monolith can be a daunting task — but also an exciting engineering and business challenge. Get started with practical advice from leaders and experts in the field.

Contributors



[Lawrence Hecht](#) is research director at The New Stack. He has been producing research reports about information technology markets for the last 15 years. Most recently, Lawrence managed “voice of the customer” surveys for 451 Research and

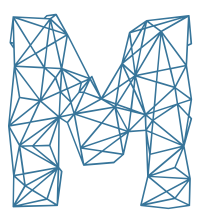
TheInfoPro about enterprise IT B2B markets such as cloud computing, data analytics and information security.



[Michelle Gienow](#) writes regularly for The New Stack, including the weekly Code Noob column. She is a frontend web developer in the making, erstwhile journalist and late-night recreational baker of peanut butter cookies.

CHAPTER 01

Introduction to Cloud Native Microservices



Microservices are an architectural approach to optimize resources that provide the compute, storage and networking for at scale services and software on sophisticated, fast, distributed infrastructure. Most organizations with any IT history have traditionally built software on virtualized technology stacks that run on machines that can be maintained manually by teams of operators. Today, developers use cloud services at scale to build application architectures and automate workloads. The days of machine-oriented architectures are passing — application-oriented infrastructures are what's in vogue. Today, the resources provide what a full-stack developer requires to build application architectures. The need of developer teams to more fully open resources for application architectures is testament to the deep demand for DevOps tooling to run on powerful distributed architectures.

Demand for technology tools, services and platforms is encompassed in what constitutes microservices. The balance of unlimited compute, networking and storage resources to run any number of services presents opportunities and obstacles. Complexity is often not discussed amid the hype that surrounds microservices these days. It's like any over-excited, new approach that catches

the technology community's attention. What may, on the surface, look like the perfect way to develop, deploy and manage software is often far more complex than what first appears. It's a journey that takes companies into the depths of understanding the business objectives, team development, workflows and services they use to build out application architectures.

Often, change is not easy for people whose technical backgrounds do not match the modern approaches that microservices offer. Microservices require organizations to rethink the existing software architecture that runs their business, and how the organization can adapt to practices that require new technical skills and a cultural shift to match. It's daunting, risky and not for everyone.

Still, business and IT teams are rushing with pronouncements like, "let's get off the monolith by next quarter!" Audacious goals aside, about 90 percent of developers are at least looking at a microservice architecture for some workload. Yet, when asked more specifically about their use in production applications, the numbers drop: 29 percent in a Dimensional Research and LightStep¹ survey and 26 percent in a DZone survey.² As with any rapidly trending Next Great Thing, however, it can be tough to sort through all the hype to understand how microservices actually apply to everyday, rubber-meets-the-road project work. It helps to start with the practical basics of microservices, then weigh some high-level benefits and drawbacks to the software architecture itself.

Defining Microservices

Microservices are an architectural approach to software development based on building an application as a collection of small services. There is no standard definition for what amount of code constitutes a "small service." But some experts say it's about the size at which a team can query the service's health with a single request that returns a "yes" or "no" answer.³ Similarly, a service

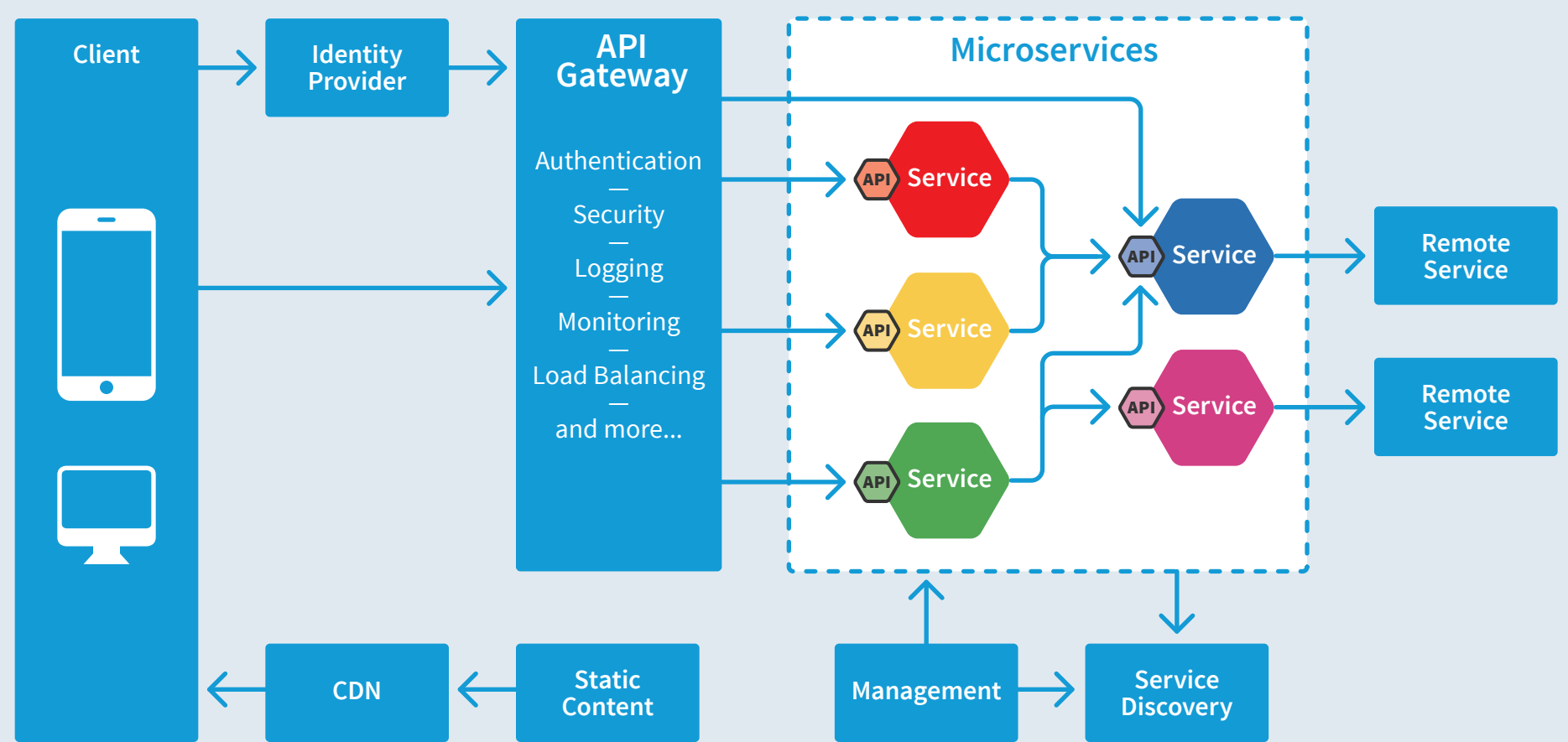
is too big if it requires more than one team to manage it. Each service has its own unique and well-defined role, runs in its own process and communicates via HTTP application programming interfaces (APIs) or messaging. Each microservice can be deployed, upgraded, scaled and restarted independently of all the sibling services in the application. They are typically orchestrated by an automated system, making it possible to have frequent updates of live applications without affecting the end users.

Individuals are comfortable with the concept of using applications. These days, an average enterprise organization uses, at minimum, a dozen different software products and integrations. Logging business expenses, schedule tracking and payroll management are a few examples of how organizations now use applications that run on cloud services.

It just makes sense to embrace compact and specialized tools that get each job done in a manner that provides an elegant user experience, similar to what

FIG 1.1: *Microservices are small, independently scaled and managed services. Each service has its own unique and well-defined role, runs in its own process and communicates via HTTP APIs or messaging.*

A Microservice Architecture



individuals get when using a consumer application for posting photos, videos and connecting with others on social networks. Microservices use the distributed architectures that embody cloud services. They scale by developing patterns that fit together in a loosely coupled manner. Like Lego™ blocks, components in a microservice snap in place to build a unified model.

First, developers identify the separate service “pieces” necessary to build their project, such as search, authentication, messaging and sales processing. Then they choose from the smorgasbord of services, libraries and code snippets available, from open source to turn-key enterprise solutions, and snap everything together into a functional application.

The Cloud Native Wave

The concept of cloud native microservices stems from the evolution of container architectures. Before container-based architectures, developers would use approaches that required building a technology stack that they then deployed on cloud services or robust enterprise architectures. The applications were machine-oriented and optimized using a generation of tools that monitored the software and its performance on cloud services and the enterprise. It was a step beyond service oriented architectures (SOA), although some would argue SOAs are simply microservices that have been rebranded by vendors to sell related offerings. There is some truth to this. Microservices can be considered a type of SOA. Containers just make the approach more widely available, and reduce the degree of risk that came with an SOA. An SOA ran on virtual machines (VMs) that required time and investment to build, deploy and run. The VMs ran on the operating system that also had to be ported to run in an SOA environment. It was heavy, manual work that left little room for taking risks to find the best way to actually run the SOA itself.

Containers changed the game, with Docker leading the charge. Docker represented the evolution of the SOA and the age of Platform as a Service

(PaaS). Docker drove adoption through its simplicity, ease of use and low risk. It packaged the Linux container technology into something accessible and usable that developers embraced. Container technologies could be built, run and managed with little overhead — a stark contrast to the heavyweight world of SOA, which required substantial investments, particularly in networking and storage.

Containers now serve as the underlying foundation for microservices, connecting through API gateways and new approaches such as gRPC. In total, containers have made SOA feasible to implement at scale by simply making technologies easier to use, with far less risk involved than ever before. Microservices are closely correlated with the use of DevOps; continuous integration and continuous delivery (CI/CD); and containers.⁴ So closely, in fact, that the terms “microservices” and “containers” are often used together. However, containers and microservices are not the same thing. A microservice may run inside a container, but it could also run as a fully provisioned virtual machine. That said, container-based — and open source — platforms, like Docker and Kubernetes, are a very effective way to develop, deploy and manage microservices. Many mature and robust tools, platforms and other services already exist in the container space, rendering containerization a natural fit for microservices-based applications.

While containers and microservices exist independently and serve different purposes, they’re often used together; consider them the PB&J of DevOps.⁵ Containers are an enabling technology for microservices, which is why microservices are often delivered in one or more containers. Since containers are isolated environments, they can be used to deploy microservices quickly and securely, regardless of the coding language used to create each microservice. Once a microservices-based application reaches any substantial size, it also becomes virtually impossible to manage without containers. Containerized microservices running on top of an orchestration platform such

as Kubernetes or Mesos — in the cloud, on premises or a hybrid of both — are the current definition of scale-out, cloud native applications.

It's important to note that although containers are the “de rigeur” way of packaging code into microservices, you could also package an entire monolithic application into a container and it wouldn't create a microservice. As cloud computing evolves further, packaging can, and likely will, change as more organizations are freed from legacy infrastructure and/or start to evaluate use cases for serverless architectures. In fact, many proponents of microservices say that they are just a stepping stone to multi-cloud and serverless computing, an emerging approach for using resources to automate functions and fill gaps in application architectures.

“I'm not happy with how our industry has coupled microservices and containers. They're an implementation detail. It's not an important abstraction, except in a world where you have a lot of legacy apps on VMs that need to migrate to the same technology stack,” said [Ben Sigelman](#), CEO and co-founder, [LightStep](#).

Benefits of Microservices

By enabling small autonomous teams to develop, deploy and scale their respective services independently, microservices essentially parallelize development — thereby speeding up the production cycle exponentially. This agility was the top reason large enterprises cited for adopting microservices in the Dimensional Research and LightStep report, followed by improved scalability.

“It's very simple: Microservices save developers from having to waste time reinventing already solved technical problems,” said [Jamie Dobson](#), co-founder and CEO of [Container Solutions](#).

Additionally, Dobson noted, continuous integration and deployment are

basically built into a microservices architecture. “Microservices take a lot of infrastructure risk out of the project straight away. With the infrastructure made almost invisible, microservice teams can iterate quickly, often in hourly cycles, so that value is increased while ‘wrong feature’ risk is decreased in a continuous fashion,” he said.

In other words, with microservices, each developer on a team gets to forget about underlying infrastructure and focus on their piece of the project. Then, in production, if individual project modules don’t work exactly right together, it’s easy enough to isolate, disassemble and reconfigure them until they do. The components are loosely coupled — again, Legos serve as an apt metaphor — and this provides the capability to run at scale with interchangeable pieces that snap into the application architecture. Their isolated and stand-alone structure brings security benefits as well, because they are easier to control through modern security platforms that automate and enforce security policies.

Engineering teams may more easily scale and maintain velocity as the organization grows. The main benefit of a microservices architecture isn’t technical — it’s in team development and people management. By contrast, monolithic applications become impossible to adapt and manage when the codebase grows to such a scale. The teams to manage application architectures of such size must never let the monolith go down. If it does, the business goes with it. Writing scripts to prevent application leakage and building varieties of patches between major version upgrades becomes an important priority for enterprise architects. Features are defined well in advance and fit into the monolith according to priority. The customer is in the middle, forcing decisions that may be short-term fixes, but pose longer-term issues, such as custom scripts that decay over time and depend on people with institutional memories of the enterprise infrastructure. That can be an ugly game in itself, as the issues customers have may not be met by the latest upgrade of the software.

“One major problem is that the [monolith] application is overwhelmingly complex. It’s simply too large for any single developer to fully understand. As a result, fixing bugs and implementing new features correctly becomes difficult and time consuming. What’s more, this tends to be a downwards spiral. If the codebase is difficult to understand, then changes won’t be made correctly.”

— [NGINX](#), “Microservices: From Design to Deployment.”⁶

Many organizations have reached a point at which the pain of managing the monolith outweighs the pain of the organizational change required to adopt a new, microservices approach. Microservices adoption is the best alternative for such organizations — though it’s not without its own challenges.

Drawbacks to Microservices

Microservices are the antithesis of the classic monolithic application, with obvious benefits. However, as with any developing technology, the early adoption learning curve can be steep. Currently, the approach is most effectively embraced by large companies like Netflix and PayPal, which have been able to shift to a microservices architecture thanks to robust in-house resources and engineering teams.

“It’s great when you are a very large, resource-rich enterprise, with individual teams able to manage each service and ensure reusability and discoverability,” said [Mathias Biilmann](#), CEO and co-founder of [Netlify](#).

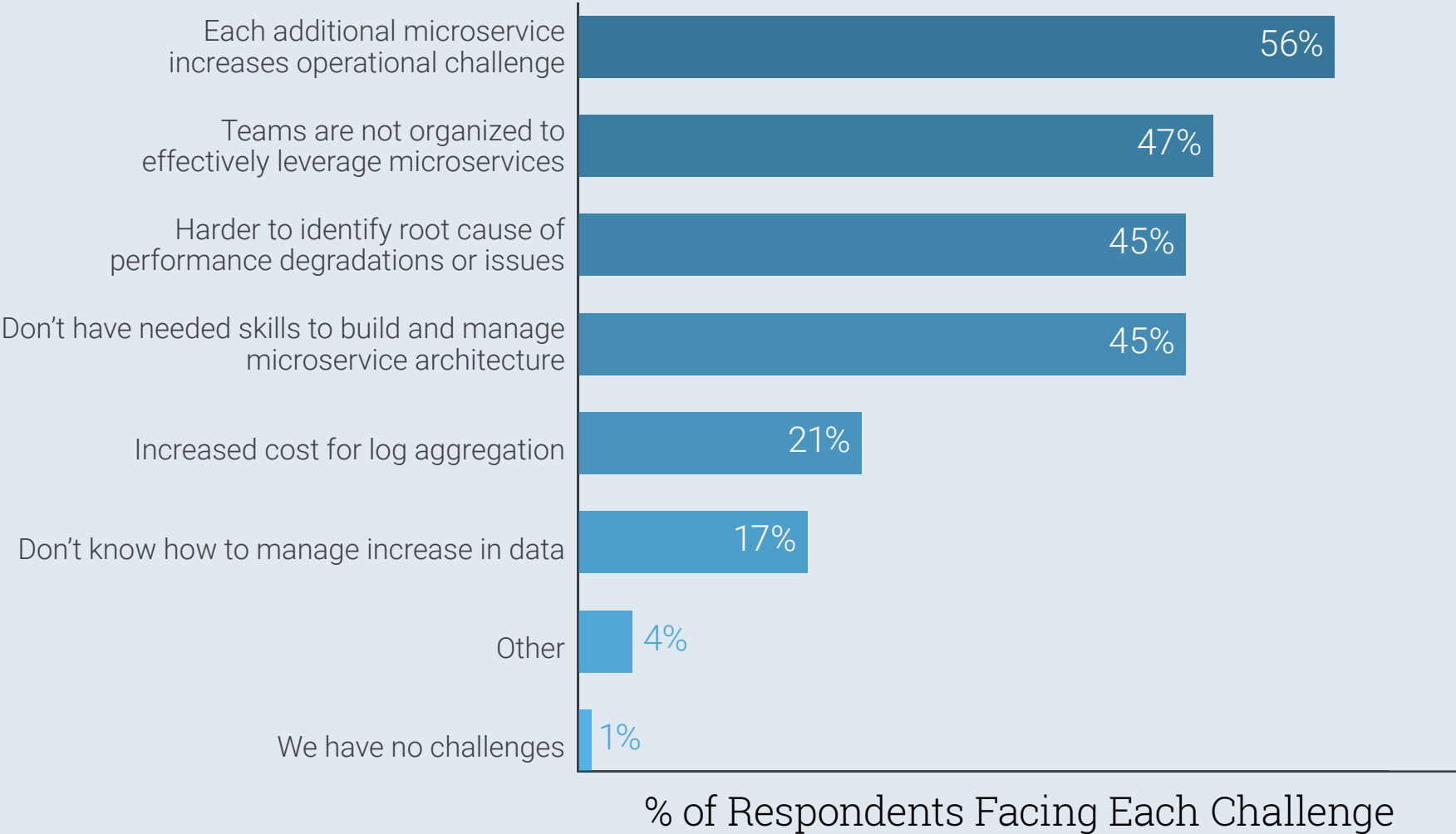
However, the pain is real for everyone else in between. Only 1 percent of enterprises using microservices said they had no challenges with the architecture, according to a Dimensional Research and LightStep report.⁷ Operational overhead, logging and monitoring challenges, and lack of skills

were cited as top challenges in the report. Moving away from a monolithic application architecture means the loss of an opinionated workflow that glues all the pieces together. Most commonly, adopting a microservices architecture increases the operational cost, as the IT team is largely responsible for integrating and maintaining the infrastructure for many different services. Teams must find the difficult balance between a microservices vision, and what realistically needs to happen to make it work and succeed.

“As we split up monoliths into microservices, we risk getting a very fragmented system where developers need to spend a lot of time and effort on gluing together services and tools, and where there’s a lack of common patterns and platforms that makes working across projects viable,” Biilmann said. “The possibilities are awe-inspiring, but in order to truly take advantage of them, we need vendors to emerge that can build the glue that enables a

FIG 1.2: *Moving to microservices most commonly creates operational challenges as the IT team is largely responsible for integrating and maintaining the infrastructure for many different services.*

Top Challenges When Using Microservices



one-click setup.”

A comparison can be drawn to the emergence of the LAMP stack. Freely available tools like Linux, the Apache web server, MySQL and PHP opened up new possibilities for web development. But the LAMP architecture truly took off when companies built integrated tooling around projects like WordPress, Drupal and Joomla.

In a true microservices approach, teams run only the exact small services they need, and nothing else. It’s a sleek setup, but these services are not aware of each other until you also step in to orchestrate them. Until recently, this implementation and orchestration piece have been beyond the engineering reach of many smaller to mid-size organizations.

Splitting a monolith into many smaller, independent services has many advantages in speed and agility, but many challenges as well.⁸ Microservices architectures can increase operational overhead for support and maintenance, as each service has its own languages and requirements. Monitoring and security become more complex and require new levels of automation and tooling. And because communication between services is now taking place over a network, it generates new requirements for service discovery, messaging, caching and fault tolerance that can strain a system and possibly lead to performance issues if not handled properly. While a service mesh addresses many of these issues, introducing one without traffic management creates its own set of problems that can lead to deep performance issues.

“The issue is, you can do all the testing that you want beforehand, and be fairly confident about the code that you’re trying to release. But then when you actually put it into production it runs into some kind of issue, because you don’t actually know how the code’s going to behave in production,” said [Christian Posta](#), chief architect for cloud development at [Red Hat](#).⁹

“Traffic management is really about decoupling a deployment from a release. A

deployment is when you have your new code, a new version, and you bring it to production, but it doesn't take any customer traffic yet. You're able to do smoke tests, and internal tests, and it lives in production. When you do a release, that's when you start to try to figure out: What traffic are we going to bring to this new version of code? If you have the ability to control the traffic to very fine grain levels [you can] segment it, control, and gradually roll out new code changes," Posta said.

Organizations without the engineering resources or skill to knit together a stable infrastructure with existing open source code and tools have struggled to make the benefits of microservices outweigh the challenges. Operational and performance issues can also plague teams that do not communicate about their services — dependencies and version compatibility — and must reverse the work that has already been written into code when they fail in production. Fortunately, a market leap forward is now happening with microservices. Many companies are now producing not just microservices themselves, but the platforms, tools and frameworks necessary for joining them seamlessly together.

Microservices Are Not for Everyone

Infrastructure for distributed services is mature, but deeper efficiencies can only come with better declarative systems that arise from refined automation techniques and improved observability. This can be tricky, as no microservice is exactly alike. They can be snowflakes, as much as any custom workflow can be. The difference is in the architecture underneath and how it fits with the ongoing development of approaches to microservices for different workloads.

It's important to set boundaries so microservices are not perceived as a panacea or a fun project offshoot that takes more management than the microservices deserve. Developers who built scaled-out microservices back in the 2014 to 2016 heyday talk about developers chatting over coffee and

deciding about the new microservice they were going to build. Now what happens if there are dozens of teams that each decide to create their own microservices? It's entirely possible to manage, but efficiencies are sacrificed and that affects the progress to optimize and attain better performance.

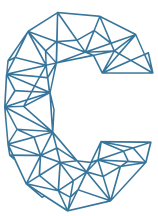
Proof that microservices are effective goes without question. But a well-built monolithic architecture can scale just as well and remains the best option in many scenarios. Running multiple instances of the same service or worker, for example, doesn't necessarily require microservices. It's also entirely possible to create unscalable microservices. It's important to first consider the problem, before identifying the best solution.

“Microservice architecture is faster, safer, cheaper — much of the time, it's the better way,” said [Chris Bach](#), Netlify's other co-founder.

However, the ecosystem is now approaching critical mass in terms of infrastructure and support around the technology. Viable workflows are now available for use by organizations of any size. And this means that microservices are fast becoming just another tool in the DevOps toolkit. The quest is for better and deeper uses of resources. That, in turn, creates new space to deliver additional services that further realize the potential of declarative and elegant workflows, tools and platforms.

CHAPTER 02

Business and Process Decisions for a Microservices Transition



Cloud native microservices are a truly exciting architectural evolution, especially when it comes to building, deploying and managing complex distributed applications. Most of the talk around microservices, however, goes straight to the technology: continuous integration and deployment, containers, orchestrators and so on. While the technical implementation is important, there's something even more critical to consider.

Microservices must fit with an organization's objectives. A developer may build microservices, but the architecture only becomes valuable when it is paired with a business objective. Critical questions must be asked, starting with the business use cases, existing teams, skills and responsibilities — the decision depends on the vision and what the organization aims to achieve.

The people within the organization who have experience in implementing complex architectures will have to pose an important question and get it answered before moving forward: Are we the right organization for a microservices architecture?

“Clients come to us looking to use microservices as a solution to a technological problem. In reality, it is often a technological solution to an organizational problem.”

— [Jamie Dobson](#), co-founder and CEO of [Container Solutions](#).

Evaluating Cloud Native Services

Evaluating cloud native microservices for enterprise adoption has less to do with a company's size, sector or even the actual technology, than with the enterprise itself. First and foremost, a microservices migration — from decision to execution — should be driven by how the enterprise is organized and managed:

- **Business model:** Is software a differentiator for the business? If so, the developer team must continue to grow and scale as the organization requires more resources and services capabilities. Microservices-based architectures allow for faster, iterative development that can be used in workflows across multiple teams. Organizations with a reliance on proprietary, monolithic solutions will not be as well-suited to a microservices approach. The commercial software agreements to keep systems-of-record managed to service-level agreements (SLAs) means a radical shift for companies if they choose to follow a route that takes them into microservices discussions. Microservices will likely be more costly to implement for organizations fully reliant on commercial software platforms. The engineering support and overhead needed for microservices will cost more than they are worth in agility and scalability.
- **Culture and internal processes:** Microservices require a new set of tools and processes — and the breaking of old ones. It can just be a difficult shift for an organization that is responsible for managing monoliths to make an abrupt change in workflows. Embracing DevOps principles is the key to

success with microservices. Yet, as an example, teams may be resistant to moving from a traditional waterfall methodology to an agile approach.

“And the resistance is not entirely unreasonable, if you realize that the humans involved have what they’re used to— they have maybe retirement in sight, not too far in the future — and they dislike the idea of everything changing when they just got it the way they liked it,” said Bridget Kromhout, principal cloud developer advocate at Microsoft.¹⁰

The fundamental complexity of microservices is in the application architecture itself: Every service can require its own support team, tools and infrastructure depending on the architecture. And not every company is in the right place to make the move. Not that adopting the architecture becomes impossible, experts stress, just that the process will be lengthier or more complicated. For many organizations with the wrong business motivations or culture, the cost will be higher than the benefits.

“We can’t solve ... every single problem that we’re going to have in our organization by just implementing the right technical solution, right? Because our organizations are complex systems that also have people in them who may act in unpredictable ways.”

— [Bridget Kromhout](#), principal cloud developer advocate at [Microsoft](#).¹¹

So when might microservices not be a good fit for an enterprise?

- **Sector sensitivities:** Certain industries, for example, financial services and health care, face legal, reporting and compliance requirements that need to be reconciled with newer technology.
- **The venerability factor:** A global company in business for decades, especially one with an average workforce retention of more than ten years,

will very likely have a harder time navigating the seismic shift to a completely new architecture than will a younger, more compact or simply more agile organization.

- **“Stuck” companies:** These are risk-averse companies with a long decision-making chain and rigid hierarchy. Ultimately these organizations are not well suited, and possibly even resistant, to the rapid adaptation required when adopting a new and responsive microservices paradigm.

Jonathan Owens, lead site reliability engineer at [New Relic](#), suggests that organizations considering a move to a container and microservices architecture should ask themselves the following questions: **12**

- What product does your operations group provide to developers and what abstraction layer does that product use?
- Is that product the right one for your business or are containers a better fit?
- Are containers so much better that you’re willing to change the abstraction, and therefore the entire product your operations team offers, in order to use them?
- Are you ready to create new roles to manage the scale and reliability of this new abstraction?

“No organization changes like this overnight. The journey from an idealized new architecture to the first production deploy requires changing many minds and creating new processes, which isn’t always fun,” Owens said.

Finding engineers with microservices expertise who can make the necessary tooling and architecture decisions can also be tough. Such experts include the elusive “full stack developers” who understand the application at every layer: from the network and hosting environment, to data modeling, business logic, APIs and user interface and user experience (UI/UX). **13** These individuals are

in the unique position to see how the technical architecture and the organization are interrelated. To make a successful microservices transition, an organization needs a technical architecture that is built to scale, but equally important is the team to maintain the structure.

This is why many organizations undertaking the transition to microservices choose to work with a professional services company that specializes in helping clients build cloud native applications using a variety of different architectures. Such consultants can help assess the organization's need and suitability for microservices, or direct them to more appropriate alternatives.

Are Microservices the Best Fit?

An organization that has a business reason for microservices makes the transition with the confidence of its team. The teams that lead the projects have weight in the organization and can start setting best practices that fit with existing workflows. Services can be adopted that propel the overall development of the application architecture and ready the organization for using more resources to run microservices.

Getting to the point of readiness takes skill and people management. The services that suit a developer team will define the microservice. The goal is to make the microservice a value that builds upon its base and continually optimizes the developer experience.

Evaluating the application's responsibilities is the first step in defining the components of a microservices application, said [Netlify](#) Chief Technology Officer (CTO) [David Calavera](#) — a microservices veteran from previous work at Docker and GitHub.

Determining the interdependencies of the application's responsibilities sets the structure for the microservice. [Connascence](#) is a metric to evaluate an application's components and interconnections. Two or more components are

said to be connascent if when you change one of them, you also have to change the other.

“With this relationship in mind, you can better evaluate if it’s worth having different microservices, or if you should keep your monolith architecture,” Calavera explained. In addition to these interdependencies, he said, teams must bear in mind that separating these components into microservices will introduce a network connection between them — which inevitably adds complexity to the system.

Here you can see that application architecture development is a direct result of how individuals and teams interact and communicate about their own — and overlapping — orchestrations. In this, it’s apparent how architectures, such as Kubernetes, have become more important. As more developers are added and the application gets more sophisticated, so does the overall complexity of the architecture. But as we have well seen, these application architectures are not for everybody.

“You don’t want to end up adding unnecessary complexity at the cost of an ideal architecture,” cautioned Calavera.

Assessing a Team’s Readiness

Having determined that there is a fit for microservices, the process of iterative development begins. That’s the approach to follow, no matter what the project. The transition should be gradual and distributed across small teams, with each team tracking its workflow. In this manner, teams can map the architecture independent of the underlying delivery mechanism. The most important aspects of development are the workflows the teams use to build, deploy and manage the application architecture. By breaking their workflows into tasks that are documented and automated in Git, the process becomes declarative, making the transition simpler and the infrastructure increasingly automated, as it progresses to additional teams and services.

This was once called application lifecycle management (ALM), a field dominated by the big tech giants of the late 1990s. It's only now beginning to change and achieve a fuller scope with the advent of practices rooted in DevOps. The ALM market comes out of the dependencies on monoliths and the cumbersome workflows they required with so many dimensions that needed to be connected. There are countless examples of software categories that cater to the monolith market. But many of these providers were also building on monolith patterns. Scrum practices became the most sophisticated of operations, only to face their own weight and heavy constraints. Updates needed to be kept current, patches made and software agreements managed. Consultation costs became part of the setup, the deployment and, increasingly, the management of the monoliths. Go to a large vendor conference and they are all there, offering consulting services for managing a monolith. Most are focused on workflow patterns that have the sole intention of keeping the monolith running without going down. Keep labor costs low, work on a budget that follows a top-down workflow and apply that model as much as possible to the new dimensions of what cloud services offer.

Microservices strategies reflect the most vital shift of the past five years in the way software defines an organization. The monolith culture that divides developers and operations teams is finding better balance, but deep separations still exist that call for a new thinking and adjustment to new times.

The chasm may seem like it is narrowing, but the comparison of full-stack developers to mainstream developers is by far favoring the mainstream in terms of employment trends. The mainstream developers still follow Scrum practices and are there to build on the monolith. But they are finding a path in their own right through platforms and services that allow for simpler workflows. It's the workflows that constitute the most important shift in behavior patterns. The workflows create new dimensions for understanding inherent trade-offs that come when building, deploying and managing

systems and software architectures.

Deep efficiencies and capabilities come from developing patterns that can be used for compute, networking and storage. Companies that adopt these practices will benefit from the understanding that comes with comprehensive analytics that will then give them more confidence in meeting customer needs.

Full-stack developers serve as guides on these journeys. They have a relatively deeper understanding of the different layers in the technology stack. They can help determine whether or not a team is ready to undertake a transition to microservices and how to best approach the transition.

Teams must be ready to handle the extra complexity of having several services running in production, rather than only one. It will overturn every process teams already have in place, from version control and application deployment, to monitoring and management in production.

They must identify and assess each one of the processes involved in the development, testing and maintenance of a production service, then prepare specific answers for how to adapt or replace them.

Implementing microservices is more about team structure — so each team has clear ownership over a service — rather than creating a specific technical functionality, said [Matt Klein](#), Envoy maintainer and software engineer at [Lyft](#), in a media and analyst briefing at [KubeCon + CloudNativeCon](#) in Copenhagen. “This all comes back to people. ... What are the right ways of setting up people so they can work together?”

An organization may be better off with a monolith if the underlying infrastructure is tightly bound — if the interconnections are so interdependent that the monolith can’t be broken into multiple pieces without it all falling apart. Separate developer teams, in a microservices fashion, can’t work on the monolith without overwriting each other’s work. Version control is

unmanageable unless strictly supervised.

A microservices culture is loosely coupled, making services interchangeable. The parts reflect the workflow adopted by the organization and the teams that manage the overall application architectures. It's a fusion of practices that are connected through Git, declarative environments and connected services across internal and external resources.

The factors inherent to monolithic cultures preclude managers from hiring developers for microservices projects. It's just unwieldy for multiple teams to work on one monolith. The complexities come when one team may overwrite another's work. Version control becomes a nightmare. Until the pieces of the monolith are distinct as microservices, the monolith needs to be managed in a manner that keeps the pieces bound and the overall monolith running.

Another way to think about it is in terms of each microservice as a deployment, said [Dr. Donna Malayeri](#), product manager at [Pulumi](#).¹⁴ “The platform doesn't natively define what deployment means. It's up to you to define.” Some platforms, such as Amazon's EC2 VM, in some respects are better suited to a monolithic application than microservices, Malayeri said. Breaking up the application into microservices is just as much, or more, work, because now the team must deploy and coordinate all the components that must be operable for the monolith to maintain uptime requirements.

Preparing for Launch

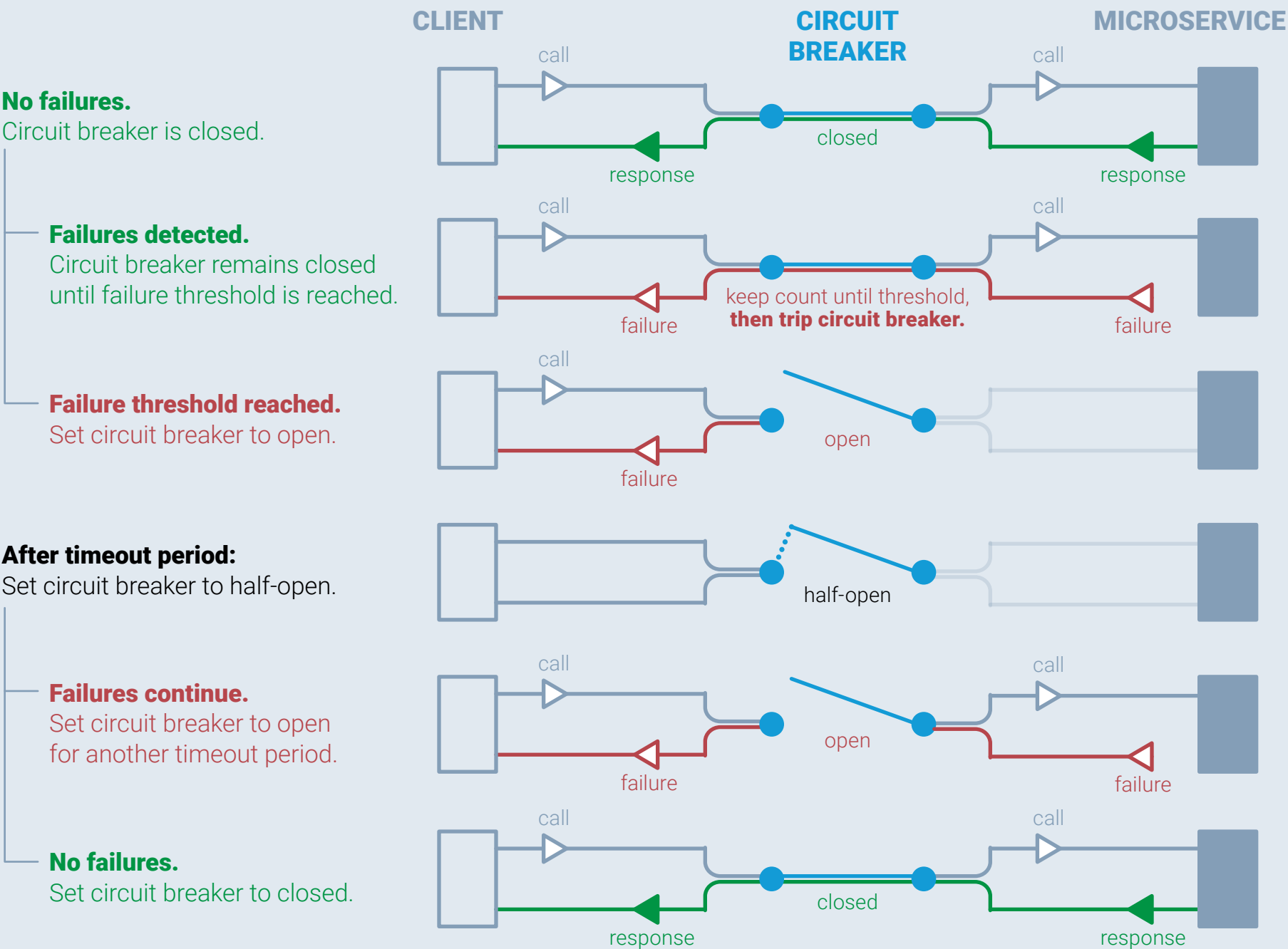
The best transition possible is the one that users never notice. Unfortunately, mistakes in the process can lead to terrible outages. But beyond an honest assessment of existing capabilities, there isn't much else a team can do to best prepare for the smoothest possible migration. There are simply too many variables at play in complex, distributed systems interconnected through a network. To some extent, the only way to know if it works is to put it into production. Thus, microservices migrations are always planned to take place in

stages. The best approach is to find a balance between the safety and the effectiveness of the migration, also known as the [efficiency–thoroughness trade-off](#) (ETTO) principle.

“ For safety and security reasons, it’s never advisable to jump all the way in to microservices. When transitioning to a different architecture, always remember that keeping the full service up and running is your first priority.”
— [David Calavera](#), CTO at [Netlify](#).

FIG 2.1: A circuit breaker object can monitor remote function calls for failures and return error messages in order to prevent cascading failures.

Circuit Breaker Pattern



Branch by Abstraction Pattern

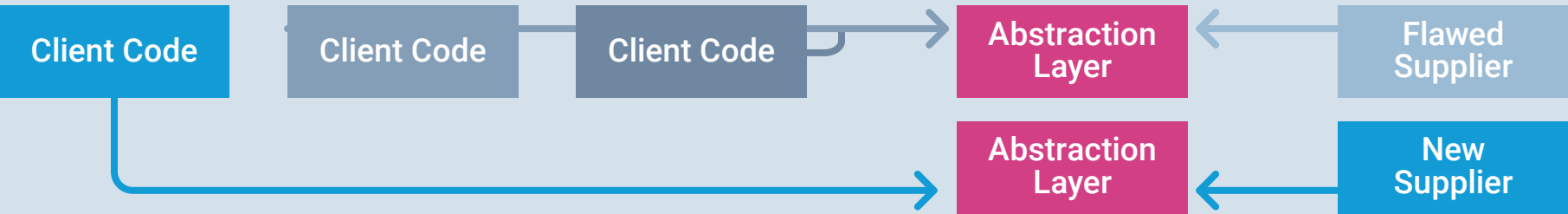
1. Current State



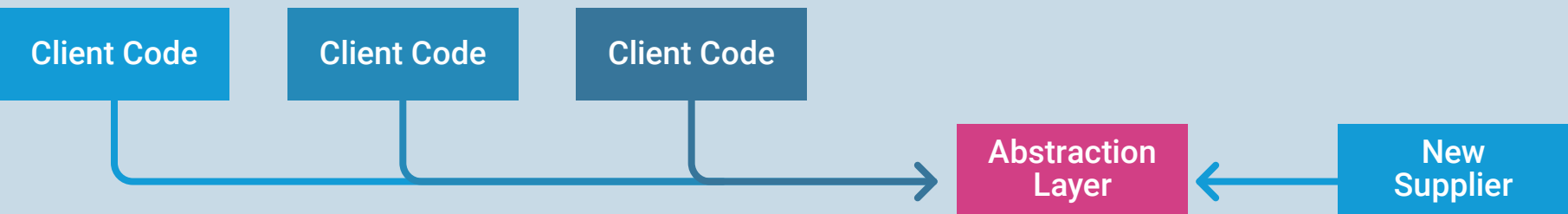
2. Create and Place Abstraction Layer as Intermediary



3. Create New Supplier and Switch Code



4. Full Swap



Source: <https://martinfowler.com/bliki/BranchByAbstraction.html>

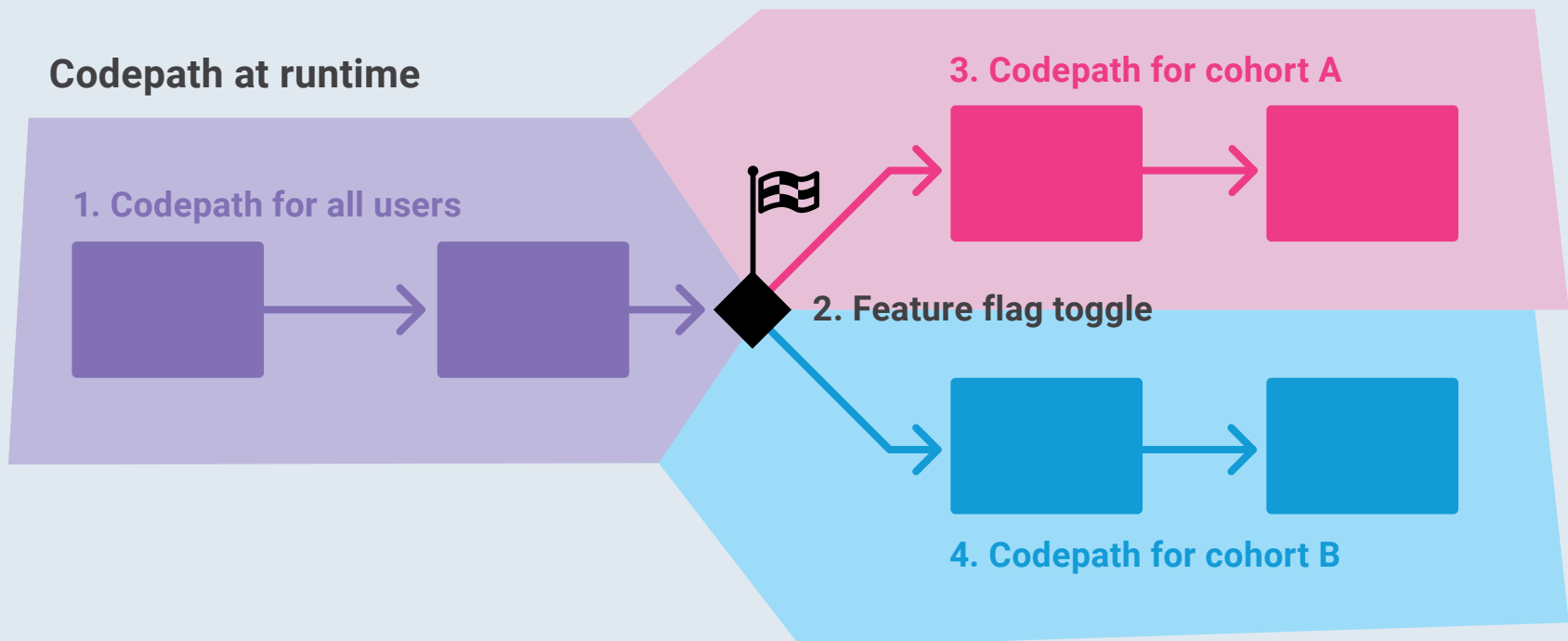
© 2018 THE NEW STACK

FIG 2.2: The Branch by Abstraction pattern allows for a gradual, large-scale system change.

Fortunately, previous pilgrims have helped prepare the path, chief among them [Martin Fowler](#). There are several microservice architecture patterns that, when combined, help create a smooth process. Very briefly, these are:

- The **Circuit Breaker** pattern: [Circuit Breakers](#) abort code execution when something unexpected happens; for example, when the network goes down and two microservices cannot communicate with each other. “This pattern forces us to think about what to do in that scenario, and there are several libraries that implement this pattern in different languages,” said Calavera.

Feature Flag Pattern



Source: <https://martinfowler.com/articles/feature-toggles.html>

© 2018 THE NEW STACK

FIG 2.3: Feature flags can divert a portion of traffic to a new microservice.

- The **Branch by Abstraction** pattern: A technique for gradually undertaking a large-scale change to a software system, [Branch by Abstraction](#) allows you to release the system regularly while the change is still in progress. “This helps introduce alternative logic to perform an operation,” explained Calavera. “In this case, the alternative logic could be to send a message to a microservice rather than using our current application’s logic.” There are several libraries that implement this pattern in different languages, including [Go](#) and [Ruby](#).
- The **Feature Flag** pattern: Also known as [feature toggles](#), these give the ability to change the execution path within an application in real time. “We can implement a flag that allows us to send some traffic to our new microservice, but not all of it,” said Calavera. Again, multiple libraries exist that implement this pattern in different languages.

In order to observe the behavior of the system during transition, Calavera added, “all the previously mentioned libraries have some kind of support to emit events, logs and metrics that we can feed to our favorite monitoring system.”

Best of all, the patterns can be combined to create a system that provides significant control over the transition to a microservices architecture. For example: By combining the branch-by-abstraction pattern with a circuit breaker, it becomes possible to implement a change that allows directing traffic to the new microservice. But if the circuit breaker gets tripped by an unexpected error, the system falls back to the old application logic.

With the right business model and cultural fit, many companies have seen great success moving away from a monolith to a microservices architecture. The key is careful planning and assessment of an organization's needs, structure and talents. The path is made by walking.

Redefining Cloud Native to Focus on Business Value



Cloud native isn't limited to containers and microservices orchestration. This was the main conclusion the Cloud Native Computing Foundation (CNCF) reached when it set out this year to re-define

the term "cloud native," said Justin Garrison, co-author of the book Cloud Native Infrastructure. Under the new definition, CNCF recognizes that cloud native is not just a set of technologies to adopt, but that it also reflects a change in an organization's structure and processes.

"Technology is not the point. The point is business value and that may be about speed of deployment and speed of shipping function," Liz Rice, technology evangelist at Aqua Security and co-chair of the CNCF's KubeCon + CloudNativeCon event, said.

This is why Garrison and Rice have lately seen even technologically mature organizations deciding to rollback their microservices architectures and return to a monolith. These organizations realized that their own internal processes and teams just weren't set up for microservices. Does that mean they're not running cloud native applications? Not at all.

"It doesn't matter if you own five little services or one big service, as long as people can iterate quickly and gain velocity to solve business problems," Garrison said.

Garrison and Rice have learned a lot from other end users and developers in the CNCF's open source community. Organizations that

get involved can avoid some of the pitfalls that those on the leading edge of adoption have encountered. For example, many organizations fall into the trap of using familiar tools to try to solve issues that are specific to cloud native environments, Garrison said.

“Do I declare my pods in Terraform? You can. Should you? Maybe not,” he said. “There’s different levels of abstractions where the tools just don’t make sense anymore.”

Conversely, organizations may also misstep by misusing a new cloud native tool, like a container image scanner that they end up rolling back because it gives too many false positives, Rice said. They don’t know that there is a good explanation for this and they stop scanning the images altogether.

Instead, “they’re pretty much just relying on hearing about people saying there’s this terrible meltdown/spectre/heartbleed/whatever, and then checking whether that applies to them,” she said. “They don’t notice when a real problem comes along.”

Garrison and Rice discuss the definition of cloud native, why some organizations have decided to move away from microservices and back to a monolith, and some of the approaches organizations are taking to cloud native implementations. [Listen on SoundCloud.](#)



Liz Rice is a technology evangelist with container security specialists [Aqua Security](#), where she also works on container-related open source projects including [kube-bench](#) and [manifesto](#). This year she is co-chair of the CNCF’s [KubeCon + CloudNativeCon](#) events taking place in Copenhagen, Shanghai and Seattle.



Justin Garrison loves open source almost as much as he loves community. He frequently shares his findings and tries to disseminate knowledge through practical lessons and unique examples. He is an active member in many communities and constantly questions the status quo.

SECTION 02

DEPLOYING MICROSERVICES

Organizations must take an informed, methodical and phased approach to make a successful microservices rollout. DevOps is the key.

Contributors



[Alex Handy](#) is a 20-year veteran technology journalist who cut his teeth covering the launch of the first iMac. His work has appeared in Wired, the Atlanta Journal Constitution and the Austin American-Statesman. He is also the founder and director of the Museum of Art and Digital Entertainment (themade.org) a nonprofit video game museum located in Oakland. He consults at VonGuard.net.



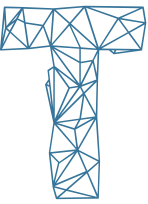
[B. Cameron Gain](#)'s obsession with computers began when he hacked a Space Invaders console to play all day for 25 cents at the local video arcade in the early 1980s. He then started writing code for very elementary games on the family Commodore 64, and programming in BASIC on the high school PC. He has since become a long-time and steadfast Linux advocate and loves to write about IT and tech. His byline has appeared in Wired, PC World, CIO, Technology Review, Popular Science, and Automotive News.



[Todd R. Weiss](#) is a technology journalist who has been covering enterprise IT since 2000. Most recently he was a senior writer for eWEEK.com covering all things mobile. In addition to writing for The New Stack, he has also written for CITEworld, Computerworld, PCWorld, Linux.com and TechTarget.

CHAPTER 03

Migration Strategies for Microservices

 he road to microservices is long, winding, and contains many off-ramps to confusing interchanges. The journey starts with the organization's business objective. People don't get very far on the microservices road of dreams when the objective is not clear. Business objectives drive team development, workflows and the adoption of services to define, build, run and maintain the microservices. Terms such as “infrastructure consolidation” and “operation cost reductions” are important only in their contextual meaning. An organization's engineering team may also be driven by the perception of marketing terminology upon the larger business and enterprise. A search for a utopian city in the sky is often unfruitful and empty. The city is gone and never really existed in the first place. In reality, it's very easy for engineers to find themselves wandering the jungle of service discovery or the dark back alleys of QEMU.

Migrating corporate applications and services to the cloud is at the top of many IT to-do lists, but the very idea covers a great deal of territory. Are the migrating services modern, or legacy? How do they communicate with one another? And perhaps most importantly, where is the data going to live? To provide a starting point for exploration, it's helpful to begin thinking in a

DevOps mindset. Rather than be limited by an existing infrastructure or application architecture, a DevOps engineer considers what may be possible to achieve with a new set of tools and practices, and then finds a way to begin iterating and evolving from the existing state.

Taking this DevOps approach, most microservices migrations are done in a piecemeal fashion — starting with a monolith and breaking off one manageable and well-defined service at a time. In this way, organizations can maintain their legacy systems, some of which may never make the transition to microservices, while simultaneously moving portions to the cloud — into containers and microservices — where it makes the most sense.

This careful, reasoned and iterative approach also allows time for teams to adapt to the new structure and processes that microservices require. They can ensure the new services are migrated and functioning well before moving on to the next service. And they can begin to adopt standard tools and approaches for creating new services that work best for the organization's business goals and culture.

“The most important things [in a microservices migration] are to have some kind of standard set of tools and technologies you use within your organization and to have the person who made those decisions know what they're talking about. Then start [small] and observe for six months, rather than start with 50 services.”

— [Ben Sigelman](#), CEO and co-founder, [LightStep](#).¹⁵

Hybrid Cloud is a Stepping Stone to Microservices

To understand how an organization might begin breaking apart a legacy application, let's use a database migration as an example. For traditional and

legacy enterprise applications, monolithic databases are the norm. A single, highly-tuned instance of Oracle, DB2 or even a small MySQL cluster handles the longer-term storage of user and application data. Meanwhile, a multitude of applications reach inside the database and get what they need done, perhaps even using stored procedures.

The cloud breaks this pattern. Data storage is now distributed according to its function for various workloads: Each service has its own data requirements and thus its own database, and data queries may span multiple services,¹⁶ so that data is shared via an API. Stored procedures should be made into microservices, databases must be highly scalable, and larger data stores that contain relational information are relegated to jobs like “data lake” storage — a vast repository of raw, unstructured data that’s off the beaten path of second-to-second transactions.¹⁷

Where once there was a monolith connected to another monolith, microservices require the vivisection of those systems, and the splaying of their innards upon the public cloud. The very act of deconstructing a monolith can require intense infrastructure rearchitecture, and a large amount of actual software development work; the sort no developer wants to handle.

There are ways around this problem, however. Many companies offer solutions designed to help bring existing applications into a hybrid cloud model.

Companies such as [Amazon](#), [Google](#), [Microsoft](#), [Platform9](#), [Portworx](#) and [Rancher](#) offer a way to bring the open source container orchestration platform Kubernetes to bear upon such systems. Such products allow teams to designate which workloads will run on the specified types of cloud infrastructure according to set policies. This is one defining feature of cloud native, microservices-based application architectures. Because the services are loosely coupled, they can be deployed and managed separately according to predefined, policy-driven resource allocations defined by a central IT team.¹⁸

For some services, this halfway point favored by Platform9 and [VMware](#) is a good stepping stone for monolithic applications that may be prohibitively expensive to turn into microservices. This is the case for the majority of companies — 57 percent of IT decision makers say their companies do a mix of building new cloud native applications and refactoring existing applications, according to a [Cloud Foundry Foundation](#) report.¹⁹

There are a generation of companies that have architectures that predate the client-server boom. Systems of record keep these businesses running. For the teams managing these legacy systems, the idea is more about determining how far back in the stack the systems can be pushed to make new space for architectures that allow the business to be more relevant through new application architectures. Offering the flexibility to manage virtual machines for legacy applications, and containers for modern ones from the same console means one layer of complexity for migration is at least abstracted away.

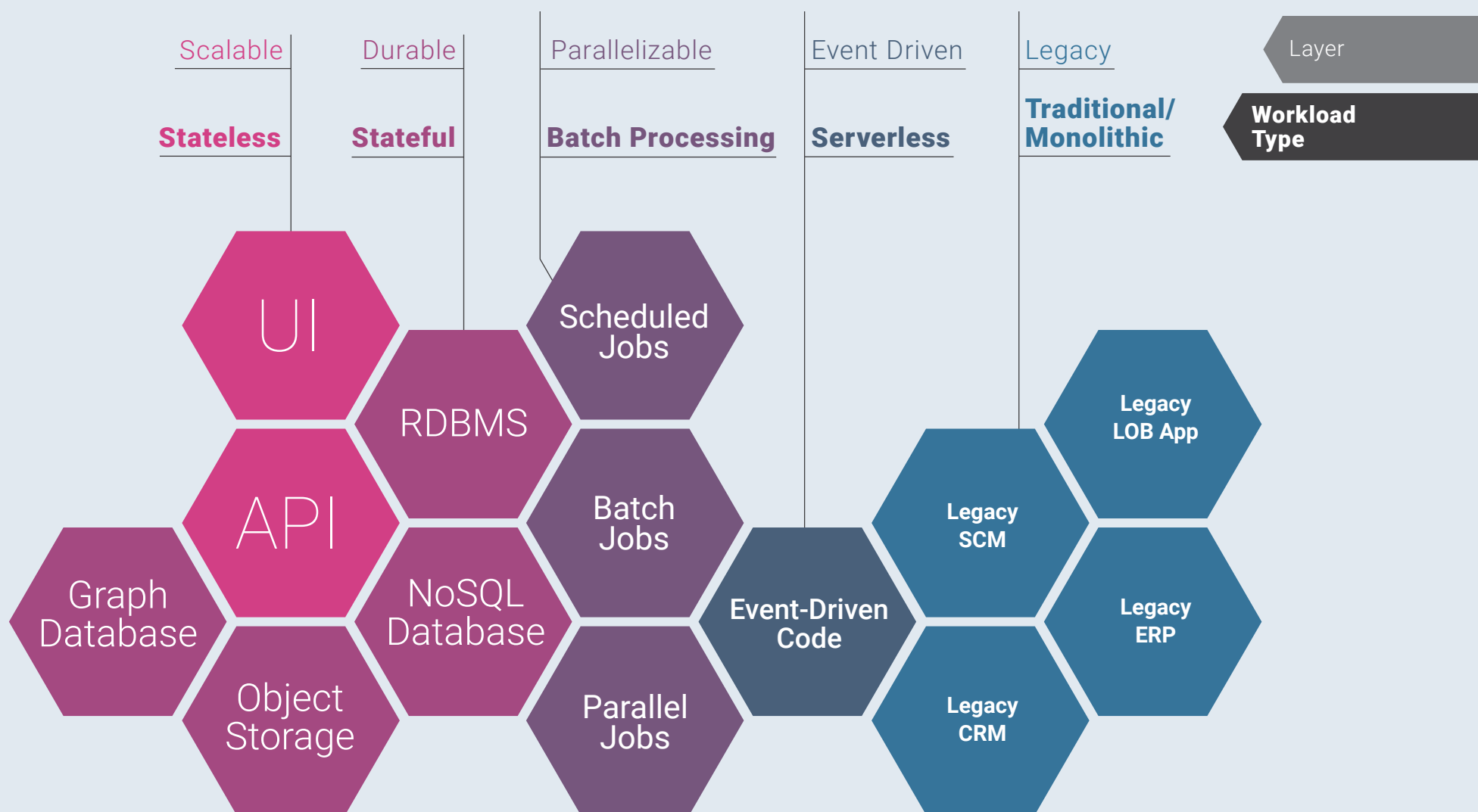
Infrastructure Considerations: A Stateless Mindset

Yet, at the end of the day, the real goal is to provide a path for past and future services to easily be integrated into a cloud architecture that fits the organization or the business team leading the project.

Considerations for architecture development include the resources and all that goes with a cloud native approach. It's a matter of having the teams that will have the knowledge and depth of experience to implement load-balancing capabilities, service discovery, automated scaling and a host of other capabilities. Large organizations with substantial resources and/or experience may choose to build and troubleshoot their own infrastructure. Platform types are numerous for those reviewing options for defining and building resources for application and microservices development.

Defining microservices according to functionality and deployment patterns is

Mapping Layers to Cloud Native Workloads



Source: Janakiram MSV

© 2018 **THE NEW STACK**

FIG 3.1: Each layer of a cloud native application runs specific microservices designed to perform a fine-grained task.

an emerging best practice for cloud native application design and a trend that we are continuing to follow. Cloud native applications are composed of various logical layers, grouped according to workload. Each layer runs specific microservices designed to perform a fine-grained task. Some of these microservices are stateless, while others are stateful and durable. Certain parts of the application may run as batch processes. Code snippets may be deployed as functions that respond to events and alerts. See our recent [ebook on CI/CD with Kubernetes](#) for a more in-depth description of each layer of a cloud native application.

As cloud native computing advances, resources are increasingly available for compute, storage and networking. It is now just a matter of programming the internet and what is increasingly described as physical computing. The world is getting programmed. Resource-driven automation is a natural outcome for

developers to make increasingly more of the services and platforms available. Application architectures, in turn, have adapted to available resources, which have changed how services are built, run and maintained. Cloud-based applications are best without state. Building an application based on cloud native microservices then becomes a matter of architecting application workloads to maximize statelessness.

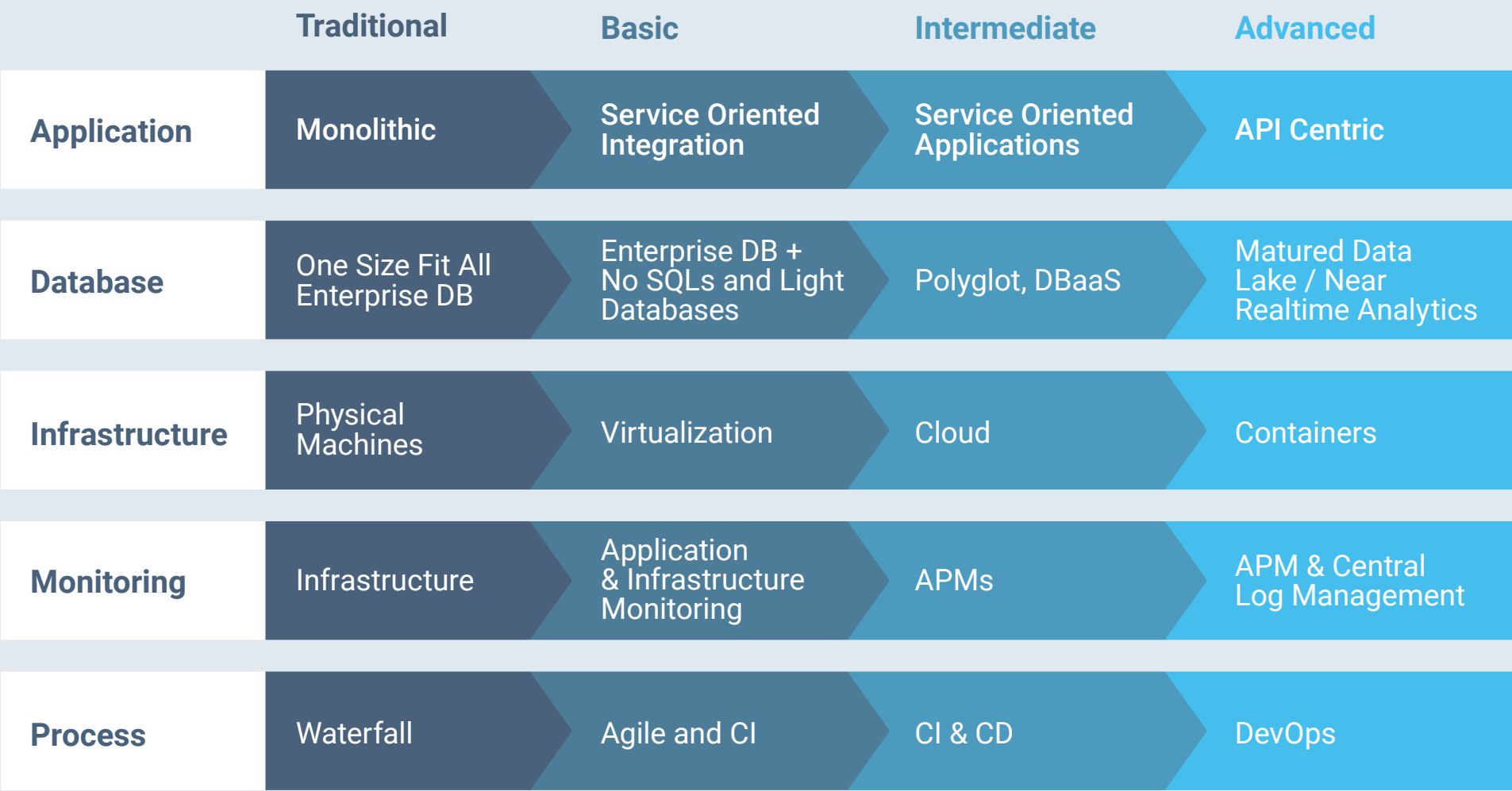
Stateful applications simply don't scale as easily as stateless ones. If memory on one machine is filled with important information, and the machine goes down, that state is lost forever. Complex memory mirroring schemes aren't appropriate for highly scalable applications, either, as they chew up bandwidth and slow down systems.

“The problem is, microservices running in containers have a much greater need for interservice communications than traditional architectures do, and this can introduce problems ranging from poor performance, owing to higher latency, to application-level failures, owing to the loss of data or state. This potential pitfall plagues, to a greater or lesser extent, each of the traditional architectures, which all struggle to scale capacity and/or throughput to handle the relentless growth in the volume and velocity of data,” writes [Paul Curtis](#), a principal solutions engineer at [MapR](#).²⁰

The abstraction of state makes for the easiest starting point for migrating to microservices. Whatever that state may be, it must first be pulled out into the open and handled in another fashion. In a way, this model mirrors that of functional programming: rather than setting a variable to a value, a value is passed through a function, yielding a unique and new result on the other side. At no point do we overwrite the original variable's value.

So, too, is it with stateless microservices: Instead of setting variables internally to represent externally-sourced data, we simply pass that data through the application, as though it were a manufacturing process. The

Microservices Maturity Model



Source: "Spring 5.0 Microservices," 2nd Edition" by Rajesh R V (O'Reilly) © 2018 THE NEW STACK

FIG 3.2: An organization reaches maturity with microservices when it adopts a DevOps process along with the technology.

application outputs a value based on the input, but that input should still be out there somewhere: on a message queue, in a streaming data server like [Flink](#) or [Kafka](#), or even in a caching layer somewhere hosted out on the edge of the network.

This is, perhaps, one of the reasons streaming platforms have been growing in popularity recently: They allow for the abstraction of state from all applications, while still enabling a fast user experience on the other side of the queue. Such streaming platforms allow any containerized microservice to publish data and/or messages in a stream, as well as receive or subscribe to any stream. Scaling an application to handle higher volumes of data is simply a matter of starting up additional containers for whatever microservices are creating a bottleneck.²¹

DevOps teams may still take snapshots of the system, capturing data and development states at a specified point in time, to reuse as blocks of state. By

incorporating these snapshots into a feedback cycle of monitoring and continuous improvement, they can provide the basis for rollback capabilities and a trail of breadcrumbs for later analysis. This method is part of a larger philosophy gaining momentum that we hear described as GitOps, an iteration of DevOps, wherein Git is a single source of truth for the whole system.

Case Study: Gilt's Microservices Migration

The path to microservices is ultimately about services and the resources needed to run an application. That's evident at Gilt, an ecommerce company that is on a path originally developed by [Emerson Loureiro](#), now a software developer at Amazon Web Services (AWS).

The year 2007 seems so long ago, but it was the time of Ruby, a frontend delight for geeks who lived by its libraries and the development of specific, often quite creative, pieces of software. Gilt had a big piece of Ruby monolithic software running its site. They ultimately dismantled the monolith and created a series of Java services, broken into components running on Scala across AWS infrastructure.

In 2016, Loureiro took the stage at AWS re:Invent: "We did realize that a lot of those Java services, they became monoliths themselves. We broke those further down into, now, real microservices, now being written in Scala. We also wrote a ton of new services as well. We broke down our frontend apps into lots of smaller components," Loureiro said.

Today, Gilt is running on a microservices architecture, added Loureiro. Along the way, that microservices journey also yielded a move to Amazon's cloud. When the cloud move was made, the team packaged up legacy applications into virtual machines. Once all of Gilt's applications were packaged up, they were then divided among five departments inside Gilt's IT team, each with its own Amazon account. Each team was responsible for running its specific services, which could be migrated over quickly, as they'd already been

packaged up and readied to go by a central IT force.

Best Practices for a Microservices Migration

The previous chapter discussed how an application architecture can be workflow-oriented and how it can be broken into services accordingly. It's time to look at how teams work together and manage projects over long periods of time.

The Gilt team organized its microservices around specific business initiatives. This meant each business group was responsible for running and maintaining its specific set of microservices. This also meant that the lines of control for, say, the accounts team, were kept inside that team. No outside developers or IT staff members were needed to fix internal accounts problems, speeding up response times for issues, and enabling the team to increase its innovative velocity.

This could also require some overall reorganization of the team, as demands increase for each group running microservices. Will the organization need more resources to maintain service levels? This has to be a first-order question. Each group will need its own frontend, backend and networking developers, though the numbers of each will likely vary from team to team. Teams should be small enough to manage independently, usually between three and five people.

However, each team is not exclusively consuming its own services. The very idea of microservices is to enable everyone in the enterprise to access the business functionality they need at any time. Thus, while individual teams control their own applications and services, they do need to interact with one another in order to consume and offer those services outside.

Some best practices for discovery that help with service sharing among teams were shared as part of Loureiro's discussion with [Derek Chiles](#), now senior

manager of Amazon Web Services' worldwide tech leader team:

1. **Use conventional naming schemes to identify services.** DNS makes this a bit simpler as it provides a straightforward way to make discovery sane. Dynamic discovery is another method of handling this problem, but it does add another layer of complexity to the architecture. Systems like [etcd](#), [HashiCorp's Consul](#), and [Eureka](#), all allow service discovery to be performed dynamically within the architecture. This does add another service and a point of failure to your architecture, but also offers a centralized place to view deployment status and control available instances in a complex system where DNS doesn't cut it. **22**
2. **Adopt API management.** While companies like [Apigee](#) (now part of Google), [MuleSoft](#) and [Oracle](#) will all sell you API gateways, there's a lot more complexity and nuance in the space than simply putting a gateway in place. Some services will inevitably be earning money based on usage; however, API gateways are a great place to implement billing and usage tracking through complex metrics captured by these commercial systems. SLA enforcement is naturally a portion of that as well.

Yet for most teams, starting out with a simpler approach to API management may be the best way to go. The first thing on the list to manage is traffic metering and throttling. API management, at its core, is load balancing with more complex rules. To this end, teams can use open source solutions, such as [NGINX](#), to handle the most basic of API management functionality.

When the migration is further along, monetization of APIs through commercial gateways may actually be needed, but many teams find that existing solutions are up to the task at a far lower cost point. The general rule of thumb, we've seen, has been that if the API directly generates revenue through usage, that's the point where a commercial solution should enter the

picture. For internal and external API traffic that is not generating business revenues, noncommercial solutions can typically handle the problems.

Gateways can get pricey very quickly, as many companies charge based on traffic, so they should be a last resort.

Conclusion

The real draw for microservices is increasing enterprise software development velocity, while lowering costs and reducing complexity. Those are very tall orders on their own, so getting all three benefits while also migrating legacy applications to the cloud can be the sort of “boil the ocean” project that chokes an IT department.

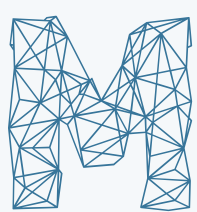
In the end, each of the benefits can be gained individually along the way; they don’t have to all come at once. Just as services don’t have to be migrated at once. Microservices are loosely coupled and self-contained with completely static code. If done right, a microservice can be optimized and left alone indefinitely, never needing updates or maintenance. That’s far easier said than done, however.

Microservices are defined by workflows, and the services used based upon business objectives. The root business model is what matters — less important is what gets built, as long as it can be developed incrementally and run with no or little management.

Microservices thrive when teams employ best practices, find ways to reflect upon their work based upon analytics, and work under the leadership of full-stack engineers. When teams are led by experienced DevOps professionals, the layers of the stack can be utilized as individual services. Adoption can be part of an overall focus, achieved by onboarding teams with recommendations, documentation and best practices.

CHAPTER 04

A Case Study of Questback's Phased Approach to a Microservices Transition



Midway through the transformation of its 18-year-old monolithic application architecture, [Questback](#) is finishing a year-long containerization program and is preparing to begin an accompanying microservices deployment. The online digital survey and market research company is rearchitecting its platform to modernize it, make it easier to add innovations for customers and make it more cost-efficient.

The transition to containers and microservices is a strategy that is gaining popularity among a growing number of enterprises, according to several IT analysts. And a phased approach like Questback's is a good way to make this shift, they said. As the company works toward its modernization goals, it is continuing to run and maintain its existing systems to make the transformation as seamless as possible.

“We are living in a parallel world,” Questback CEO Frank Møllerop said. “We are working with the old stuff as we are preparing the new stuff. All of this could have been done quicker, but we decided to take it a little bit slow at our end to be sure we were not impacting our business-as-usual too much.”

A Two-Phase Approach

Starting over from scratch was out of the question, because too much money had already been invested in the existing system and because it still did its job in most ways, Møllerop said.

“We went into the cloud in 2000 when it was called hosting,” he said, long before the proliferation and success of public clouds. “It was the cradle of our technology.”

The problem was that the old architecture needed modernization and other improvements, as the company looked at offering its products and services through public clouds. In the past, Questback had only offered the services through private clouds.

These needs became more evident, said Møllerop, as the public cloud started to gain major traction and speed in the last six years, with [AWS](#), [Google Cloud](#) and [Microsoft Azure](#) allowing businesses to easily deploy applications that could be used by customers.

After researching its options, Questback's IT and leadership teams reinvigorated the company's IT architecture through the addition of containers and microservices that the company developed while maintaining its effective and long-proven IT system. The initial plans called for a two-phase approach: first for containerization, and then microservices, which would integrate with its existing monolithic architecture.

“That's the whole point,” said Møllerop. “We don't want to replace it because our stuff works, and it can run forever.”

Containerization Completed, On to the Microservices

By April of 2018, the containerization deployment was finishing up, helped

along since the late summer of 2017 by a detailed project plan which included the creation of an internal cloud enablement team, governance procedures and help from IT staffers and a third-party technology partner. Testing is now continuing with the containerization efforts. The containerization phase, which uses Docker, Kubernetes and [Kublr](#) management components, finished on time and on budget.

The second phase, the introduction of microservices, was slated to begin in April 2018, and will include decisions about which microservices technologies to use and which individual service components the company wants to offer up to customers via microservices, said Møllerop.

The lure of microservices is that they will help simplify the complex problems of trying to make changes in large, unwieldy monolithic systems, he said. With some 200 types of technologies and standards included in its existing IT architecture, as well as more than a million lines of code, making changes creates a lot of technical issues, because each modification affects other systems. “It takes a long time before you’re able to introduce new innovations to the market,” Møllerop continued, using such a procedure.

By sectioning off services and processes within the monolithic system using containers and microservices, select customer features will operate individually and be self-sufficient and self-contained, allowing them to be used by customers through public clouds as needed, he said. By using microservices, Questback won’t have to worry about problems arising through changes to its existing architecture, as its modernization and efficiency programs continue. Changes will only happen inside the subsystems of the containers.

“With containerization, we can move things around. It’s much easier and you can control it and have a complete overview,” he said. “It enhances your mobility.”

Customers will be able to access the services through Questback's APIs, allowing the company to earn revenue while simplifying innovation and service delivery, and giving it new go-to-market models.

"This will open up a new commercial model, so we can charge for the use of our services differently than we would have done in the monolithic setup because we can, for instance, charge \$0.01 per API call," said Møllerop.

At the same time, the technology additions can be done even as the existing system continues to support the company's everyday business efforts.

Drawing a Microservices Road Map

To prepare for the microservices phase, Questback will rely on its 100-member IT engineering team, as well as a third party, EastBanc Technologies, which provides its Kublr container management system. Kublr is part of a growing field of Kubernetes service providers, including [Giant Swarm](#), [Heptio](#), [Pivotal](#), [Platform9](#), [Rancher](#), [Red Hat OpenShift](#), [StackPointCloud](#) and [Weaveworks](#), that set up and configure Kubernetes for enterprise users.²³

For the microservices strategy, the plans are in the very early days, he said. "We need to decide, first, which of our individual service components we will make available. It doesn't make any sense to do a huge microservice architecture undertaking if we are not planning on having that element of the service made available. We need to document the APIs well so customers can utilize them and understand how each of the service components will work individually."

There have been a few challenges with the process so far, said Møllerop, including resistance to change from some employees and mapping out how the new technologies can be used to fuel the company's future goals. In addition, as Europe's General Data Protection Regulation (GDPR) privacy rules take effect, Questback must ensure that the changes comply with those stricter rules, he said.

“It’s been a very thoughtful and well-orchestrated process,” Møllerop said of the containerization and microservices projects so far. “It’s not been easy. It’s been a lot of hard work and lots of late hours and some frustration as you go along. But we are now on the home stretch, so it looks very good.”

One key to making it all work was bringing in Kublr and EastBanc to assist with the work and to “give us a new perspective into how we should approach this technology-wise,” he said. “That’s what we did and that was one of the key reasons we were able to finish on time and on budget.”

Analysts Weigh In

Questback’s decision to introduce containers and microservices into its existing monolithic architecture is a strategy being told by other companies as well, said Charles King, principal IT analyst with Pund-IT.

“The migration versus starting-from-scratch issue offers a good point to compare microservices development technologies and procedures with what you already have in place,” King said. “By doing it this way, it also can allow tried and true expectations for how the process proceeds and how the final application performs. Starting from scratch seems to be more common in organizations that are already on the road with application modernization.”

Questback’s move to recognize employee resistance to change amid the projects is wise to keep the efforts on-track, while maintaining worker job satisfaction, King said. The company’s decision to bring in a trusted third-party vendor to help with the work is also a smart move, he said. It is also showing a fresh approach to adapting its existing IT infrastructure to work with microservices efforts and projects.

“You can point to the popularity and continuing rapid growth of core technologies, including Docker, as a proof point for how widespread it’s becoming,” said King.

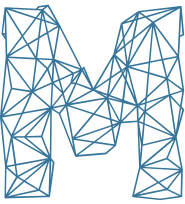
Questback's approach makes sense, said Dan Olds of Gabriel Consulting Group. Ultimately, "the best way to get to a microservice architecture is to first build the application as a monolith and then break it apart into microservices."

At the same time, other companies which are already using a variety of internal technologies and architectures potentially could have trouble migrating to microservices in bulk, he said, based on his discussions in the marketplace.

"Many of the companies that I'm aware of are starting their microservice architectures on new projects and leaving the older stuff behind until they get more experience and better tools for conversion," said Olds. "This is an evolutionary process."

CHAPTER 05

Microservices Security Strategy

icroservices can serve as an elegant way to break the shackles of monolithic architectures when building and deploying applications. But the concept is not new. Microservices have arisen in different architectural styles over the past 20 years. Critics and proponents alike are skeptical of any term that is meant to describe this architectural style in its whole, as it can't encapsulate how much different it is than web services. At first glance it may seem absurd to make the comparison, but the only essential differences are in new abstractions to make the compute, networking and storage a lot easier to use.

Technologists who created microservices in 2007 had to build their own proxies, service meshes and orchestrators. They relied on open APIs to connect services. That knowledge grew in scope over the past ten years and the ecosystem has evolved to the point at which these components are readily available as open source and commercial offerings. Container security technologies are now following the same trajectory to become part of the overall developer experience — joining the list of abstractions that are now increasingly programmable, automated and commercially available.

While this abstraction of the security layer has many benefits to the overall

security posture of the organizations deploying cloud native microservices, it also poses new risks and possible vulnerabilities. With their individual APIs, microservices can be reconfigured and updated separately, without interrupting an application that might rely on many microservices to run. However, microservices also come with many separate APIs and ports per application, thus exponentially increasing the attack surface by presenting numerous doors for intruders to try to access within an application. While their isolated and standalone structure within applications makes them easier to defend, microservices bring unique security challenges. The attack surface widens further for Kubernetes users because of the orchestrator's comprehensive reach in the container runtime environment.²⁴

As a result, organizations have begun to take take a “shift-left” approach that rests security practices deeper into the development process, so that security teams are more involved in engineering and vice versa — an emerging DevOps practice appropriately named DevSecOps. The shift-left approach, which gives developers more responsibility for application security, itself has two sides: Most developers do not fully understand how applications are connected across a network mesh. What they are looking for often are open end-points — APIs to interconnect applications. That in itself creates a security gap and an opportunity to isolate issues. At the same time, it allows teams to detect anomalies faster through automated container image scanning — going further left by baking security into the code itself.

“I often use the analogy: If you were writing “War and Peace,” would you rather edit and course correct as you write, or pile all the corrections on at the end?” writes Mike D. Kail, CTO of Everest, on The New Stack.²⁵ “Security has traditionally been viewed as a barrier to velocity and innovation. But by integrating it seamlessly and continuously into the software development life cycle, it can actually accelerate innovation.”

The security advantages and disadvantages that come with microservices are

still emerging, as are the tools and best practices for securing them. A microservices architecture requires a new approach to security — network security, in particular — and new tools that were built for this purpose. Though services are so closely connected, there needs to be a way to ensure latencies, failed instances and security threats are contained to each service, so that even if one service is taken down, the others perform their function until the affected service is reinstated.²⁶

Organizational changes that put more responsibility in developers' hands must be combined with emerging best practices for securing microservices-based applications. Companies are thinking more deeply about developing separate pipelines for various workload types. Other emerging best practices include:

- Programming languages for maintaining compliance.
- Container image scanning.
- Policy-based network security.
- Canary testing technologies.
- Threat detection at runtime.
- Log analysis.

Containers Secure Microservices

Microservices and container security are sometimes incorrectly referred to interchangeably, even though they are two different things. This may be due, in part, to how most enterprises run microservices on containers. According to an [Enterprise Management Associates study](#), for example, 63 percent of enterprises surveyed run microservices on containers, while an additional 30 percent are planning to do so in the next 12 months.²⁷ However, microservices can just as easily run on a virtual server as in a container.

The trustworthiness of the source for the container that serves as the base for deployment, and the level of access the container has to the host operating system are two distinct container-specific security concerns.²⁸ The focus centers on the maintenance of the container to reduce the attack surface of the application and its corresponding microservices. Emphasis is on keeping container images small, short-lived and well-scanned. Security protocols require managing the containers to prevent infected images from spreading. Such protocols help protect against potential vulnerabilities of microservices within containers.

Container images are the key source of vulnerabilities. Publicly shared images are frequently infected and thus unsafe for production environments, writes Twain Taylor for [Twistlock](#).²⁹ Image scanning features of container registries make it easy to scan all container images downloaded. They run checks for the known vulnerabilities and follow the CIS Docker Security Benchmark.

Importantly, when running containers, they need to be run as unprivileged, non-root containers, according to Twistlock. This prevents vulnerabilities from spreading to other containers and the host. Additionally, steps need to be taken to keep container images as small as possible. This reduces the potential surface area for an attack. Ensuring container lifespans are short — not more than a week — can also keep your system more dynamic and less prone to vulnerabilities.³⁰

A container has an architectural advantage over the traditional application development platforms before it. Container technologies are isolated from the operating system, in contrast to a hypervisor-centric approach such as Platform as a Service (PaaS), which is limited by performance and capacity. They are architected for another age, when updates were events of importance that required pre-scheduling on a calendar.

In comparison, container architectures are essentially platforms for developing

microservices. They are declarative, predictable and lightweight, compared to virtual machines that historically would require configuration mechanisms to identify the software within the machine.

Containers operating in a service mesh are also far easier to secure. The mesh connects the endpoints of the container network infrastructure which operates on policy frameworks. The mesh connects the platform monitoring with the security infrastructure. Monitoring is more granular and can be managed declaratively with little human involvement.

Containers create a machine-learning environment almost by default. They're based on the ongoing learning of the homegrown data loads the system is processing. By using containers to hold the data and apply it to predictive algorithms, network security policies may evolve faster and be used with machine learning to manage updates declaratively.

The ability to measure and compare observed network behavior to a predictive model for a microservice is a key advantage that containerized architectures provide, said Twistlock CTO [John Morello](#). "While it's theoretically possible to create a reference model for traditional, non-containerized services, this is almost impossible to do at scale in practice," Morello said. "Few developers truly understand all the network dependencies in a complex app and fewer still have the time to manually codify these interactions into external systems and maintain those definitions over time as the app evolves. The fact that containers make machine learning about these network flows practical and effective is a fundamental security advantage that enables more tightly defined, application-tailored network policies to be used at scale."

The stateless qualities that can add agility to how microservices are deployed and administered can also be applied to microservices security in container environments, said [Suzy Visvanathan](#), director of product management, for [MapR](#). "Microservices need to be lightweight and execute in a flexible, agile

manner depending on business events and the required context. Microservices also need the ability to share data to drive the operations and analytics required for the specific use case,” said Visvanathan. “The most efficient and secure way to support this is through an underlying data fabric that can support the required data access, while providing the robust security and access controls.”

Containerizing microservices and managing them through [Kubernetes](#) is one such way of providing this, Visvanathan said. “Each microservice is guaranteed secure access, and regardless of where that microservice executes, the access method and security is consistent.”

Containers can serve as an excellent security perimeter for microservices, thanks to their design. “Containers enable you to apply security to each individual service, making them ideal for microservices. And no matter the application, putting it in a container provides an added layer of security,” said [David Lawrence](#), a senior software engineer at [Docker](#). “We see a common trend across enterprises is to containerize legacy applications, and as a result, gain the immediate benefit of hardened security, in addition to cost-efficiencies and portability to hybrid cloud environments.” Wrapping and delivering microservices in containers is an approach for developing a secured environment, but many more steps are needed to guarantee a secure application.

The Microservices Security Checklist

An organization must adjust its own internal security practices for a new, cloud native approach. In its [Application Container Security Guide](#), the National Institute of Standards and Technology analyzes the unique risks posed by containerized applications and advises organizations how to secure them. Their first recommendation: “Tailor the organization’s operational culture and technical processes to support the new way of developing, running, and supporting applications made possible by containers.”

This guidance implies “that modern data centers require a major shift in enterprise strategy and means of securing them, in order to keep pace with the new methodologies of developing and running applications,” writes [Nitzan Niv](#), a system architect at [Alcide](#).³¹

With a DevOps culture and processes in place, organizations are in a better position to take advantage of the new processes and tooling around securing microservices applications.

Considering how the underlying configurations of applications running on microservices are very different than those of monolithic architectures, standard cybersecurity practices will be inadequate. However, it may be a while before security practices catch up to microservices’ deployment on an industry-wide scale. Cloud deployments may serve as an example of why this will likely be the case.

According to analyst firm McKinsey, a full 78 percent of more than 100 firms recently surveyed are not reconfiguring their security tools when migrating to the cloud.³² Similarly, most security standards in place among the enterprises listed in the survey are unsustainable for cloud networks. While data about microservices projects was not provided in the report, it can thus be inferred that at least a relative percentage of firms are not revamping their on-premises security practices and protocols for microservices.

“Where a traditionally monolithic application can be delivered in a large container model, moving an application from a traditional monolithic architecture to microservices requires complete refactoring,” said [Carson Sweet](#), CTO of [CloudPassage](#).

Microservices thus involve a major change in how applications are delivered, deployed and protected, [Rani Osnat](#), vice president, product marketing, for [Aqua Security](#), said. Key security considerations include:

- **Dynamic code delivery and updates:** By automating testing, code must be vetted to ensure that it represents an acceptable risk in terms of vulnerabilities, malware, hard-coded secrets, etc. “The old way of gating a version, stopping to test it for an extended time, etc. will simply not fly,” said Osnat.
- **Tooling:** This is especially critical when deploying microservices applications with a new set of management tools, such as Kubernetes. “There’s a knowledge gap around these tools that leads to mistakes around authentication, authorization, hardening and other best practices,” said Osnat. “We’re talking about basic hygiene here.”
- **Abstraction from a host environment:** Since existing endpoint and server security tools don’t have the required access and granularity to adequately monitor microservices, visibility and control become issues. “Since microservices can be scaled up and down rapidly, and across different types of infrastructures, it’s a challenge to track them,” Osnat said. “An infrastructure-based approach is doomed to fail, and you have to find a way to secure microservices in advance of their deployment.”
- **Networking:** Microservices represent the best of both worlds when seeking to secure networking vulnerabilities, since microservices essentially expose networking deeper inside the application. “The opportunity is that you can secure the application at the microservice level, which would prevent an attack from spreading much sooner [in a much smaller radius] than was ever possible before,” Osnat said. “On the other hand, microservices networking is more complex and dynamic, once again rendering traditional network security tools inadequate.”

One might assume overzealous security practices can create bottlenecks, as the sharing of microservices among applications is needlessly blocked. However, microservices can never be too tight, especially as compliance regulations and

governmental security mandates become stricter, according to [Dave Nielsen](#), head of ecosystem programs at [Redis Labs](#). One example of why this is the case is GDPR, which imposes a much stricter data sharing limitation within the European Union trading block.

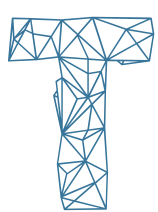
Microservices security, in many ways, represents many unique and especially difficult challenges. DevOps and security teams, for example, must be especially vigilant against unauthenticated access to data and insecure client-side connections leading to man-in-the-middle leakage and attacks, Nielsen said.

“With that said, tight security during the dev/test phase seems like overkill,” Nielsen said. “But in production, security can never be tight enough.”

As in all cloud native technologies, DevOps practices are key to success. By taking a shift-left approach and addressing security throughout the entire application life cycle —providing feedback loops between security teams, developers and operations — organizations will more easily adapt to the new tools and protocols required to keep security tight for a microservices application in production.

CHAPTER 06

Deploying Microservices



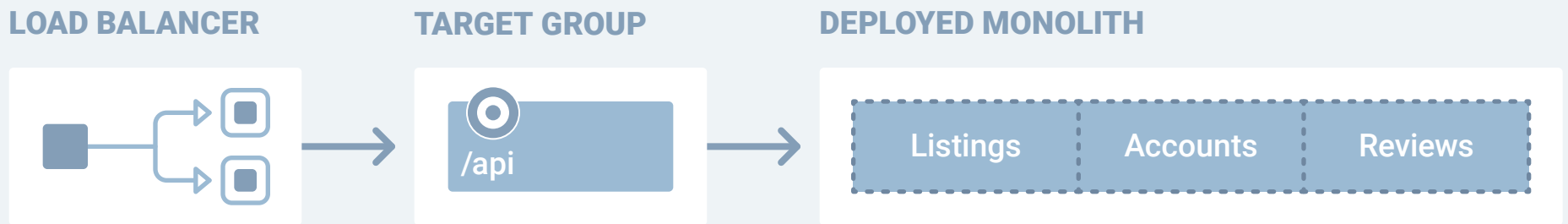
There's a lot to do when deploying microservices into an enterprise software environment. Right from the start there are decisions to make which trickle down from the full data center considerations to the operating systems, to the container management and orchestration layer, and finally to the application itself. Organizations must also weigh their commitments to cloud services that provide tools to help customers build out microservices on proprietary architecture. These services are convenient, but come with a price and may even constrain investments in open source projects that may be longer sustaining. Organizations that balance convenience with long-term health may best be positioned to consider the different approaches to microservices, using open source tools to increase speed and agility.

Hidden within this decision-making strata are nooks and crannies where singular choices can make lasting impacts on performance, application velocity and the actual business value generated. For these reasons, it's worth taking the time to make all of these decisions properly, and from a position where teams are well informed of the constraints and possibilities.

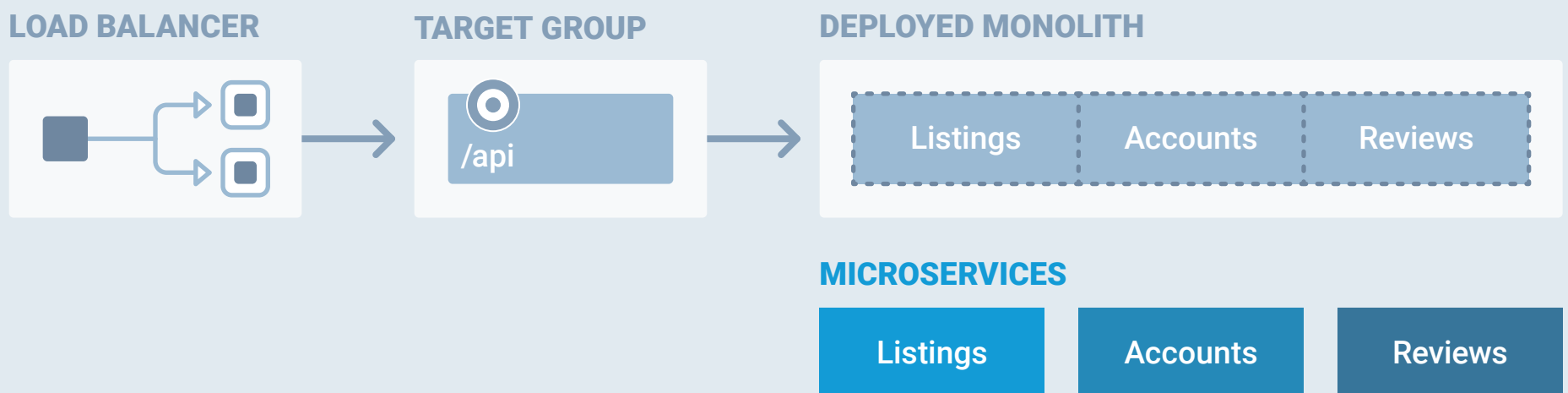
Starting with the infrastructure, the actual location of these deployments is, perhaps, the largest influencer of the other decisions in the stack. Deploying

How to Move From a Monolith to Microservices

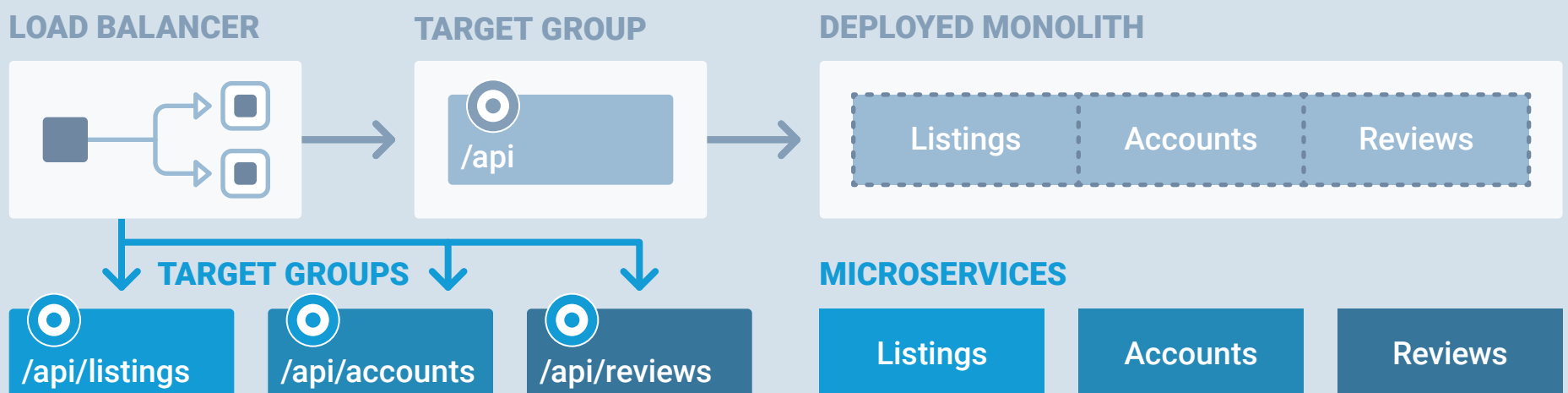
1. Current / Deployed Monolith



2. Start Microservices



3. Configure Target Groups



4. Switch Traffic & Shut Down Monolith

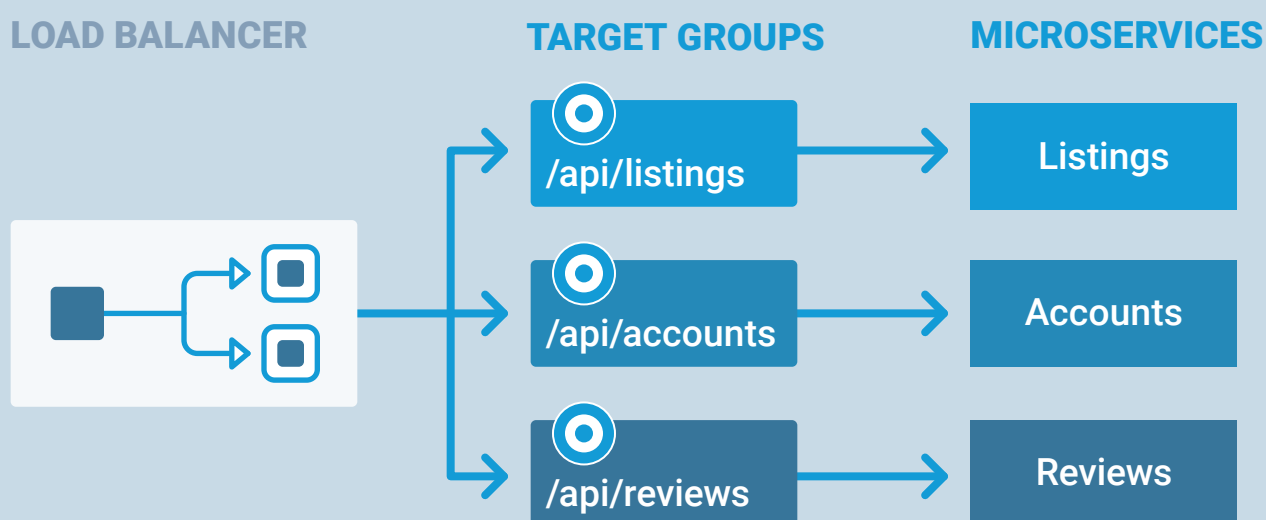


FIG 6.1: The load balancer deployment model allows for traffic to be shifted to individual groups of servers in order to safely transition traffic away from the monolith.

services into Amazon Web Services, Google Cloud or Microsoft Azure each comes with its own deployment choices. For example, Amazon relies on its application load balancer service to bear the weight of the deployment and safely transition traffic away from the monolith.³³ [NGINX](#) also identifies multiple deployment patterns for microservices, including multiple service instances per host, service instance per virtual machine, service instance per container and serverless.³⁴ The load balancer model is a common pattern, repeated in private clouds as well. This model allows for traffic to be shifted to individual groups of servers, thus providing for those groups to be updated on their own, in order. As one group is updated, it takes the place of another that has not, and in this fashion, a rollout is performed.

The rollout deployment models now used are also compatible with the approach to creating subdivided microservices from monolithic applications. For the purposes of decomposition, a distributed architecture allows for better performance when making the transitions to a preferred platform to architect, deploy and manage application components with microservices as the context for further development. By provisioning microservices into groups, they can be brought online as the monolith is removed from the equation, ensuring a smooth transition, and preventing a gap in the data that flows through enterprise services. Zero downtime for upgrades and updates comes when a path is followed encompassing an iterative, DevOps approach.

Deconstructing a monolithic application also comes with benefits for the business as a whole. Aside from increasing agility, an individual microservice can be modified without worrying about redeploying an entire monolith. This pattern also allows for businesses to break out their most essential and difficult application aspects to be replaced with best-in-class products.

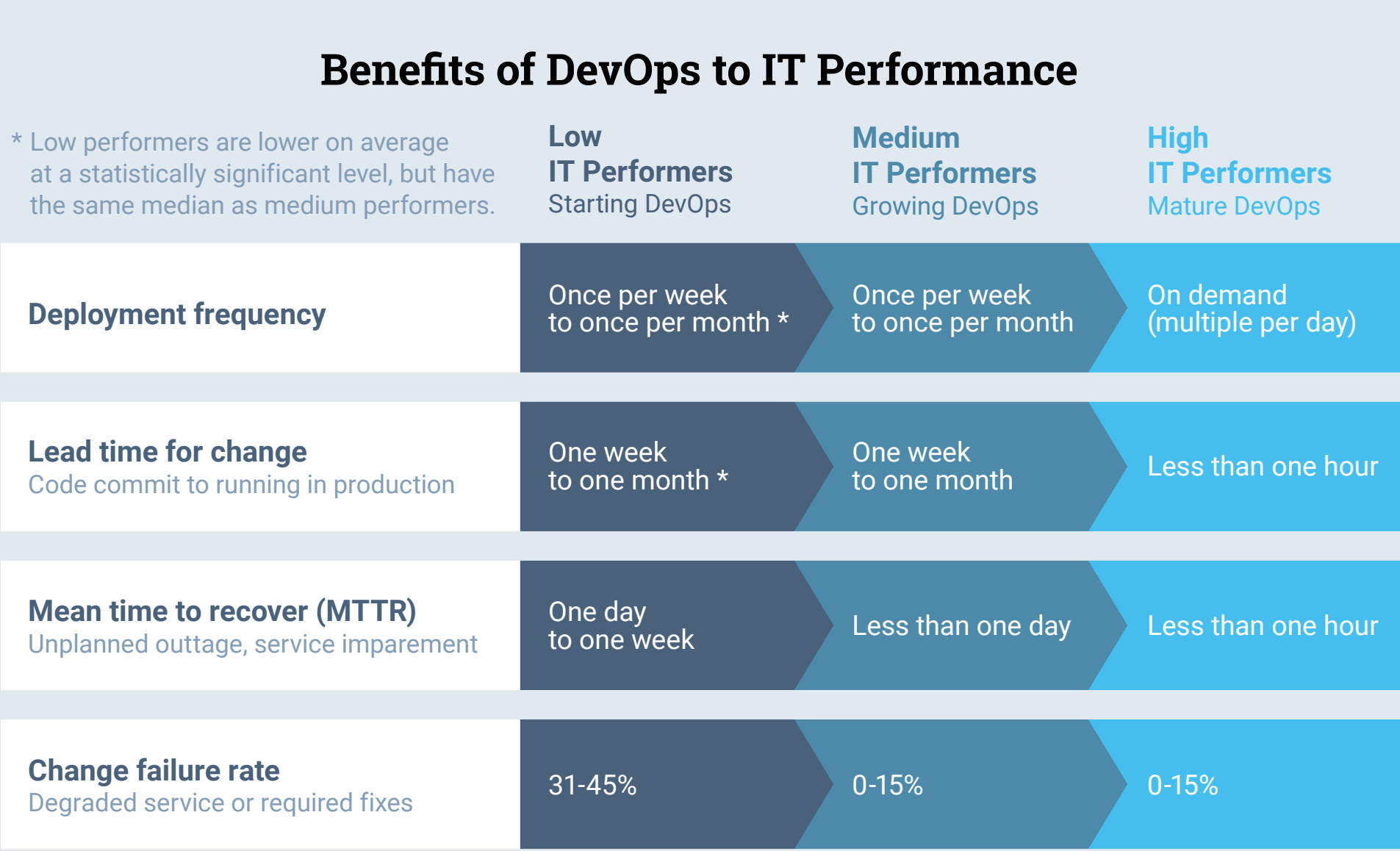
Many monolithic applications already include things like user content storage, payment processing or voice over internet protocol (VOIP). Allowing a team to break these aspects out and turn them into a line item on their budgets —

thanks to [Filestack](#), [Stripe](#) or [Twilio](#) — means more engineering resources can be focused on the actual differentiators for the business application.

[Sameer Kamat](#), CEO of [Filestack](#), said that one of the most important factors of running microservices is maintaining uptime. His company offers APIs to intake and manage user-generated content. “Uptime is a big factor for us. That’s where reliability, redundancy and load balancing is a big factor for us. We use autoscaling because some of our biggest clients have seasonality in their business. We have to scale up and map to that seasonality. ... You cannot rely on hope. Your infrastructure has to be designed in a way that handles scalability in an automated fashion.”

Microservices provide what all API services offer. They enable the agility and efficiencies afforded by the loosely coupled nature of these lightweight services. With fewer moving parts, and dependencies tied to API interfaces

FIG 6.2: Puppet and DORA’s latest State of DevOps Report shows the benefits of DevOps to IT performance are related to deployment frequency, lead time for changes, MTTR and rate of change failure.



instead of OS libraries, microservices can, in theory, be written in any language the developer wishes, and use any environment that's desirable. As these factors will be restricted only to the microservices container, this pattern allows for greater flexibility within the development team.

That's not to say that there are no restrictions on development once the transition to an API-based application is made. Once the API is rolled out, in fact, it cannot change; it can only grow. If original functionality changes, those wild applications written for version 1.0 will stop working, resulting in SLA violations.

"With an API comes a lot of responsibility, making sure it is compatible, making sure it's super simple: Our whole promise to developers is that we will save you time, and provide access across languages," Kamat said.

"Architecturally, we have to make sure we're very aware of any breaking changes to the API. That includes building out microservices for various elements and making sure they can be kept up to date. It has to be a nimble architecture."

Thus, deploying microservices requires a good deal of infrastructure to be in place just for the rollout of those new services. Load balancers, monitoring systems, orchestration and administration systems and security products all must be ready to go before even rolling out service one.

Choosing an Operating System

In days past, the choice of an operating system was generally between Red Hat Enterprise Linux (RHEL) or Windows. The past three years demonstrate how containers are symbolic of an overall shift from machine-oriented infrastructure that runs on-premises software stacks, to a world of services and software that runs across fast, distributed and sophisticated infrastructure. The operating system (OS) will continue its importance, but containers reflect a practice to use the resources available and run them

through services that allow for loosely coupled, proxy-oriented application architectures that abstract the OS. The OS of choice for container architectures is Linux, though Windows has early support. The actual Linux distribution chosen may have wide-ranging effects on performance and maintainability of the microservice.

CentOS and RHEL remain viable choices, but are also fairly large distributions. Red Hat's concession here is [Project Atomic](#), a tiny Linux OS designed to do little more than host containers. [Alpine Linux](#) currently holds the prize of being the smallest popular distribution, but it can have some sharp edges for the inexperienced user and is intended for use inside containers as the base OS on which container images are built. VMware has [Photon OS](#) and Rancher has [RancherOS](#) which are used as host-level operating systems which share their kernel with running containers. Once an OS is chosen, further customization can be had, thanks to the lack of other dependencies within each container OS. Just as microservices can be written in any old programming language, they can also be hosted in just about any environment the team can support. This also allows for the quick testing of new technologies, such as [Nix](#): Each microservice can be an island unto itself, with green fields, or brown overgrowth.

And this is the true promise of microservices deployments at scale: With a well-oiled container construction, testing and orchestration pipeline, the internal minutiae of each application becomes confined. The team building that application will maintain that expert knowledge of its internals and will share that knowledge when needed. But in the end, the goal is to push each service to solidification. Even ossification.

Just as developers once spent months building enterprise services in assembly language to ensure the fastest possible execution on mainframes, microservices leverage deployment pipelines to facilitate rapid refinement, iteration and feedback. This allows the development team to become utterly

obsessed with the minutiae, rather than constantly fretting over external variables. Done properly, a microservice can eventually become provably correct, and beyond further optimization.

Even without perfect internals, the microservices model is built to allow for solidification of the APIs themselves. As with any API, versioning is essential, and new features can be added, but old ones should rarely be subtracted. When a microservice is deployed for the first time, it immediately becomes a dependency somewhere else. This is another reason uptime remains the most important focus for deployments.

Orchestration Platforms

Orchestrating deployments is where enterprises can show their core IT competencies. Administrators and operators should already be chomping at the bit to try out the hottest new tools, like [Kubernetes](#), [Terraform](#), [Rancher](#) and [Spine](#).

Choosing orchestration platforms, however, is more complicated than simply picking Kubernetes and installing it. While this wildly popular open source project has gained many adherents in the past year, it still remains a complex piece of infrastructure software.

The entire cloud-based microservices architecture requires some basic relearning, as well, said [Dave McJannet](#), CEO of [HashiCorp](#). “One thing cloud has done is inspired people with an operating model which is different from the model of the past. It’s characterized by infrastructure on demand and zero trust networks. [It] means investing in security differently, and thinking about networking differently; from physical host networking to service networking.”

While the new model of deployment parallels the old application server model, the infrastructure plays a far more important role than in past systems.

“There’s a parallel for sure. That’s why when we draw the pictures, we have

three elements of the stack: The core infrastructure, the security layer and then runtime on top. That picture hasn't changed for 30 years. The thing is, now at the runtime layer, instead of an application server, perhaps you're using a container orchestration platform, but you still have the other parts of the puzzle," said McJannet.

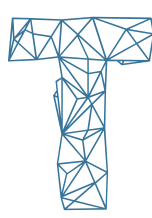
Today, instead of deploying the application server, he said, teams are deploying the entire service with rolling updates and automated scaling support. That's a different go-to-production model than most IT shops are used to, and it requires all of the infrastructure for microservices to be in place before anything can be deployed at all.

To this end, McJannet's company offers [Terraform](#), an environment provisioning tool designed to stand up multiple services at the same time, and to interconnect them. McJannet said he sees many customers using Terraform to provision Kubernetes, as the Kubernetes world expands to include new services like [Istio](#) and [Giant Swarm](#).

This is the type of meta-thinking required to undertake proper microservices deployments. As Carl Sagan said, "If you wish to make an apple pie from scratch, you must first create the universe."

CHAPTER 07

DevOps Practices for Microservices

 The move to microservices typically requires a great deal of automation, and automation implies operations. In today's cloudy, agile world, the science of operations is veering towards good DevOps practices.

Naturally, automation requires code, and operators today have some of the most powerful tools and systems ever created for boiling down the complexity of such environments into more easily programmable bite-sized chunks.

To get there, however, the real grind of microservices is the necessity of having intricately woven systems in place, balancing traffic, monitoring gateways and distributing a security model across the entire application stack before bit one can even be deployed. It's a bit like a chef requiring a properly cleaned, tooled and prepped kitchen in order to prepare a dinner service. This even extends to the waiters and ticketing systems one needs to pass those well-poached plates of salmon to the starving customers.

The skill in DevOps is not being a great chef, but a great manager: managing the waiters, the hot window, the prep chefs and the money, all from a vantage point above the floor, with full visibility of the entire chain of processes, products and people.

In the microservices world, this means it's generally the duty of DevOps engineers to set up all of the infrastructure required to build out at-scale environments: web application servers, registries and repositories, OS and container images, virtualized networking, firewalls, load balancers, message queues and reverse proxies. It's also up to the DevOps team to support new technologies demanded by the development teams: [HTTP/2](#), [gRPC](#) and reliable SSL.

“With the explosion of microservices, you're getting more and more projects. Companies are finding that not only do they have to automate the DevOps life cycle, but it has to be automated out of the box,” said [Sid Sijbrandij](#), CEO of [GitLab](#). “For every project, you have to set up your creation tools, packaging and management. If you have to do that every time, that's going to take a lot of time. The increased complexity of microservices makes it needed to have a closer collaboration between development and operations. It's much more important to have them both on the same page.”

Database Automation for DevOps

One area of a microservices architecture that can become tricky to automate is data.³⁵ From making databases run reliably at scale in Kubernetes, to the sudden proliferation of outside data stores developers can adopt when freed by a microservices architecture, data is a big problem for cloud-based infrastructure, so much so that the earliest days of cloud saw the holistic advice to “expel state from your application.”

Today, we know that stateful and stateless applications can both happily coexist in the cloud, but the actual day-to-day work of managing that data isn't always easy. “What we are seeing is that data is following the same pattern as we've seen on the compute side. As things break into smaller and more logically sized components, the same makes sense on the data side,” says [Georgi Matev](#), head of product at [Kasten](#).

While that sounds good, it's a different thing entirely to move data with agility than it is to move code. Not only are there database design and operational complexities with sharing or dividing data across services, but security and compliance hurdles as well. That's no excuse for not automating the data layer, says [Datical](#) CTO [Robert Reeves](#). "Everyone is getting a lot faster with deploying the application, the compiled bits. But then, they're still relying on manual database changes to support that application. That causes a velocity gap. You've got one part of the house leaving for home on time, while the other side of the house — the database folks — are on suicide watch," said Reeves.

The need for automation, driven by new distributed architectures, including microservices and serverless, is leading many organizations to migrate their databases to the cloud, according to Forrester Research.³⁶ Managed Database as a Service (DBaaS) offerings from cloud providers or platform vendors such as [EnterpriseDB](#), [MongoDB](#) and [Redis Labs](#), provide automated provisioning, backup, scalability and integrated security. About 28 percent of global infrastructure decision makers claim they are already supporting a DBaaS deployment and Forrester predicts that this number will likely double by 2021.

"We need to get the human out of this," said Reeves. "We need to remove intervention. The first thing you need to get over is this idea that we need a human to manually review and execute our SQL scripts," said Reeves. "We're big fans of microservices, but you need to put the power of updating these services and the databases that support the microservices in the hands of the product team."

This is the true realization of DevOps — the team that builds the service, also runs the service. In this way, services remain independent and development moves faster. What's the point of giving teams the ability to push code into production in seconds, Reeves asks, if they then must wait more than a week so that an external services team can run a script on the database? DBaaS gives developers the means to control their service's database and removes the

burden of operations to build and maintain any number and flavor of databases across the whole set of services.

Kubernetes and DevOps

The introduction of Kubernetes as the new standard for container orchestration brings another layer of complexity to the microservices infrastructure. At the same time, when properly configured, it's a means to manage scale on distributed architectures. Kubernetes has transformed the entire DevOps ecosystem — which is ultimately transforming businesses. As Kubernetes adoption has grown, some of the questions around how it fits into existing infrastructure patterns are still being answered. Still, it falls to DevOps to understand and administrate Kubernetes so that developers can have their on-demand databases, pipelines and deployments. Some benefits that Kubernetes brings to DevOps, which The New Stack has covered previously in our ebook "[CI/CD with Kubernetes](#)," include: **37**

- **Independently deployable services:** You can develop applications as a suite of independently deployable, modular services. Infrastructure code can be built with Kubernetes for almost any software stack, so organizations can create repeatable processes that are scalable across many different applications.
- **Deployment frequency:** In the DevOps world, the entire team shares the same business goals and remains accountable for building and running applications that meet expectations. Deploying shorter units of work more frequently minimizes the amount of code you have to sift through to diagnose problems. The speed and simplicity of Kubernetes deployments enables teams to deploy frequent application updates.
- **Resiliency:** A core goal of DevOps teams is to achieve greater system availability through automation. With that in mind, Kubernetes is designed to recover from failure automatically. For example, if an application dies,

Kubernetes will automatically restart it.

- **Usability:** Kubernetes has a well-documented API with a simple, straightforward configuration that offers phenomenal developer UX. Together, DevOps practices and Kubernetes also allow businesses to deliver applications and features into users' hands quickly, which translates into more competitive products and more revenue opportunities.

In addition to improving traditional DevOps processes, along with the speed, efficiency and resiliency commonly recognized as benefits of DevOps, Kubernetes solves new problems that arise with container and microservices-based application architectures. This is one reason why many companies are already building out infrastructure on top of Kubernetes.

As with all the major cloud providers, IBM is focusing on making Kubernetes easier to consume, so DevOps won't have to be monopolized by turning the knobs on such a complex system. The Istio service mesh project, for example, allows DevOps teams to have full control of the data and traffic flows around their microservices. [Daniel Berg](#), [IBM](#) distinguished engineer, container service and microservices, wrote to The New Stack in an email that, "In the open community, [IBM has] worked with other tech leaders such as [Google](#) and [Lyft](#) to build [Istio](#), which equips developers with an orchestration layer on top of their containers to better secure and manage the traffic flowing through them. We've also worked with Google to launch [Grafeas](#), which helps secure the supply chain of code around containers as they're deployed and used."

A New DevOps Mindset

Behind all these services is the need for a unified set of processes. The various teams invested in the microservices inside a company need open lines of communication, and they need to be implemented in a way that cannot be sidestepped or avoided. It's an undertaking, and it's about encoding human behavior into automation, deployment and development pipelines.

As Craig Martin writes in “CI/CD with Kubernetes,” DevOps is a journey and not a destination.³⁸ It means building cross-functional teams with common goals, aligning the organization around the architecture — reversing Conway’s Law — and creating a culture of continuous improvement. One of the higher-level achievements in a DevOps journey is continuous delivery. Here’s how [ThoughtWorks](#) encapsulates the ideal CD mindset: “Continuous Delivery is the natural extension of Continuous Integration, an approach in which teams ensure that every change to the system is releasable, and release any version with the push of a button. Continuous Delivery aims to make releases boring, so that we can deliver frequently and get quick feedback on what users care about.”

CD does not happen “automagically” when databases are automated. The advantages come with how it makes CD easier to put in place. Kubernetes defines a container deployment as managing instances. It is up to the user to develop the microservice and automate deployments and integrations across different environments.

“You have to change around your thinking in terms of how you do application development,” said [Chris Stetson](#), chief architect and senior director of microservices engineering at [NGINX](#). “One of the things that we have been doing a lot of recently has been creating a uniform development and deployment process, where you have your application developers working in a Dockerized version of the application, and doing their coding and testing essentially in that Docker environment, which very closely mimics the environment that we will be deploying to ultimately for our customers. Having that process built out so it’s easy for developers to get started with is incredibly valuable.”

NGINX has implemented an almost ancient, but no less effective, solution to this problem: Makefiles. “We’ve been using Makefiles a lot to encapsulate the more complex Docker compose commands we’ve put together to make a build

target for our frontend developers to be able to do their Webpack frontend development,” Stetson said. “They’re connecting back to all the services they need dynamically reloading the changes they’re working with, and we like using a Makefile because its like declarative Bash scripting essentially.”

No single tool yet offers the perfect solution for managing cloud native applications from build through deployment and continuous delivery. Draft, Helm, Skaffold, Spinnaker and Telepresence are DevOps tools that were purpose-built for cloud native applications. Such tools are transforming the way teams collaborate, placing transparency and observability at the center, while also increasing automation and codifying practices through policy as code.

Automation Makes Microservices Security Practical to Deliver



The microservices philosophy and architectural approach has existed for a while in the form of a service-oriented architecture (SOA). A new set of sophisticated tooling makes this elegant architecture practical to deliver. The number of services, and their ephemeral nature, makes it virtually impossible to secure the environment using the tools and manually-driven processes of the past.

“It really forces you to change the approach that you take for security from human-designed and maintained with a lot of direct manipulation to a much higher degree of automation,” John Morello, CTO of Twistlock, said.

A new breed of security tools can understand and model an application’s typical traffic patterns, develop a reference model that reflects that known good state, and search for anomalies that violate that model. At the same time, new patterns and practices for developers, operations and security teams help integrate that security knowledge from the very beginning of the application development life cycle.

“One of the more important, but less obvious, changes is the general shift in responsibility for finding — but more so for correcting — security vulnerabilities upstream left in the development life cycle,” Morello said.

Morello, who helped author the National Institute of Standards and

Technology's Application Container Security Guide, says it's still possible to follow the same security patterns — with developers passing a build off to operations to scan and produce a bug report. But containers provide an opportunity to deploy and manage applications more efficiently and securely.

“A security team can actually have quality gates in the [dev] process to say if this build contains a critical vulnerability, fail the build and force the developer to fix it right then and there before it ever leaves the dev environment,” he said.

Learn how security considerations change with a microservice architecture, the new patterns and practices teams are following to secure containerized microservices, and how advanced tooling can help automate and streamline security for cloud-native applications. [Listen on SoundCloud.](#)



John Morello is the chief technology officer at [Twistlock](#). As CTO, Morello leads the work with strategic customers and partners, and drives the product roadmap. Prior to Twistlock, Morello was the chief information security officer of Albemarle, a Fortune 500 global chemical company, and spent 14 years at Microsoft.

SECTION 03

MANAGING MICROSERVICES IN PRODUCTION

New patterns and practices are needed to manage the complexity that comes with running microservices at scale.

Contributors



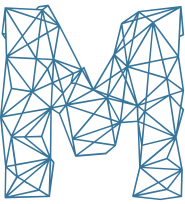
B. Cameron Gain's obsession with computers began when he hacked a Space Invaders console to play all day for 25 cents at the local video arcade in the early 1980s. He then started writing code for very elementary games on the family Commodore 64, and programming in BASIC on the high school PC. He has since become a long-time and steadfast Linux advocate and loves to write about IT and tech. His byline has appeared in Wired, PC World, CIO, Technology Review, Popular Science, and Automotive News.



Todd R. Weiss is a technology journalist who has been covering enterprise IT since 2000. Most recently he was a senior writer for eWEEK.com covering all things mobile. In addition to writing for The New Stack, he has also written for CITEworld, Computerworld, PCWorld, Linux.com and TechTarget.

CHAPTER 08

Microservices Monitoring

icroservices are already in production at companies with a business requirement to run application architectures that are scalable and elastic. The microservices allow the organization to develop application architectures that can run workloads in an efficient and automated manner based upon a host of factors. There is most certainly infrastructure to manage that requires traditional maintenance and operations expertise. The workloads are diverse and require different processes, depending on the technology stack. But traditional methods no longer suffice. Container architectures have changed the dynamic of how applications are managed and monitored.

Containers are pretty much the accepted manner for managing microservice architectures. That's true for hosted services that have adopted Kubernetes and offer services based upon a container infrastructure. It's also true for organizations that use containers to increasingly manage their workloads and adapt to new market conditions.

While these technologies bring the benefits of scale and agility, they also introduce complexity when managing them at scale. For example, the Kubernetes container orchestrator makes it easy for you to deploy an

application that scales across a 1,000-node cluster and deals with deployment and node failures. However, it can be complex to route traffic for that service, to monitor the overall health of the service — not just of individual nodes and pods — and to ensure fair resource allocation between this service and others within the cluster, Liron Levin and John Morello write on The New Stack.³⁹

The space to iterate that comes with containers is now opening new patterns in application monitoring and management that, in turn, create more defined methods for establishing patterns in application-oriented environments.

Kubernetes is serving as a way to use containers for managing state as well as the abstraction to manage the container workloads. New service mesh technologies, for example, when paired with Kubernetes, enable traffic management, service identity, policy enforcement and telemetry for microservices. The advent of artificial intelligence (AI) and machine learning (ML) have also led to new uses for data streaming at the core of the infrastructure, with developers using it to run deep learning frameworks and DevOps teams following these patterns through observability practices. With AI and ML added to the mix, this relatively new and powerful computing architecture will become even more powerful, increasing automation and even more unknown wonders, while also putting many aspects of these complex systems beyond any individual's comprehension.

But as organizations break the shackles of monolithic applications in favor of deployments that run on a collection of microservices packaged inside containers, they quickly learn life can become more complicated than expected. First of all, microservices, which might serve various applications with very different computing needs, need to communicate and they do so over a network. The resulting exponential number of potential interdependencies and errors thus makes managing and monitoring microservices that much harder.

Teams quickly learn that they can't remain completely isolated and deliver the

same amount of business value. Someone has to have the big picture of the application as well.⁴⁰ “Any time you’re digging in complex systems like microservices, it’s really hard to understand what’s going on,” [Matt Chotin](#), senior director of technology evangelism at [AppDynamics](#), said. “And there’s only so much prodding and poking you can do in production before everyone starts getting, quite rightly, super nervous.”

Cloud native monitoring, then, is less about incident response — finding and fixing problems in whack-a-mole fashion — and is instead focused on collecting and analyzing data that allows for automated responses, such as scaling, rollbacks and load balancing. At the same time such observability practices provide insight that feeds back into the development cycle to improve user experience and business outcomes.

Companies such as LinkedIn, Target and Netflix use this cloud native approach, gauging their success by first measuring the quality of the end user experience that is rendered. “When the digital experience becomes the bellwether measurement, their DevOps-savvy site reliability engineering (SRE) teams spend less time chasing misleading alerts, and, instead, focus their efforts on how they can deliver the best experience possible across every touchpoint of customer engagement,” [Kieran Taylor](#), senior director of product marketing at [CA Technologies](#), writes on The New Stack.⁴¹

Microservices Analytics Tools

The obvious solution, as the momentum for massive-scale deployments of microservices builds, is the adoption of advanced microservices analysis tools, as IT organizations wake up to the need to be able to monitor microservices down to every individual module, while also getting a comprehensive, system-wide view.

“People have been focusing on just setting up microservices since their adoption began a couple of years ago. But people are just starting to realize

that complexity becomes an issue when managing hundreds or perhaps thousands of interdependent microservices,” [Jakob Freund](#), co-founder and CEO of [Camunda](#), said. “Now, people are beginning to see on a granular level how they work and want more visibility about how they all play together.”

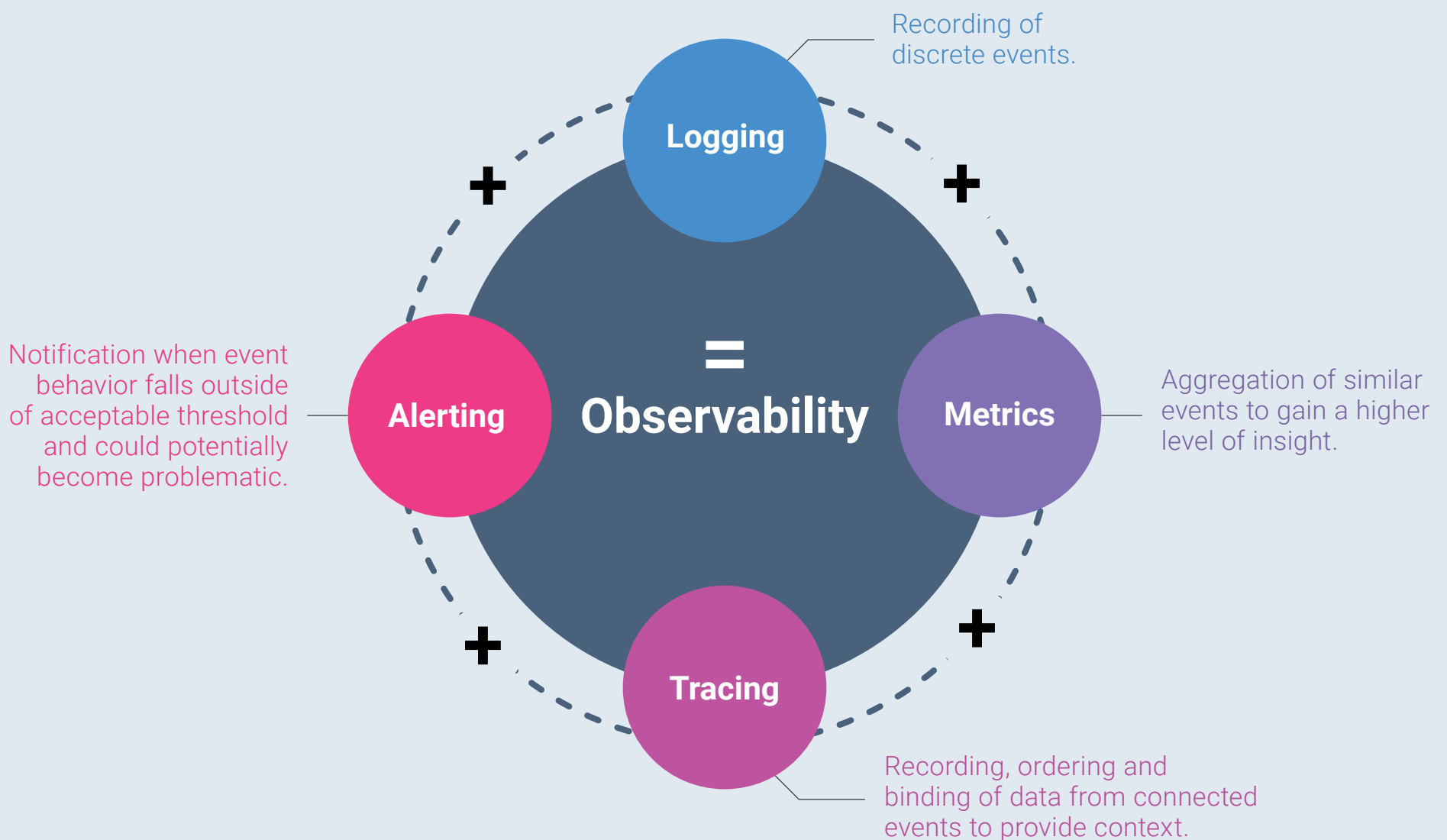
Monitoring has a different meaning today. In the past, developers built the applications. Deployments were done by operations teams who managed the applications in production. The health of services was mainly determined based on customer feedback and hardware metrics such as disk, memory or central processing unit (CPU) usage. In a cloud native environment, developers are increasingly involved in monitoring and operational tasks. Monitoring tools have emerged for developers who use them, to set up their markers and fine tune application-level metrics to suit their interests. This allows them to detect potential performance bottlenecks sooner. [42](#)

Observability — or the ability to understand how a system is behaving by looking at the parameters it exposes through metrics and logs — gives engineers the information they need to adapt systems and application architectures to be more stable and resilient. This provides a feedback loop to developers which allows for fast iteration and adaptation to changing market conditions and customer needs. Without this data and feedback, developers are flying blind and are more likely to break things. With data in their hands, developers can move faster and with more confidence. Our ebook “[CI/CD with Kubernetes](#)” discusses observability and cloud native monitoring in more depth.

Since the advent of large-scale deployments of microservices just a couple of years ago, Netflix, Uber, ING and Amazon have developed their own engines for microservices orchestration and analytics in-house.

Netflix, for example, offered a seminal look at the possibilities that microservices analytics, as well as orchestration, offer when the video

The Four Pillars of Observability



Source: <https://www.weave.works/technologies/monitoring-kubernetes-with-prometheus/>

© 2018 **THE NEW STACK**

FIG 8: Monitoring in a cloud native environment is built around observability, or the ability to understand how a system is behaving by looking at the parameters it exposes through metrics and logs.

streaming company revealed its open source [Conductor platform](#) over a year ago. Netflix continues to rely on the platform to allow millions of users to select, stream and download movies and TV series, as microservices run transparently in the background. Conductor's orchestration engine manages these thousands of microservices with a JSON DSL-based blueprint, which defines execution flow. Conductor enables Netflix developers and administrators to analyze, test, pause and stop, and repair individual models within the different processes, all of which power Netflix's worldwide network.

"As the number of microservices grow and the complexity of the processes increases, getting visibility into these distributed workflows becomes difficult without a central orchestrator," content Netflix engineer [Viren Baraiya](#) wrote

in a blog post.⁴³ “We built Conductor ‘as an orchestration engine’ to address the following requirements, take out the need for boilerplate in apps, and provide a reactive flow.”

Open source analytics and orchestration platforms, such as Apache Spark and Elastic, Logstash and Kibana (the ELK stack), offer a wealth of information and code for organizations that want to take the plunge and develop their own deployments to collect, store and analyze application performance in-house. These deployments offer code for visibility into how individual microservices, which are really separate applications, are performing and communicate with each other, many of which have separate persistent storage interfaces.

The ultimate goal is for processes, errors and bottlenecks to be managed in ways that are totally transparent to end users, as microservices-based platforms fix themselves with the help of microservices analytics. In the event of a bottleneck, for example, an end-use customer who tries to buy a widget or service on the web would ideally never receive an error message that might prompt the user to “try again later.”

Developing microservices orchestrations and associated analytics capabilities are easier said than done in-house. Design and development of homegrown solutions for application and user monitoring require a considerable amount of time and effort. What’s more, the challenge only grows as machine learning and artificial intelligence for IT operations (AIOps) become core components of automated problem remediation. Common performance problems are recognizable, and machine learning means pattern recognition can be employed to automatically detect and remediate issues. However, for that to work, tools must have a library of these performance problems and their remedies.⁴⁴

As a result, organizations that lack resources to develop the analytics tools in-house turn to third parties with solutions and services often built on the

same open source tools. Such commercial monitoring solutions benefit from decades of learning and evolution, but have historically lacked the ability to correlate across silos. That has changed through the adoption of the latest big data and open source technologies that can normalize and correlate analytics to eliminate the traditional silos of monitoring that previously limited insight and control of modern applications.

“Microservices are moving toward mainstream use today and often show many integration points with existing monolithic enterprise applications,” [Torsten Volk](#), an analyst for [Enterprise Management Associates](#) (EMA), said. “Meanwhile, vendors of DevOps-centric application and infrastructure analytics software are stepping up to monitoring this often complex and dynamic world of applications consisting of shared services with often disconnected release schedules.”

To fill in the void for organizations seeking third-party alternatives that offer microservices analytics and monitoring capabilities within [Business Process Model and Notation](#) (BPMN), Oracle’s BPM offering or IBM’s WebSphere Processor Server serve as alternatives for large-scale deployments.

Microservices analytics technologies are in abundance. The market is now witnessing a whole new generation of services, platforms and tools that are largely built on open source platforms such as Prometheus, which recently graduated from the CNCF as a project that has gained acceptance.

Security Dynamics

Companies are stepping up to offer tools to closely track performance and other metrics, but tighter security monitoring for microservices bundled with analytics software are only emerging from companies such as Sysdig. However, third-party vendor tools and services geared for microservices security monitoring do exist as stand-alone services. They might technically fall under the microservices analytics product services category, but are

geared for security.

The monitoring and security challenges associated with microservice architectures arise from how microservices were created to be highly scalable — which means they may replicate themselves across nodes rapidly, run for minutes, and then shut down. Security tools geared for static locations, even virtual ones, will not work. “Additionally, the network becomes how you do dynamic scaling, so network controls must be able to keep up with the changes and have the visibility for intra-host and inter-host communication between microservices,” [Rani Osnat](#), vice president of product marketing for [Aqua Security](#), said.

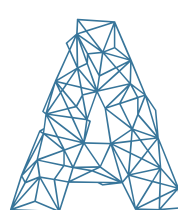
Security, as well as monitoring, will, of course, continue to evolve as microservice deployments and underlying code evolve. For those organizations just looking to get their feet wet, the main concern, in addition to data security, is how to use analytics to track the complexity of the underlying architecture in the near term.

“Organizations are just now adopting microservices and setting up these somewhat complex infrastructures,” Camunda’s Freund said. “The next problem is handling this complexity and that is where big workflow automation [can help].”

In the longer term, expect to see evolving tooling and capabilities around ML-driven anomaly detection, service mesh integration and automated compliance checks, among other features, that will more seamlessly integrate security monitoring into DevOps practices and the infrastructure itself.

CHAPTER 09

Microservices Pricing

 As enterprises update their IT infrastructures, they want to know what it's going to cost to add microservices to the mix. So far, though, figuring out these costs ahead of time has been challenging. Much of this is because the costs depend on what enterprises want to do, how their existing infrastructures are built, and a myriad of other questions which will affect the final costs and estimates.

Yet figuring out the costs of microservices, from integration to operations, is critical to determining whether or not they are worthwhile, as well as figuring out if they should be used to migrate existing applications or to build new ones. Even IT consultants, industry analysts and some microservices companies admit they don't have all the answers at this point.

"I don't think anyone has this down to a hard science with dollar figures," said [Joe Beda](#), the CTO and co-founder of Kubernetes container integration vendor, [Heptio](#).

The key is to know your existing infrastructure and development costs so that you can adequately estimate both the new costs you might incur with a microservices approach, as well as any savings you might realize. Then work

with an architect who has the right knowledge of microservices to devise a rigorous plan and roll it out in small steps that you can test and measure.

Why Microservices Can Be Worth the Pricing Headaches

The lure of microservices is that they can simplify the complex problems of trying to make changes in large, unwieldy monolithic IT systems that are built from a wide assortment of components, technologies and applications. By sectioning off services and processes within the monolithic system using containers and microservices, smaller application components can operate individually and be self-sufficient and self-contained, allowing them to be used without affecting other related code. By using microservices, IT administrators don't have to worry about problems arising through changes to existing architecture, as the changes only happen inside the smaller subsystems, or microservices, set up in the containers.

This is also why it's hard to quantify the costs, said [Chris Priest](#), a senior consultant with U.K.-based cloud consultancy [Amido](#). "It just depends hugely on what you are doing," he said. "The ongoing costs for microservices are far less than it would be for a monolith-based system. I think your costs could go down easily by up to 90 percent less. It depends on how bad it was in the first place."

But the savings don't just come by using container systems such as Kubernetes, said Priest. The savings also come through changes in the culture involving developers.

With a monolithic infrastructure, a company can have 100 developers working on the same code. This can be a nightmare because their changes don't jibe with those of others, adding to complexity and problems, he said. But under a microservices approach, developers can be independently working on different parts of the code, which allows more work to be completed with less overlap

and complication, Priest said.

“Switching the team to a microservices way of work, you can get much more productive,” said Priest. “But again, it’s hard to estimate the savings of such a beneficial move.”

[Robert Starmer](#), a cloud advisor and founding partner of [Kumulus Technologies](#), a Las Vegas-based cloud consultancy, said one place to start when trying to determine microservices costs is to look at microservices as part of DevOps, and then break down those costs to derive estimates. Begin with the company’s development environment, he said.

“If you already have containers, you should have an idea of how your system scales, so that can help with figuring microservices costs,” said Starmer. “We’ve done that with a couple of companies.”

Next, look at the fixed costs of what each developer needs in terms of access to testing and development servers or virtual machines, then add in other related development costs, he said. Create a model based on the scale of the application, how many developers are going to work on the problem and other factors.

Ultimately, though, most IT leaders don’t take these steps. “And that’s part of the reason ... it’s hard to say what a microservices deployment on Kubernetes is going to cost, because of the lack of rigor in going through an application planning process,” Starmer explained.

To help make these calculations easier, Starmer said he is working on developing an application to give these kinds of estimates to IT leaders as they eye the benefits of microservices inside their operations.

The costs for running microservices with Docker is dictated by the size of a company’s server and compute infrastructure, and not based on the number of containers being used, according to Docker. The enterprise platform does not

differentiate in pricing from legacy or microservices containers — it supports both application types.

Good Advice: Start Small

Begin tracking and gauging the costs of microservices, said Heptio's Beda, by starting small and proceeding with an open mind.

“Breaking off one or two services will help [you] understand the trade-offs for that particular application and environment,” he said. “As that is understood, it will become much more reasonable to further invest in microservices.”

It's also smart to avoid creating services that are too micro, Beda added.

“There is overhead for each new service, and that should be taken into account and balanced against the benefits in terms of velocity of decoupling the architecture. Tools, like Kubernetes, can help reduce that overhead and complement a microservices approach.”

Meanwhile, using a container platform rather than just raw virtual machines can also increase efficiency and reduce costs, Beda explained. This is because operations teams tend to overprovision VMs in order to accommodate spikes in load, and containerized workloads are often deployed on infrastructure with per-VM pricing.⁴⁵ A container platform eliminates the need to provision those resources in advance; its resources simply scale with the load. “This isn't microservices, per se, but is often paired with that type of architecture. It isn't unusual to see utilization go from less than 10 percent to less than 50 percent. This can drive huge savings from reduced hardware, data center or cloud usage, as well as reduced administrative costs.”

At the same time, be sure to keep in mind the cost-saving benefits of a microservices-based architecture, which allows developers to move faster, said Beda. “This means faster time to market for applications, happier and more productive developers and improved competitive positioning.”

Ask Vendors for Customer References

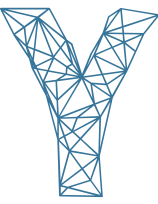
As the process continues, don't forget to seek old-fashioned customer references, said [Rob Enderle](#), principal analyst at [Enderle Group](#).

"The obvious questions would involve locating accounts from the vendor that are like yours and then asking to interview them to learn from their experiences," he said. "In the end, with microservices, the IT shop owns the eventual solution and needs to have both the requisite skill set to build it and the tolerance for the lack of another throat to choke, other than their own, if the project goes south."

Using microservices correctly and effectively comes down to the core skills in the IT group, said Enderle. "If you don't know the questions to ask, you likely don't have the core skills needed to make microservices work and would do far better with complete solutions. It reminds me of folks asking about building their own car. If they have the skills, they know the questions to ask. If they don't, a book of questions won't get them to the skills they need."

CHAPTER 10

Disaster Recovery for Microservices

 You can always spot the hot jobs in technology: they're the ones that didn't exist 10 years ago. Site Reliability Engineers (SREs) did exist a decade ago, but they were mostly inside [Google](#) and a handful of other Silicon Valley innovators. Today, however, the SRE role exists everywhere. From Uber to Goldman Sachs, everyone is now in the business of keeping their sites online and stable.

Microservices and SREs evolved in parallel inside the world's software companies. SREs live and die by their broader perspectives of the system they are maintaining and optimizing. The SRE role combines the skills of a developer with those of a system administrator, producing an employee capable of debugging applications in production environments when things go completely sideways. Some would argue their fuller perspectives about the resources that are managed doesn't provide the more granular perspective that DevOps engineers need to manage individual services. But in an organization and infrastructure as large as Google, it's impossible for an SRE to have a complete view. Still, SREs provide context that a DevOps team working closer to the services may not have.

At scale, when teams are managing hundreds or thousands of services

communicating over a network, it's helpful to remember that 1) the network is not reliable, and 2) failure is inevitable. These assumptions create a whole new approach to site reliability and problem-solving for engineers who are supporting a microservices application, writes Sam Newman in “Building Microservices”.⁴⁶

“Even for those of us not thinking at extreme scale, if we can embrace the possibility of failure we will be better off. For example, if we can handle the failure of a service gracefully, then it follows that we can also do in-place upgrades of a service, as a planned outage is much easier to deal with than an unplanned one.

“We can also spend a bit less of our time trying to stop the inevitable, and a bit more of our time dealing with it gracefully. I’m amazed at how many organizations put processes and controls in place to try to stop failure from occurring, but put little to no thought into actually making it easier to recover from failure in the first place.”

As Google engineers essentially invented the SRE role, the company offers a great deal of insight into how they manage systems that handle up to [100 billion requests a day](#).

“The initial step is taking seriously that reliability and manageability are important. People I talk to are spending a lot of time thinking about features and velocity, but they don’t spend time thinking about reliability as a feature.”

— [Todd Underwood](#), site reliability engineering director at [Google](#).

Top Considerations for SREs

Reliability and availability should be considered at every level of a project. As an example, Underwood cites the way Gmail fails by dropping back to a bare

HTML experience, rather than by halting altogether. “I’ll take the ugly HTML [version], but I can read my email. Availability is a feature and the most important feature. If you’re not available, you don’t have users to evaluate your other characteristics. Organizations need to choose to prioritize reliability.”

Underwood stipulated that every organization is different and that some of the issues Google encounters are not typical. But he did advocate for some more holistic practices.

“For distributed applications, we’re running some kind of [Paxos](#) consistent system. [Google’s book on SREs] has a whole chapter on distributed consensus.⁴⁷ It seems like a computer science, nerdy thing, but really if you want to have processes and know which ones are where, it’s not possible without Paxos in place,” said Underwood.

Paxos is the algorithm for distributed consensus gathering, often used to work out inconsistencies that can arise in distributed systems in which multiple copies of the same software are running on different machines. Paxos allows for a consistent view of the state of the system, despite failures in individual components.⁴⁸

Underwood highlights another aspect of the SRE job that is essential: visibility. When microservices are throwing billions of packets across constantly changing ecosystems of cloud-based servers, containers and databases, finding out what went wrong where is essential to troubleshooting any type of problem. The second most cited challenge for microservices in production — beyond increased operational challenges with each additional service — is identifying the root cause of performance issues, with 56 percent of microservices adopters surveyed citing this as a concern, according to Dimensional Research and LightStep.⁴⁹ Furthermore, when asked to compare the difficulty of troubleshooting different environments, 73 percent reported microservices were harder, while only 21 percent said they were easier than

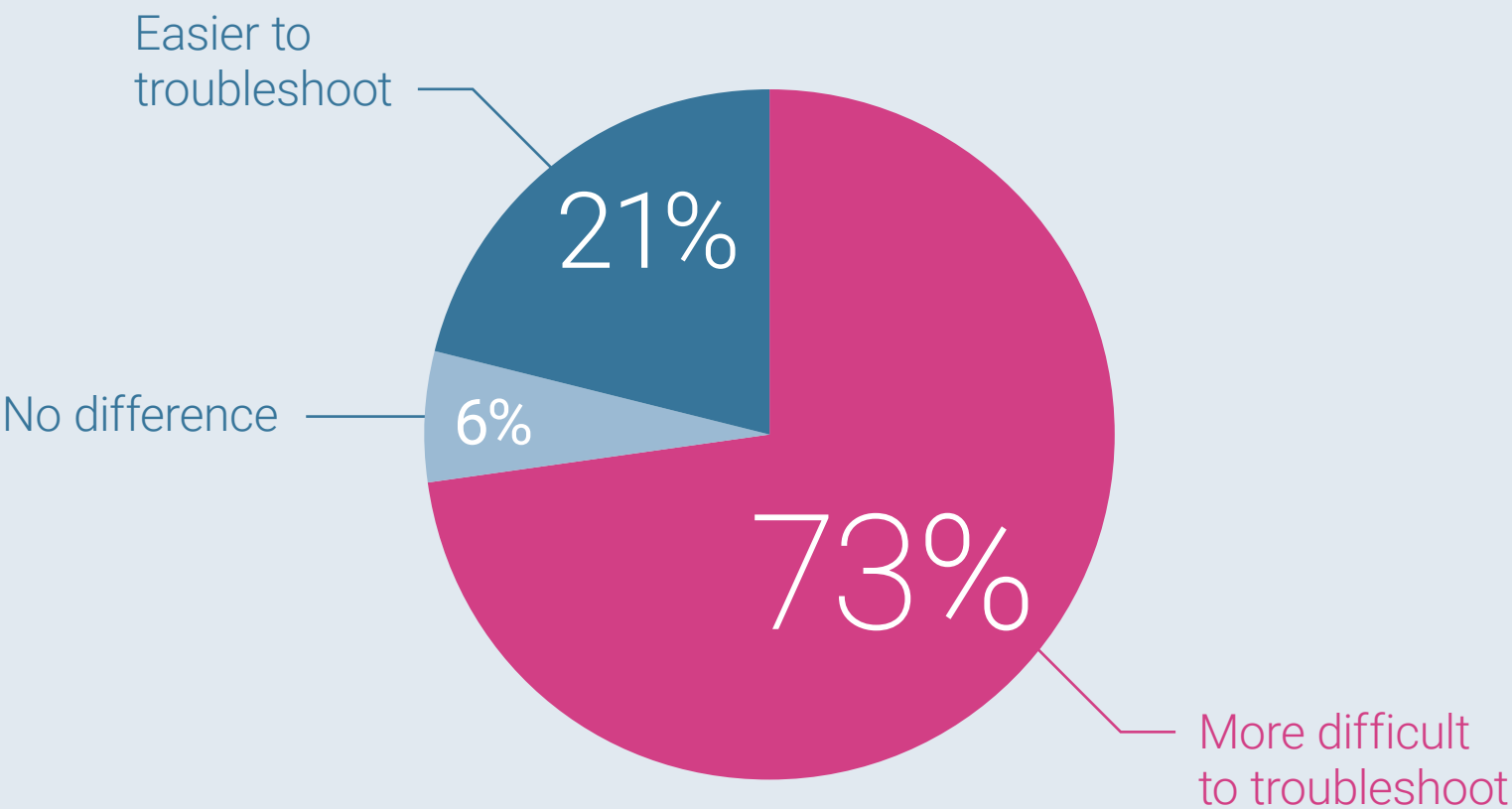
monoliths. This is where the full-stack aspects of an SRE’s job come into play. Google recently introduced a number of tools just for this type of work.

“If you have 100 containers, things like doing a stack trace on a monolith become very difficult. You need a distributed trace,” said [Morgan McLean](#), product manager on Google Cloud Platform.

To remedy this, open source tools, such as the Cloud Native Computing Foundation’s [Jaeger](#) and [OpenTracing](#) projects, as well as a host of products from companies such as [AppDynamics](#), [Dynatrace](#), [Google](#), [LightStep](#) and [New Relic](#), offer distributed tracing capabilities. Google’s own recently released tools include [Stackdriver Trace](#), [Stackdriver Debugger](#), and [Stackdriver Profiler](#). And there’s a reason these tools sound like old-school testing and operations tools from traditional enterprise vendors: They perform the more traditional troubleshooting tasks developers and operations people are used

FIG 10: *Microservices make it harder for teams to track down the cause of application performance problems.*

Adopters Say Microservices are More Difficult to Troubleshoot Than a Monolith



to, but with a focus on microservices and performing these duties in the cloud.

Stackdriver Profiler is in beta, but allows for direct CPU utilization monitoring on applications running inside of a cloud, while Stackdriver Debugger offers a way to essentially insert breakpoints into cloud-based, microservices-based applications. And Stackdriver Trace offers the full-stack tracing capabilities McLean alluded to.

“This is really powerful for general performance improvements and powerful for cost reduction,” said McLean of Stacktrace Profiler. “Snapchat tried it out, and within a day of collecting data they realized a very small piece of code — I think it was a regular expression — which should not have even been showing up in Profiler, was actually consuming a fairly large amount of CPU. This could happen to anyone. It happens to Google. The Snapchat demonstration was just a really great demonstration of the power of this profiling technology.

“Without tools like this, this generally isn’t possible,” said McLean.

Monitoring is essential to microservices management. While a service mesh can help manage the application components, observability is critical to understanding how the services are behaving. Such observability is what creates the ability to develop management patterns that are resilient and manageable. It also provides feedback loops that enable developers to iterate and improve based on data and results.

Chaos engineering is another emerging area which allows SREs to design resilience into their distributed applications by breaking them over and over again with software specially designed for the purpose.⁵⁰ According to co-founder and CTO of [ChaosIQ](#) Sylvain Hellegouarch, chaos engineering allows SREs to “ask questions about your system’s behavior under certain conditions, and [enables] you to safely try it out live so that you can, collectively with your team, see if there is a real weakness and learn what the

right response should be.” [51 Gremlin](#)’s Failure as a Service and Litmus are examples of a new class of cloud native tools to help SREs manage microservices.

New Thinking

The focus on new style tooling is shared by [Matt Chotin](#), senior director of technical evangelism at [AppDynamics](#). He said that teams need to rethink the way they determine the health of entire applications, once they’ve been moved from monolith to microservice.

“You have a myriad of systems. The joy of microservices is that you get to pick the stack that’s right for a particular piece. Each thing might have its own way of monitoring, its own metrics, etc. To understand the health of your entire application and see how a transaction is going to flow through all these different microservices, you have to have a system that is going to help you navigate that. You want something that is going to think in terms of the transaction,” Chotin said.

The engineer shouldn’t think in terms of whether the service is up or down, Chotin said. “Your DevOps team cares about looking at a service to know general availability, but as far as whether or not you are serving the business correctly, you need monitoring that can traverse the entire ecosystem, from application code to infrastructure code,” said Chotin.

New tooling isn’t always the answer, however. New patterns in site reliability engineering are emerging to complement the tooling and inspire new features and functionalities. These patterns embrace the cloud native mindset that failure is inevitable, and are thus designed to help ensure safety and reliability in spite of failure — and not as a means to avoid it. Such patterns include putting timeouts on all out-of-process calls and a default timeout on everything; instituting circuit breakers for failed requests; bulkheads for isolating failure and designing services for isolation; and multiple scaling, load

balancing, performance testing and service discovery patterns.⁵²

Integration testing, for example, becomes uniquely challenging in a microservices environment because dependencies aren't always known or available for testing until a service reaches production.⁵³ Due to the complexity of all the connectivity and dependencies in a cloud-based setting, testers cannot build their own on-premises testing systems to be as complete as production systems. Stricter data regulations under GDPR also make it more difficult to test with real data.

[Alex Martins](#), CTO for continuous testing at [CA Technologies](#), thinks this testing problem will be solved by management and process rather than tools. Modern companies don't look at production as a big event where they cut things over and another team handles it and really doesn't understand what the code does. Instead, DevOps and continuous delivery practices provide a much more holistic approach to application deployment and management.

"I don't think we need more tooling. We need a different way to manage production," Martins said. "It's really more of a mindset change, and a process and culture change. The tools are out there today [for] building environments into cloud. Leveraging containers and proper orchestration is a start, but it takes a lot of effort to make it happen and to make it get to any sort of ROI, but it's possible. The technology allows you to do that today," said Martins.

[Matt Swann](#), CTO at [StubHub](#), said that testing in production is part of his company's daily routine. When moving teams from "standard testing" to new methodologies for testing in production he suggests:

- **Clear communication:** Help teams understand the new process, its benefits and their role in it.
- **Start small and collect some wins:** Start investing time, training and money in one or two teams. Gather details of what worked and what didn't

work, the wins and fails, and adapt the process to fit the organization.

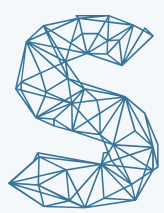
- **Scale:** As your initial teams begin to thrive, roll out the changes to other teams.

This is good advice, whether you're rolling out new processes and practices, or trying out new tools. Communication and collaboration are at the heart of DevOps, and the foundation for any successful microservices migration.

Microservices are still a nascent technology with a fast evolving ecosystem of tools and patterns for implementation. But it is certain that the tools are built for people and teams familiar with DevOps practices. The services, platforms and developer tools reflect the practices of developer teams; the experience of the people on the team, workflows and business objectives. The best approach for one organization may be the absolute wrong way for another. By starting small, iterating and, above all, evaluating and adapting best practices and patterns for its own needs, an organization may make a successful transition to a microservices architecture that is performant, resilient and secure and allows teams to move and innovate faster.

CHAPTER 11

A Case Study of How WeatherBug Uses Microservices Without Containers



Since 2015, weather forecasting service [WeatherBug](#) has been using microservices in its IT infrastructure to dramatically simplify and transform its old, large, home-built monolithic applications by turning their components into lighter, more manageable systems.

But instead of running microservices the common way — inside of containers — WeatherBug has been taking an innovative approach and running them directly on Amazon Web Service’s Elastic Compute Cloud (Amazon EC2) platform without containers, simplifying its deployment even more. It may have been an unusual road, but it has turned out to be the correct path for the company and for the IT problems he and his staff were trying to correct, said [Edward Dingels](#), senior vice president of engineering for WeatherBug.

WeatherBug, which has been around since 1993, was built on large, database-driven applications dating back to the company’s founding, Dingels said. That meant that many of them were unwieldy and massive and dated back to before mobile phones and applications were ever envisioned. The old monolithic applications made it difficult to update or modify them quickly, which had become a necessity in the world of mobile devices and the many applications consumers want to use on them.

By 2015, it was clear that changes were needed, so the company began exploring alternatives.

“We were finding that doing a new deployment was hard, so a lot of our rationale [for a new architecture] came down to increasing the speed and the pace of innovation,” Dingels said. When they first looked at microservices, it became clear that they could be used to decouple the development processes of different development teams, so the needed applications and services could be scaled independently and operate more autonomously.

That was the beauty of microservices — they break apart monolithic applications in a way that allows development to be more efficient, and deliver code into production at a much faster rate. That’s also when the idea of using EC2 arose.

Building Directly on EC2

[EC2](#) is an environment that lets companies use microservices without necessarily having to add the extra overhead of a container platform such as Docker, Dingels said.

“If you look at AWS, there’s a number of different ways you can run compute,” he said. With EC2, you have a host and a host environment that can run pretty much anything, even microservices. That structure is a virtual host, which is just like a container but without the formal container.

For WeatherBug, microservices help connect the various mobile and desktop applications it offers to new features, configurations, code and changes that happen on the fly within the company’s infrastructure. That includes using microservices to help applications get real-time information on a user’s location to enable accurate weather details.

“Each one of these is independently-running services, independent pieces of code that are running on top of EC2 and they’re not connected into a single

service,” Dingels said. “What’s key here is we’re breaking apart the functions that our services do into the smallest possible elements that we can independently deploy and iterate on.”

They could have done the migration using a container platform, but they were happy to avoid the extra complexity. Using EC2, each of the microservices ends up in its own auto-scaling group and can be managed independently from a scale perspective.

“Now you might say we’re not taking advantage of Kubernetes and Docker containers and the many advantages to using those,” said Dingels. “They are definitely on our roadmap, but I don’t believe you need to do them to use microservices.”

If someone says that containers must be used to run microservices, Dingels said, “I would argue that they’ve limited themselves. The whole idea of microservices is that you don’t limit your technology choices.”

By breaking up applications into smaller components with microservices, it allows those components to be easily verified and deployed elsewhere with less disruption to other code.

Immediate Results

After the first few weeks of the original microservices discussions in early 2015, it became clear to the WeatherBug team that it was the right strategy to use to dramatically overhaul their development processes.

“It almost immediately made sense,” said Dingels. “It wasn’t like there was a large amount of convincing to do. Everybody was feeling the problem, which was really around deployment and regression” and how much time it took doing those things with the monolithic applications from the past. “What we found was that breaking apart the services and deploying them on AWS made perfect sense and that it worked very well without necessarily putting in

Docker, which was the only thing available at the time.”

Significant improvements have been seen inside WeatherBug since the move to microservices began, including faster release cadences — multiple times per week compared to months — as well as increasing code stability and reducing troubleshooting time and QA regression time, according to Dingels. “We did find that regression can be done extremely quickly when you’re dealing with microservices.”

Today, about 99 percent of the old monolithic applications inside the company have been transformed using microservices as part of an ongoing process, and for several years, all new coding has been done using them as well. “There are some remnants of those monoliths that still exist, legacy pieces, but it just hasn’t made sense to go back in and break them apart yet,” said Dingels.

It’s almost impossible to determine how much the switch to microservices has cost WeatherBug, since the work has been progressive rather than starting and ending on specific dates, Dingels said. “We were kind of doing the work as the plane was flying.”

Challenges and Ways to Improve

There were some challenges along the way, including some they expected and some that were unexpected. First, dependencies became harder to map because the code was being separated, but that could be resolved using new tools such as [X-Ray from AWS](#), which automates the process, said Dingels. In addition, the log files that were helpful to find problems in the monoliths were not effective with microservices due to how the code was being segmented. But other tools, including the [ELK stack](#), could help resolve those problems.

Yet despite its container-free approach so far, Dingels said that WeatherBug could one day also end up deploying Docker or Kubernetes or another container

management system, because there are some potential benefits to those approaches as well.

For some uses, a container management system could increase the company's IT capabilities even more by providing better utilization of its underlying infrastructure by introducing containers. Another benefit is that containers would make it easier to quickly scale when using Amazon EC2, he said.

"Getting new compute resources takes more time running on EC2 alone than it does using containers," said Dingels. "Now you can imagine that with weather we see a pretty wide variation in usage, especially depending on where our users are located."

"I would say it's on the roadmap," he said of a future container system deployment as well. "One of the things we have found is that it really comes back to where you're going to get the most bang for your buck."

When Breaking Up a Monolith, Consider Data as Much as Code



One of the most challenging aspects of migrating to a microservice architecture is deciding which pieces of the monolith to break off first into separate services.

Architects must understand the business objectives, how the services function and how they will relate to the rest of the whole. Then, the team faces more challenges once services are up and running. Operational complexity increases because each service may have its own languages, toolsets and infrastructure.

To help solve these and other problems that arise when moving to microservices, a new set of commercial monitoring solutions, such as Dynatrace, have emerged on top of existing open source tools like Prometheus. By pulling in data from multiple sources via an API, such monitoring tools can provide a complete view of an application, from its traffic patterns on the network and the database statements it executes, to which container, platform and host a service runs on.

“Once you install Dynatrace, we’re monitoring and tracking every single service, process, host, log, and also end user,” said Andreas Grabner, DevOps activist, Dynatrace. “We have a complete live dependency map from your complete environment.”

This comprehensive view can help architects determine how best to further break apart the codebase, for example. It can also help identify the technical root cause of a particular problem and provide additional context, such as how many users are impacted and in what regions to

help remediate the problem faster, Grabner said.

“Commercial offerings on top of open source tools give you confidence that these tools will last, and are not just supported by a small community that may no longer exist in a year,” Grabner said. “They also give you the best practices to support the services.”

One emerging best practice for breaking up a monolith into microservices that Dynatrace has observed among its customers is to give each service its own data store. If an application was supported by a big monolithic relational database, then an organization must first decide how to extract the data that is relevant for that service into its own data store.

“Sometimes when people talk about breaking the monolith into services, they only think about the code,” Grabner said. “But, it’s important also where the data lives that this service is depending on. You have to treat your current data store just as another monolith.”

Learn about use cases for cloud native monitoring tools, best practices for breaking down the database monolith, and how to tap into old and new systems to get the information teams need to make business-critical changes to an application. [Listen on SoundCloud.](#)



Andreas Grabner has 20+ years of experience as a software developer, tester and architect and is an advocate for high-performing cloud-scale applications. He is a regular contributor to the DevOps community, a frequent speaker at technology conferences and regularly publishes articles on blog.dynatrace.com.

SECTION 04

BIBLIOGRAPHY

The list of source materials for this ebook is a good starting point to gain a new perspective or to dig deeper into how to build, deploy and manage cloud native microservices.

Bibliography

1. [Global Microservices Trends](#)

by [Dimensional Research](#) and [LightStep](#), April 2018.

A report on the challenges of performance monitoring in microservices environments, including insights on how to address these challenges.

2. [The DZone Guide to Microservices: Breaking Down the Monolith](#)

by [DZone](#), 2017.

A detailed analysis of microservices and how they are built and deployed.

3. [Philipp Strube](#), Director of Technology for [Container Solutions](#) in a media and analyst briefing at KubeCon + CloudNativeCon Europe, Copenhagen, May 2018.



After his law studies at the University of Bonn in Germany, Strube has spent over a decade as a serial entrepreneur, to include founding cloud-control and Kubestack.com.

4. See #2

5. [Containers and Microservices: Two Peas in a DevOps Pod](#)

by [Matt Chotin](#), senior director of technical evangelism at [AppDynamics](#), The New Stack, March 16, 2018.



Chotin describes how containers and microservices offer synergies when the right monitoring and security management tools are adopted.

6. [Microservices: From Design to Deployment](#)

by [NGINX](#), 2016.

A guide that describes microservices deployments in detail and how the architectural style can improve applications' speed, flexibility and stability.

7. See #1

8. [How to Build and Scale with Microservices](#)

by [AppDynamics](#), 2016.

A how-to ebook that describes what microservice architectures consist of, with specific instructions on how to build and deploy them to scale.

9. [Red Hat's Chief Architect of Cloud Development Talks Traffic Management](#)

The New Stack Makers podcast, with [Christian Posta](#), chief architect for cloud development at [Red Hat](#), July 30, 2018.



The next big problem Kubernetes adopters will face once they've gotten their systems containerized is traffic management. The open source Istio project is the planned system for handling it all.

10. [Bridget Kromhout on How Microservices Affect Managing People](#)

The New Stack Makers podcast with [Bridget Kromhout](#), principal cloud developer advocate at [Microsoft](#), May 2, 2018.



Many traditional management styles and principles hardly apply to engineers who build, deploy and manage microservice architectures. A new collaborative culture of communication and responsibility must emerge for successful deployments.

11. See #10

12. [Five Things to Know Before Adopting Microservice and Container Architectures](#)

by [Jonathan Owens](#), site reliability engineer at [New Relic](#), The New Stack, April 5, 2018.



A discussion of what microservices and container deployments involve, based on Owens' deployment-management experiences.

13. [What is a Full Stack Developer?](#)

by [Laurence Gellert](#), software developer at [Launch Gate](#), Laurence Gellert's blog, August 1, 2012.



This post describes qualities successful full stack developers should have, including expertise in multiple software layers and programming languages, passion and curiosity about software technology in general and a mindset that embraces cross-team collaborations.

14. [Tackling Operational Serverless and Cross-Cloud Compatibility](#)

The New Stack Analysts podcast with [Dr. Donna Malayeri](#), product manager at [Pulumi](#), June 21, 2018.



Dr. Malayeri addresses concerns and challenges with serverless deployments and the importance of multicloud architectures for redundancy and agility.

15. [Ben Sigelman](#), co-founder at [LightStep](#)

in a media and analyst briefing at KubeCon + CloudNativeCon Europe, Copenhagen, May 2018.



A former senior staff software engineer at Google, Sigelman is the cofounder and CEO of LightStep, which offers monitoring solutions for software running on web, mobile, monolithic and microservices platforms.

SPONSOR RESOURCE

- [Cloud-Native Application Performance Monitoring Requires a New Approach](#)

by [Scott Kelly](#), product marketer at [Dynatrace](#), Dec. 11, 2017.



Learn why APM is essential for dealing with the complexity of a microservices architecture and containers and how monitoring differs for cloud native applications.

16. Pattern: Database Per Service

by [Chris Richardson](#), CEO and founder at [Eventuate](#), Microservices.io, 2017.



This article outlines database structures for microservices architectures in a schematic and detailed way.

17. What is a Data Lake?

by [Amazon Web Services](#), 2018.

A description of how a data lake serves as a data repository for all data, both structured and unstructured.

18. “Cloud-Native Application Patterns” in “CI/CD With Kubernetes”

by [Janakiram MSV](#), principal analyst at [Janakiram & Associates](#), The New Stack, June 2018.



In this chapter of the ebook, Janakiram MSV describes how DevOps sets policies that determine how Kubernetes manages resources, and offers details about cloud native patterns.

19. Where PaaS, Containers and Serverless Stand in a Multi-Platform World

by [Cloud Foundry Foundation](#) with [ClearPath Strategies](#) and [Pivotal](#), June 2018.

Based on the responses of 600 IT decision makers, this report covers trends affecting multiplatform deployments of cloud native architectures.

20. Why Microservices Running in Containers Need a Streaming Platform

by [Paul Curtis](#), principal solutions engineer at [MapR](#), The New Stack, December 20, 2017.



This article shows how a streaming platform can solve common issues experienced in the data pipeline when running microservices in containers.

21. See #20

22. [Service Discovery: 6 questions to 4 experts](#)

by [HighOps](#), May 7, 2015.

An article that outlines how leaders in the field define the key aspects and benefits of service discovery.

23. [Kublr Offers Kubernetes for the Enterprise](#)

by [Susan Hall](#), The New Stack, June 8, 2017.



Kubernetes service providers all seem to have their own niche, such as application management on top of Kubernetes. Kublr is focused on the needs of enterprises.

24. [“Kubernetes Security Patterns” in “Kubernetes Deployment & Security Patterns”](#)

by [Dr. Chenxi Wang](#), managing general partner at [Rain Capital Management](#), The New Stack, February 2018.



This chapter outlines best practices for security management of Kubernetes platforms by covering login privileges, user authentication, container isolation, compromised logins and best practices and policies.

25. [DevOps and Security: How to Overcome Cultural Challenges and Transform to True DevSecOps](#)

by [Mike D. Kail](#), an independent technical advisor, The New Stack, January 22, 2018.



DevOps and DevSecOps do not mean much if organizations fail to ensure they have fostered a culture conducive to the creation of secure code early in the software production pipeline.

26. [Defining the Perimeter in a Microservices World](#)

by [Twain Taylor](#), for [Twistlock](#), The New Stack, February 12, 2018.



Security perimeters for microservices are complex, yet they offer levels of security previously unavailable for older platforms.

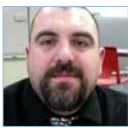
27. [Ten Priorities for Container Management and DevOps in Production and at Scale in 2018](#)

by [Enterprise Management Associates \(EMA\)](#), February 2018.

Based on input from 300 U.S. enterprises, this report offers analysis, trends and guidance for using DevOps to build container infrastructures at scale.

28. [Security Differences: Containers vs. Serverless vs. Virtual Machines](#)

by [Vince Power](#), for [Twistlock](#), The New Stack, August 7, 2018.



The security advantages and disadvantages of containers, serverless platforms and virtual machines are compared and contrasted.

29. See #26

30. See #26

31. [Security in the Modern Data Center](#)

by [Nitzan Niv](#), a system architect at [Alcide](#), The New Stack, Feb. 27, 2018.



Security for multiplatform and multicloud environments has become enormously complex: Security tools and practices must rise to the challenge by becoming at least as agile and efficient as the platforms they support.

32. [Making a Secure Transition to the Public Cloud](#)

by [Arul Elumalai](#), [James Kaplan](#), [Mike Newborn](#) and [Roger Roberts](#), of [McKinsey & Co.](#), January 2018.



McKinsey offers four practices it says are critical for organizations to follow after migrating their data and applications to a public cloud.

SPONSOR RESOURCE

• [CNCF Cloud Native Interactive Landscape](#)

by the [Cloud Native Computing Foundation](#) (CNCF)

A detailed, interactive directory of cloud native products, services and open source projects.

33. [Learning Paths on AWS: Break the Monolith](#)

by [Amazon Web Services](#), 2018.

Learn how to shift from a monolithic platform to a microservices platform using an application load balancer (ALB).

34. [Choosing a Microservices Deployment Strategy](#)

by [Chris Richardson](#) for [NGINX](#), February 10, 2016.



Several deployment patterns for microservices have emerged, including service instance per virtual machine and service instance per container, while AWS Lambda offers an option for serverless deployments.

35. [The Hardest Part About Microservices: Your Data](#)

by [Christian Posta](#), Christian Posta's blog, July 14, 2016.



Data and database management remain an often neglected component of microservices; reassessing your data and domain can help you better create microservice-based systems.

36. [The Forrester Wave: Database as a Service Q2 2017](#)

by [Forrester Research](#) via [Redis Labs](#), April 24, 2017.

Forrester analyzes, compares and critiques 13 of the leading Database as a Service (DBaaS) vendors, based on 30 sets of criteria.

37. [“DevOps Patterns” in “CI/CD with Kubernetes”](#)

by [Rob Scott](#), site reliability engineering architect at [ReactiveOps](#), The New Stack, 2018.



This first section of the ebook traces the history of DevOps and its effects on cloud native architectures, and also covers how Kubernetes has reshaped DevOps.

38. [“Continuous Delivery with Spinnaker” in “CI/CD with Kubernetes”](#)

by [Craig Martin](#), senior vice president of engineering and operations at [Kenzan](#), The New Stack, 2018.



Modern application architecture releases should be frequent, fast and, above all, boring. To this end, the growing tech movement to organize software teams and technologies around the notions of DevOps has created great interest in continuous delivery (CD) platforms.

39. [Twistlock Makes Istio’s Security Layer More Robust, Easier to Monitor](#)

by [Liron Levin](#), chief software architect, and [John Morello](#), chief technology officer, at [Twistlock](#), The New Stack, June 7, 2018.



Combining Twistlock data analysis and Istio’s service mesh management platform can improve microservices security layers and their management across cloud deployments.

40. [Characterizing the State of Microservices Adoption](#)

The New Stack Makers podcast with [Daniel Bryant](#), an independent consultant, speaker and writer; and [Matt Chotin](#), senior director of developer initiatives at [AppDynamics](#), March 29, 2018.



A discussion about the state of the microservices market and the business trends driving its adoption and growth.

41. [The New DevOps: Site Reliability Engineering Comes of Age](#)

by [Kieran Taylor](#), senior director of product marketing at [CA Technologies](#), The New Stack, July 5, 2018.



A description of the emergence of site reliability engineers (SRE) and the increasingly important role they play in DevOps for network and application monitoring.

42. [“Monitoring in the Cloud Native Era” in “CI/CD with Kubernetes”](#)

by [Ian Crosby](#), [Maarten Hoogendoorn](#), [Thijs Schnitger](#) and [Etienne Tremel](#), of [Container Solutions](#), The New Stack, 2018.



This ebook chapter describes how monitoring cloud deployments running on container platforms must offer advanced levels of



observability and scalability in addition to traditional monitoring capabilities.

43. [Netflix Conductor, a Microservices Orchestrator](#)

by [Viren Baraiya](#), engineering manager at [Google](#), Netflix Tech Blog, December 12, 2016.



This blog post describes how Netflix uses container orchestration and microservices to help boost the time to delivery and stability of its streaming services.

44. See #41

45. See #34

46. [Building Microservices: Designing Fine-Grained Systems](#)

by [Sam Newman](#), O'Reilly Media, February 2015.



A description of the trials and tribulations associated with building microservices, as well as the benefits the architectural style offers, while providing many practical examples.

SPONSOR RESOURCE

- [Cloud Native Security: What It Means, Why It's Hard & How To Achieve It](#)

by [Twistlock](#).

A white paper that outlines how to adopt a cloud native security strategy. This involves thinking beyond the perimeter, securing all the clouds, integrating security with CI/CD and supporting multiple deployment models.

47. [Managing Critical State: Distributed Consensus for Reliability](#)

by [Laura Nolan](#), site reliability engineer at [Google](#), O'Reilly Media, 2017.



How site reliability engineers can keep systems running despite various system failure risk, underscoring, among other things, the importance of monitoring.

48. [Paxos, a Really Beautiful Protocol for Distributed Consensus](#)

by [Mark Chu-Carroll](#), Good Math/Bad Math blog, January 30, 2015.



Discussion on the power of Paxos, a tool Chu-Carroll says helps developers “straddle the line between pure math and pure engineering.”

49. See #1**50. [How Chaos Engineering Can Drive Kubernetes Reliability](#)**

by [Jennifer Riggins](#), The New Stack, June 12, 2018.



The article discusses tools that use chaos theory, which has long been applied in mathematics and computing, to help build and maintain stability for Kubernetes deployments.

51. [Build System Confidence with Chaos Engineering and GitOps](#)

by [Sylvain Hellegouarch](#), co-founder and CTO at [ChaosIQ](#), Medium, February 22, 2018.



Learn how Chaos Toolkit can be applied to ensure a system remains in a steady state when, among other incidents, an application is no longer connected to the database.

52. See #46**53. [How To Do Microservices Integration Testing in the Cloud](#)**

by [Alex Handy](#), The New Stack, August 6, 2018.



It's nearly impossible to replicate data center environments, let alone cloud services-based architectures. What's a tester to do when it's not possible to build testing systems to be as complete as production systems?

Closing

The narrative about cloud native microservices starts with business objectives and evolves through organizational structure and practices. As with containers and Kubernetes, the adoption of microservices encompasses the drive for faster and continuous deployment, and reaches its full potential with DevOps.

As teams grow, we now see more of this need for declarative infrastructure. Application architectures built on DevOps practices work better and run with less friction. The developer has more control over their own resources, and the performance of the application becomes the primary focus. The better the performance, the happier the end user and the more uniform the feedback loop between users and developers. In this way, cloud native microservices provide game-changing business value.

With great execution can come great results. But the opposite is also true. There are barriers to overcome that may lead an organization to steer away from microservices adoption. These challenges may be technical, such as increased operational overhead and complexity, but just as often lie in business and process decisions.

This ebook has been the first to follow The New Stack's new approach to the way we develop ebooks. In the process, we've found an emerging theme centered on people and processes for the next ebooks in the series. Look for books on serverless and cloud native DevOps still to come this year with corresponding podcasts, in-depth posts and activities around the world wherever pancakes are being served.

Thanks and see you again soon.

Alex Williams

Founder and Editor-in-Chief, The New Stack

Disclosure

The following companies mentioned in this ebook are sponsors of The New Stack:

Alcide, AppDynamics, Aqua Security, Blue Medora, Buoyant, CA Technologies, Chef, CircleCI, CloudBees, Cloud Foundry Foundation, Cloud Native Computing Foundation, Google, InfluxData, LaunchDarkly, MemSQL, Mesosphere, Microsoft, Navops, New Relic, OpenStack Foundation, PagerDuty, Pivotal, Portworx, Pulumi, Puppet, Raygun, Red Hat, Rollbar, SaltStack, Stackery, The Linux Foundation, Tigera, Univa, VMware, Wercker and WSO2.

При поддержке



официального дистрибьютора решений



bakotech.ua
dynatrace.com

dynatrace@bakotech.com