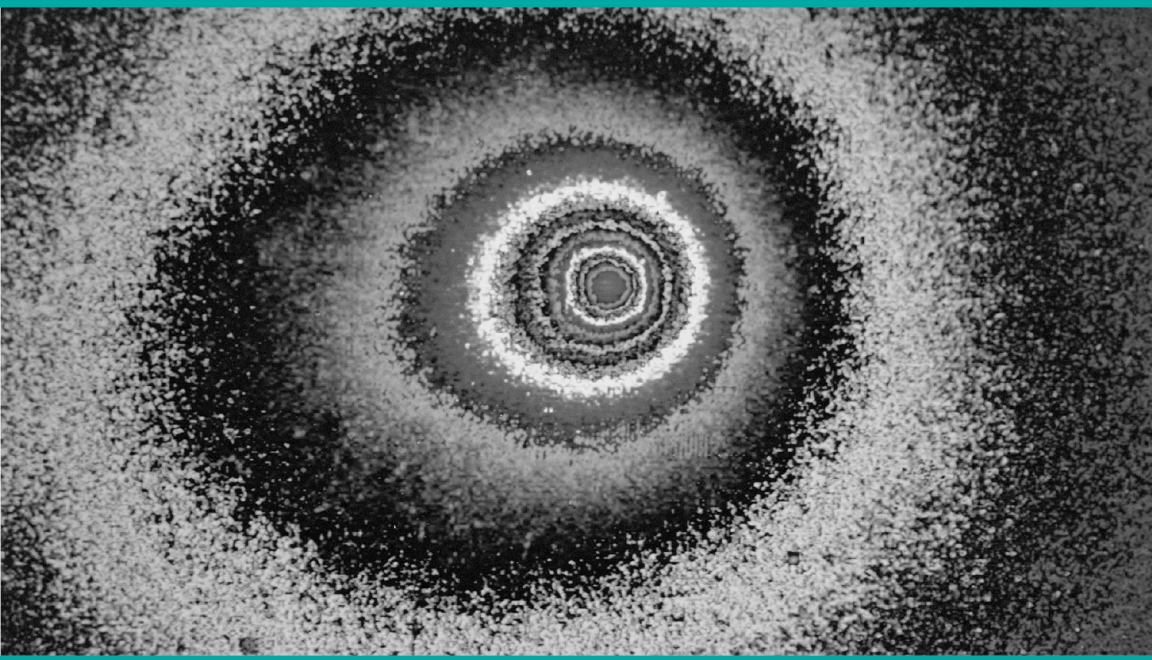
O'REILLY®

Cloud-Native Evolution

How Companies Go Digital



Alois Mayr, Peter Putz & Dirk Wallerstorfer
with Anna Gerber

Cloud-Native Evolution

How Companies Go Digital

Alois Mayr, Peter Putz, Dirk Wallerstorfer
with Anna Gerber

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Cloud-Native Evolution
by Alois Mayr, Peter Putz, Dirk Wallerstorfer with Anna Gerber

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://www.oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Brian Anderson
Production Editor: Colleen Lobner
Copyeditor: Octal Publishing, Inc.
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

February 2017: First Edition

Revision History for the First Edition
2017-02-14: First Release

978-1-491-97396-7

[LSI]

Table of Contents

Foreword

Every company that has been in business for 10 years or more has a digital transformation strategy. It is driven by markets demanding faster innovation cycles and a dramatically reduced time-to-market period for reaching customers with new features. This brings along an entirely new way of building and running software. Cloud technologies paired with novel development approaches are at the core of the technical innovation that enables digital transformation.

Besides building cloud native applications from the ground up, enterprises have a large number of legacy applications that need to be modernized. Migrating them to a cloud stack does not happen all at once. It is typically an incremental process ensuring business continuity while laying the groundwork for faster innovation cycles.

A cloud-native mindset, however, is not limited to technology. As companies change the way they build software, they also embrace new organizational concepts. Only the combination of both—new technologies and radical organizational change—will yield the expected successes and ensure readiness for the digital future.

When first embarking on the cloud-native journey company leaders are facing a number of tough technology choices. Which cloud platform to choose? Is a public, private or hybrid approach the right one? The survey underlying this report provides some reference insights into the decisions made by companies who are already on their way. Combined with real world case studies the reader will get a holistic view of what a typical journey to cloud native looks like.

— Alois Reitbauer, Head of Dynatrace Innovation Lab

CHAPTER 1

Introduction: Cloud Thinking Is Everywhere

Businesses are moving to cloud computing to take advantage of improved speed, scalability, better resource utilization, lower up-front costs, and to make it faster and easier to deliver and distribute reliable applications in an agile fashion.

Cloud-Native Applications

Cloud-native applications are designed specifically to operate on cloud computing platforms. They are often developed as loosely coupled microservices running in containers, that take advantage of cloud features to maximize scalability, resilience, and flexibility.

To innovate in a digital world, businesses need to move fast. Acquiring and provisioning of traditional servers and storage may take days or even weeks, but can be achieved in a matter of hours and without high up-front costs by taking advantage of cloud computing platforms. Developing cloud-native applications allows businesses to vastly improve their time-to-market and maximize business opportunities. Moving to the cloud not only helps businesses move faster, cloud platforms also facilitate the digitization of business processes to meet growing customer expectations that products and services should be delivered via the cloud with high availability and reliability.

As more applications move to the cloud, the way that we develop, deploy, and manage applications must adapt to suit cloud technologies and to keep up with the increased pace of development. As a consequence, yesterday's best practices for developing, shipping, and running applications on static infrastructure are becoming anti-patterns, and new best practices for developing cloud-native applications are being established.

Developing Cloud-Based Applications

Instead of large monolithic applications, best practice is shifting toward developing cloud-native applications as small, interconnected, purpose-built services. It's not just the application architecture that evolves: as businesses move toward microservices, the teams developing the services also shift to smaller, cross-functional teams. Moving from large teams toward decentralized teams of three to six developers delivering features into production helps to reduce communication and coordination overheads across teams.

NOTE

The "two-pizza" team rule credited to Jeff Bezos of Amazon is that a team should be no larger than the number of people who can be fed with two pizzas.

Cloud-native businesses like Amazon embrace the idea that teams that build and ship software also have operational responsibility for their code, so quality becomes a shared responsibility.[1]

> Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.
>
> —Werner Vogels, *CTO Amazon*

These shifts in application architecture and organizational structure allow teams to operate independently and with increased agility.

1 *http://queue.acm.org/detail.cfm?id=1142065*

Shipping Cloud-Based Applications

Software agility is dependent on being able to make changes quickly without compromising on quality. Small, autonomous teams can make decisions and develop solutions quickly, but then they also need to be able to test and release their changes into production quickly. Best practices for deploying applications are evolving in response: large planned releases with an integration phase managed by a release manager are being made obsolete by multiple releases per day with continuous service delivery.

Applications are being moved into containers to standardize the way they are delivered, making them faster and easier to ship. Enabling teams to push their software to production through a streamlined, automated process allows them to release more often. Smaller release cycles mean that teams can rapidly respond to issues and introduce new features in response to changing business environments and requirements.

Running Cloud-Based Applications

With applications moving to containers, the environments in which they run are becoming more nimble, from one-size-fits-all operating systems, to slimmed down operating systems optimized for running containers. Datacenters, too, are becoming more dynamic, progressing from hosting named in-house machines running specific applications toward the datacenter as an API model. With this approach, resources including servers and storage may be provisioned or deprovisioned on demand. Service discovery eliminates the need to know the hostname or even the location where instances are running—so applications no longer connect via hardwired connections to specific hosts by name, but can locate services dynamically by type or logical names instead, which makes it possible to decouple services and to spin up multiple instances on demand.

This means that deployments need not be static—instances can be scaled up or down as required to adjust to daily or seasonal peaks. For example, at 7 a.m. a service might be running with two or three instances to match low load with minimum redundancy. But by lunchtime, this might have been scaled up to eight instances during peak load with failover. By 7 p.m., it's scaled down again to two instances and moved to a different geolocation.

This operational agility enables businesses to make more efficient use of resources and reduce operational costs.

Cloud-Native Evolution

Businesses need to move fast to remain competitive: evolving toward cloud-native applications and adopting new best practices for developing, shipping, and running cloud-based applications, can empower businesses to deliver more functionality faster and cheaper, without sacrificing application reliability. But how are businesses preparing to move toward or already embracing cloud-native technologies and practices?

In 2016, the Cloud Platform Survey was conducted by O'Reilly Media in collaboration with Dynatrace to gain insight into how businesses are using cloud technologies, and learn their strategies for transitioning to the cloud.

There were 489 respondents, predominantly from the North America and European Information Technology sector. The majority of respondents identified as software developers, software/cloud architects, or as being in IT operations roles. Refer to Appendix A for a more detailed demographic breakdown of survey respondents.

94 percent of the survey respondents anticipate migrating to cloud technologies within the next five years (see Figure 1-1), with migration to a public cloud platform being the most popular strategy (42 percent).

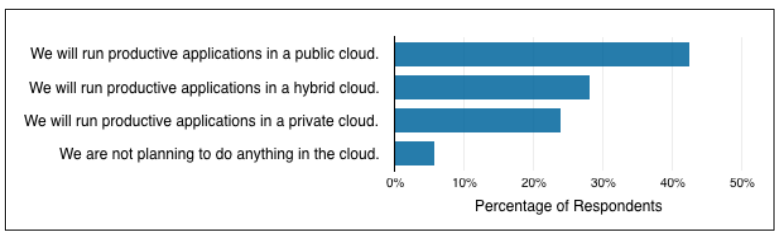


Figure 1-1. Cloud strategy within the next five years

The book summarizes the responses to the Cloud Platform Survey as well as insight that Dynatrace has gained from speaking with companies at different stages of evolution. An example of one such company is Banco de Crédito del Perú, described in Appendix B.

Based on its experience, Dynatrace identifies three stages that businesses transition through on their journey toward cloud-native, with each stage building on the previous and utilizing additional cloud-native services and features:

- Stage 1: continuous delivery
- Stage 2: beginning of microservices
- Stage 3: dynamic microservices

How to Read This Book

This book is for engineers and managers who want to learn more about cutting-edge practices, in the interest of going cloud-native. You can use this as a maturity framework for gauging how far along you are on the journey to cloud-native practices, and you might find useful patterns for your teams. For every stage of evolution, case studies show where the rubber hits the road: how you can tackle problems that are both technical and cultural.

CHAPTER 2

First Steps into the Cloud and Continuous Delivery

For businesses transitioning to the cloud, migrating existing applications to an Infrastructure as a Service (IaaS) platform via a "lift-and-shift" approach is a common first step. Establishing an automated continuous delivery pipeline is a key practice during this transition period, to ensure that the processes for delivering applications to cloud platforms are fast and reliable, and go hand-in-hand with implementing an Agile methodology and breaking up organizational silos.

This chapter examines challenges identified by respondents to the Cloud Platform Survey that businesses face as they take their first steps into the cloud. It also describes key best practices and enabling tools for continuous integration and delivery, automation, and monitoring.

Lift-and-Shift

The lift-and-shift cloud migration model involves replicating existing applications to run on a public or private cloud platform, without redesigning them. The underlying infrastructure is moved to run on virtual servers in the cloud; however, the application uses the same technology stack as before and thus is not able to take full advantage of cloud platform features and services. As a result, applications migrated following the lift-and-shift approach typically make less efficient use of cloud computing resources than cloud-

native applications. In addition, they might not be as scalable or cost effective to operate in the cloud as you would like. However, lift-and-shift is a viable strategy: redesigning a monolithic application to take advantage of new technologies and cloud platform features can be time consuming and expensive. Despite applications migrated via a lift-and-shift approach being less efficient than cloud-native applications, it can still be less expensive to host a ported application on a cloud platform than on traditional static infrastructure.

Challenges Migrating Applications to the Cloud

Although the applications can remain largely unchanged, there are a number of challenges to migrating applications to virtual servers, which organizations will need to consider when developing their cloud migration strategies in order to minimize business impacts throughout the process. The biggest challenge identified by survey respondents was knowing all of the applications and dependencies in the existing environment (59 percent of 134 respondents to this question—see Figure 2-1).

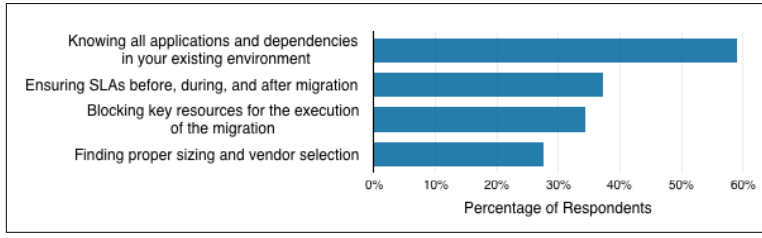


Figure 2-1. Challenges migrating to the cloud

Not all applications are suitable for hosting in the cloud. Migrating resource-intensive applications that run on mainframes, doing data-crunching, media processing, modeling, or simulation can introduce performance or latency issues. It can be more expensive to run these in a cloud-environment than to leave them where they are. Applications that rely on local third-party services also might not be good candidates for migration, because it might not be possible to (or the business might not be licensed to) run the third-party services in the cloud.

Some parts of an application might require minor refitting to enable them to operate or operate more efficiently within the cloud envi-

ronment. This might include minor changes to the application's source code or configuration; for example, to allow the application to use a cloud-hosted database as a service instead of a local database. Getting a picture of current applications and their dependencies throughout the environment provides the basis for determining which applications are the best candidates for migration in terms of the extent of any changes required and the cost to make them cloud-ready.

Starting small and migrating a single application (or part of an application) at a time rather than trying to migrate everything at once is considered good practice. Understanding dependencies through analyzing and mapping out connections between applications, services, and cloud components, will help to identify which part to migrate first, and whether other parts should be migrated at the same time as well, as to reveal any technical constraints that should be considered during the migration.

Understanding an application's dependencies and how it works can provide some clues for predicting how it might perform in a cloud environment, but benchmarking is an even better strategy for determining whether the level of service provided by a newly migrated cloud application is acceptable. The second biggest cloud migration challenge identified by 37 percent of the survey respondents in Figure 2-1, was ensuring service-level agreements (SLAs) before, during, and after migration. The level of service in terms of availability, performance, security, and privacy should be assessed through performance, stress, load, and vulnerability testing and audits. This can also inform capacity planning as well as vendor selection and sizing (simply for the sake of cost savings)—a challenge reported by 28 percent of respondents from Figure 2-1.

Continuous Integration and Delivery

Migrating applications to the cloud is not an overnight process. New features or bug fixes will likely need to be introduced while an application is in the process of being migrated. Introducing Continuous Integration and Continuous Delivery (CI/CD) as a prerequisite for the migration process allows such changes to be rapidly integrated and tested in the new cloud environment.

Continuous Integration

Continuous Integration (CI) is a development practice whereby developer branches are regularly merged to a shared mainline several times each day. Because changes are being merged frequently, it is less likely that conflicts will arise, and any that do arise can be identified and addressed quickly after they have occurred.

Organizations can achieve a faster release cycle by introducing a CI/CD pipeline. A deployment pipeline breaks the build-up into a number of stages that validate that recent changes in code or configuration will not result in issues in production. The purpose of the deployment pipeline is threefold:

- To provide visibility, so that information and artifacts associated with building, testing and deploying the application are accessible to all team members
- To provide feedback, so that all team members are notified of issues as soon as they occur so that they can be fixed as soon as possible
- To continually deploy, so that any version of the software could be released at any time

Continuous Delivery

The idea behind Continuous Delivery (CD) is that software is delivered in very short release cycles in such a way that it can be deployed into production at any time. The extension of Continuous Delivery is Continuous Deployment, whereby each code change is automatically tested and deployed if it passes.

Automation

Operating efficiently is key to becoming cloud-native. CI/CD can be performed through manually merging, building, testing, and deploying the software periodically. However, it becomes difficult to release often if the process requires manual intervention. So in practice, building, testing, and deployment of cloud applications are

almost always automated to ensure that these processes are reliable and repeatable. Successfully delivering applications to the cloud requires automating as much as possible.

Automated CI/CD relies on high-quality tests with high code coverage to ensure that code changes can be trusted not to break the production system. The software development life cycle (SDLC) must support test automation and test each change. Test automation is performed via testing tools that manage running tests and reporting on and comparing test results with predicted or previous outcomes. The "shift-left" approach applies strategies to predict and prevent problems as early as possible in the SDLC. Automated CI/CD and testing make applications faster and easier to deploy, driving frequent delivery of high-quality value at the speed of business.

Monitoring

During early stages of cloud migration, monitoring typically focuses on providing data on the performance of the migrated application and on the cloud platform itself. The ultimate goals for a monitoring solution are to support fast delivery cycles by identifying problems as early as possible and to ensure customer satisfaction through smooth operations. Monitoring solutions adopted during the early stages of cloud migration should support application performance monitoring, custom monitoring metrics, infrastructure monitoring, network monitoring, and end-to-end monitoring, as described here:

Application performance monitoring
: Modern monitoring solutions are able to seamlessly integrate with CI/CD and yield a wealth of data. For example, a new feature version can be compared to the previous version(s) and changes in quality and performance become apparent in shorter or longer test runtimes. Thus monitoring becomes the principal tool to shift quality assurance from the end of the development process to the beginning (the aforementioned shift-left quality approach). Ideally, a monitoring tool identifies the exact root-cause of a problem and lets developers drill down to the individual line of code at the source of the trouble.

Creating custom monitoring metrics
: Another approach is to look at the CI pipeline itself and to focus on unusual log activities like error messages or long compilation times. Developers can create their own custom logs and metrics

to detect performance issues as early as possible in the development process.

Infrastructure monitoring
: A monitoring platform also needs to provide insights into the cloud infrastructure. The most basic question for any cloud platform user is: do we get what we pay for? That refers to the number of CPUs (four virtual CPUs might not be equivalent to four physical CPUs), the size of memory, network performance, geolocations available, uptime, and so on. Cloud instances tend to be unstable and fail unpredictably. Does this lead to performance problems or is it corrected on the fly by shifting the load or by firing up new instances? The ephemeral nature of cloud instances (cattle versus pets) makes monitoring more difficult, too, because data needs to be mapped correctly across different instances.

Network monitoring
: Network monitoring becomes essential for a number of reasons. The network is inherently a shared resource, especially in a cloud environments. Its throughput capacity and latency depend on many external factors and change over time. The network in a cloud environment is most likely a virtual network with additional overheads. It is important to understand the impact of all this for the application performance in different geolocations but also locally on the traffic between separate application components.

End-to-end monitoring
: If users experience performance bottlenecks, it can be the "fault" of the cloud or caused by the application itself. For reliable answers, you need a full stack monitoring solution that correlates application metrics with infrastructure metrics. End-to-end monitoring also provides valuable data for capacity planning. In what components do you need to invest to increase performance and availability of services? Or are there overcapacities and potentials for cost savings?

Infrastructure as a Service

In the early stages of moving into the cloud, the tech stack for most applications will remain largely unchanged—applications use the same code, libraries, and operating systems as before. Porting appli-

cations to the cloud involves migrating them from running on traditional infrastructure, to virtual machines (VMs) running on an Infrastructure as a Service (IaaS) platform. Seventy-four percent of respondents to the Cloud Platform Survey reported that they are already running IaaS in production (364 out of 489).

IaaS technologies provide virtualized computing resources (e.g., compute, networking, and storage) that you can scale to meet demand. The switch to virtual servers rather than physical servers facilitates faster and more flexible provisioning of compute power, often via API calls, enabling provisioning to be automated.

Enabling Technologies

In addition to IaaS platforms for hosting the virtual servers, enabling technologies for this first stage of cloud evolution include Configuration Management (CM) tools, to manage the configuration of the virtual server environments, and CI/CD tools to enable applications to be be deployed to these virtual servers, quickly and reliably.

Continuous Integration and Delivery Tools

There are many tools available that you can use to automate CI/CD. Many of these tools support extension and integration via plug-ins. Some popular tools and services are listed here:

Jenkins
: This is an open Source CI tool originally designed for use with Java. Plugins support building software written in other languages, a range of SCM tools and testing tools. The Jenkins server runs in a servlet container (e.g., Apache Tomcat) and is often run on-site, but providers like CloudBees also provide hosted versions.

JetBrains TeamCity
: This is a commercial Java-based CI server, with a more modern and user-friendly UI than Jenkins. Free and enterprise versions are available. Like Jenkins, TeamCity supports integrations through plug-ins.

Travis CI
: An open source hosted CI solution for projects hosted on GitHub. It is free for open source projects. A self-hosted enterprise version is also available.

CircleCI
: This is a hosted CI/CD tool with support for unit and integration testing and deploying applications to Docker containers as well as to a range of Platform as a Service (PaaS) platforms including Heroku, Amazon Web Services (AWS), and Google Cloud Platform. It provides a number of integrations supporting additional SCM systems as well as testing and deployment services, such as assessing code quality and coverage, and for automated browser testing for web applications.

Atlassian Bamboo
: Commercial Professional CI/CD platform that integrates with Atlassian tools. Extensible through developing add ons, with an extensive add-on marketplace supporting integrations, language packs, reporting, admin tools, connectors, and so on.

With most CI servers, builds are triggered automatically when code is checked into a Source Control Management (SCM) repository, but you can schedule these (e.g., through cron), or activate them via an API.

CM Tools

CM and deployment management tools (also sometimes referred to generally as IT Automation Tools) automate applying configuration states to establish and synchronize the state of virtual servers. When a new virtual server is deployed, CM tools are used to automatically provision the server instance on the cloud platform of choice as well as running configuration scripts to set up the server environment, install required libraries and binaries, deploy the application into the sever, and so on. Popular CM tools include Chef, Ansible, Puppet, and Salt. These tools support extension through development of custom plug-ins and each have an active open source community established around them. Let's take a look at each of these tools:

Puppet
: A model-driven CM tool that uses JSON syntax for manifests, and one of the more mature CM tools. The Puppet Master syn-

chronizes the configuration across puppet nodes (i.e., the target servers being managed by puppet).

Chef
: Chef uses a Ruby-based domain-specific language for configuration scripts (known as *recipes*) that are stored in Git repositories. A master server communicates with agents installed on the managed servers.

Ansible
: Ansible was developed as a lightweight response to performance concerns with Puppet and Chef. It does not require the installation of an agent—communication occurs via Secure Shell (SSH). Configuration of Ansible playbooks is in YAML (YAML Ain't Markup Language) format. Ansible Tower is an enterprise offering built over Ansible that provides a web-based dashboard that teams can use to to manage and scale Ansible deployments.

SaltStack
: This tool supports Python commands at the command-line interface (CLI), or configuration via PyDSL. Like Chef, SaltStack uses a master server and agents, known as minions, to manage target servers. Salt was designed to be scalable and resilient, supporting hierarchically tiered masters to provide redundancy. SaltStack uses a ZeroMq communication layer for communication between master and target servers which makes it very fast compared to Puppet or Chef.

For IaaS-hosted cloud applications to be effectively scaled to multiple instances, migrated between datacenters, or recovered quickly after failure, it is imperative that the virtual servers hosting the application are able to be replicated quickly. CM Tools automate this process, eliminating yet another source of risk and enabling fast, reliable deployment to IaaS platforms.

IaaS Technologies

IaaS platforms provide computing resources in the form of VMs. Rather than physical CPU cores, virtual processors (vCPUs) are assigned to VMs, by default one vCPU per machine. VMs can be allocated more than one vCPU, to enable multithreaded applications to execute tasks concurrently. These are typically provided on a per-use basis, and the customer is typically billed for using these resour-

ces by the hour, or month. Virtual cores might not map 1:1 to physical cores; for example, a business might choose to over-provision vCPUs to pCPUs in their private VMWare cloud to make more effective use of physical hardware resources. IaaS allow effective management of other resources including network interfaces and storage, allowing them to be added flexibly to instances as required.

IaaS platforms can be public (hosted by a third-party like Amazon), private to the organization, or a hybrid of the two.

Private clouds are usually hosted using private infrastructure on-premises. For organizations dealing with sensitive data, a private cloud provides the benefit of more control. The close proximity also reduces latency.

A hybrid cloud is a blend of both private and public cloud for which services are spread across both public and private infrastructure with orchestration between. A hybrid cloud can provide a best-of-both-worlds solution; for example, an organization might primarily use a private cloud and only spin up instances on a public cloud for failover, when there aren't enough resources in the private cloud to meet demand. This strategy ensures that organizations retain flexibility and resiliency while capping private infrastructure costs.

Tables 2-1 and 2-2 show the breakdown of adoption of public versus private IaaS technologies among survey respondents who indicated they are using IaaS (364 respondents in total).

Table 2-1. IaaS platform usage (public cloud)

IaaS platform (public cloud)	Count	%
AWS EC2	175	74 percent
DigitalOcean	25	11 percent
Azure Compute	18	8 percent
Google Cloud Platform CE	17	7 percent

Public cloud platforms were the most frequently adopted with 65 percent of the survey respondents using an IaaS platform in production. Of those respondents using a private IaaS platform, slightly more than half were using VMware vSphere, with OpenStack being the other commonly adopted platform.

Table 2-2. IaaS platform usage (private cloud)

IaaS platform (public cloud)	Count	%
VMware vSphere	45	54 percent
OpenStack	39	46 percent

Of the 235 respondents who have adopted a public IaaS in production, 42 percent (86 respondents) indicated that they intend to migrate from their current strategy, either within the next four years or as an ongoing process (Figure 2-2). Of those 86 respondents intending to migrate, 70 percent anticipated moving to a hybrid cloud rather than switching exclusively to a private cloud solution.

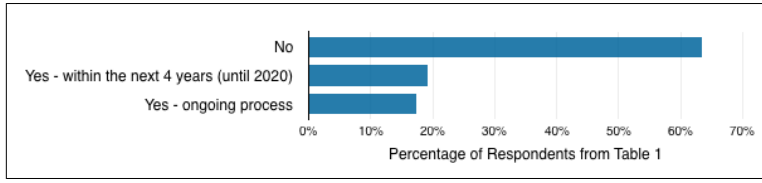


Figure 2-2. Are you planning to migrate (parts of) your public cloud to a private or hybrid cloud?

The top drivers for migrating from public to private or hybrid cloud are listed in Figure 2-3. Reducing costs was the most common motivating factor, reported by 67 percent of the 86 respondents.

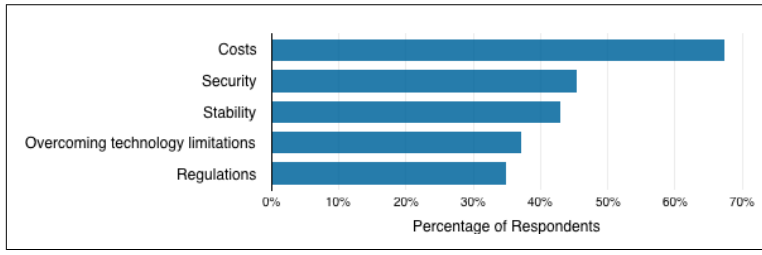


Figure 2-3. Motivations for migrating from public cloud

Of the 84 respondents using private IaaS platforms in production, 57 percent intend to migrate to a public or hybrid cloud (Figure 2-4), with 77 percent of those respondents indicating that they plan to adopt a hybrid approach. This is comparable to the results for public cloud migration, with a combined figure of 72 percent of respondents who are currently using IaaS in production planning to migrate to a hybrid cloud.

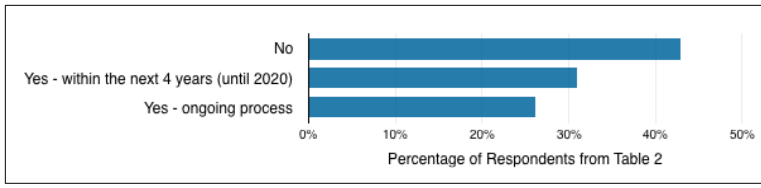


Figure 2-4. Are you planning to migrate (parts of) your private cloud to a public or hybrid cloud?

Figure 2-5 shows that scalability was the number one reason given for migrating from a private cloud (71 percent of respondents) with flexibility also a major consideration.

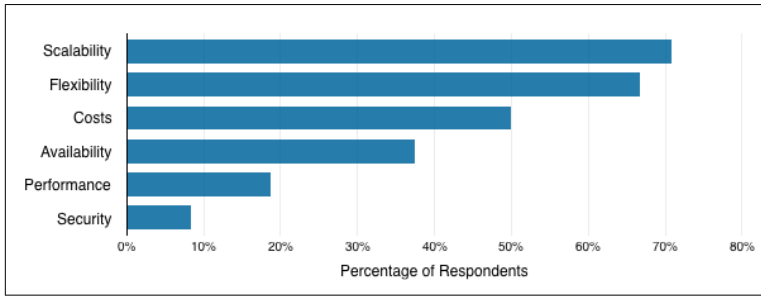


Figure 2-5. Motivations for migrating from private cloud

Public cloud IaaS technologies

The top public IaaS technology platforms in use by survey respondents included AWS EC2, DigitalOcean, Azure Compute, and Google Cloud Platform CE.

AWS EC2
: AWS Elastic Compute Cloud (EC2) is central to Amazon's widely adopted cloud platform. AWS EC2 was the most popular of the public IaaS offerings from Table 2-1, used by 74 percent of the 235 public cloud adopters.

DigitalOcean
: Digital Ocean provides low-cost Unix-based virtual servers (known as droplets) via a minimalistic user interface and simple API aimed at software developers. Eleven percent of survey respondents in Table 2-1 have adopted DigitalOcean.

Azure Compute
: This is Microsoft's compute service with support for Linux and Windows VMs. Eight percent of survey respondents use Azure Compute.

Google Cloud Platform CE
: Google's Compute Engine offers Linux and Windows-based virtual servers and is in use by 7 percent of the survey respondents from Table 2-1.

Private cloud IaaS technologies

Of those respondents on private clouds, VMWare vSphere and OpenStack were the top platforms adopted.

VMWare vSphere
: This is VMWare's suite of professional products built around VMware ESXi VMs. This was the most popular private IaaS option among the survey respondents, with 54 percent of respondents from Table 2-2 reporting that they use VMWare vSphere for their private cloud.

OpenStack
: OpenStack is an open source cloud operating system managed by the non-profit OpenStack Foundation, designed to be used for both public and private cloud platforms. Thirty-nine respondents (12 percent of all respondents from Tables 2-1 and 2-2) reported that they are using OpenStack in production. Fifty-four percent of respondents using OpenStack indicated that they were responsible for operating and maintaining IaaS in their OpenStack cluster, whereas 51 percent were responsible for maintaining applications running on IaaS.

Figure 2-6 shows the reported number of physical cores in the OpenStack clusters by those 39 respondents. Slightly more than a quarter (26 percent) of respondents were running between 10 and 99 cores, with a further 26 percent running between 100 and 999 cores. Thirty-eight percent of respondents were running a cluster with more than 1,000 cores.

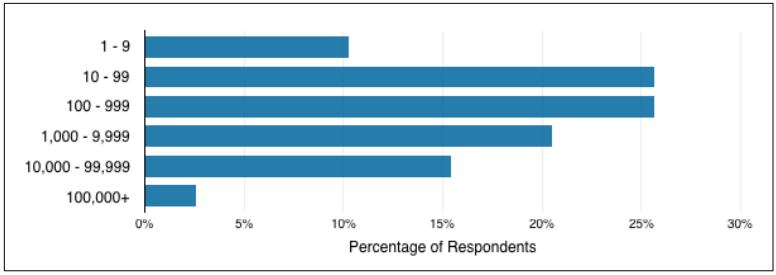


Figure 2-6. How many physical cores does your OpenStack cluster have?

Conclusion

Initial steps into the cloud typically involve migrating existing applications to a public or private IaaS cloud platform via a lift-and-shift approach, and establishing a CI/CD pipeline to speed up application development and delivery. Key practices in this early phase of cloud migration include automated testing and monitoring:

- The processes for checking code quality need to keep pace, or the end result will be shipping poor-quality code into production faster; hence, automated testing is a key practice during this phase.
- Monitoring keeps tabs on the performance of the application and cloud platform and helps to identify and diagnose any problems that might have arisen during the migration.

Case Study: Capital One—A Bank as a World-Class Software Company

Founded little more than 20 years ago, Capital One is today one of the largest digital banks and credit card issuers in the United States. Given the fast digitization in the banking industry, Rich Fairbank, the founder and CEO is convinced that "…the winners in banking will have the capabilities of a world-class software company."

The goal of digitization is to "deliver high-quality working software faster." This requires a novel approach to software engineering, changed organizational roles, and a completely new set of individual skills. Software engineering strategies adopted for delivering quality

software rapidly include using open source technologies and developing an open source delivery pipeline, as described here:[1]

Capitalizing on open source technologies

> Capital One is building its own software relying on open source technologies, microservice architectures and public cloud infrastructures (mostly AWS) as production, development, and testing environments. And it subscribed to Continuous Delivery and DevOps.

Delivery pipeline as key technology

> The CI/CD pipeline covers the complete technology stack: all application software, the infrastructure (it is code, too!) as well as all testing procedures. The tests include static scans (security antipattern checks), unit testing, acceptance tests, performance tests, and security tests. Capital One developed its own open source pipeline dashboard Hygieia. The goal was to increase the visibility of code moving through the pipeline, identify bottlenecks and ultimately speed up the delivery process even more.[2]

Capital One employs about 45,000 people, including thousands of software engineers. The technical challenges required developing completely new skills sets, changing existing workflows and practices, and enabling continuous learning and knowledge sharing. This was achieved by moving to cross-functional teams and facilitating learning and knowledge sharing through communities of practice, open spaces, meetups, and conferences, as described here:

Individual skills

> There are discussions in the broader DevOps community whether in the near future software testers will be out of their jobs and replaced by programmers. Adam and Tap talk about the evolution of skills. Previously, testers and quality assurance engineers had a single skill set for specialized tasks like security testing, performance testing, functional testing, test data generation, and so on. Now testers are part of cross-functional teams, where everybody knows a programming language. They need to

1 Auerbach, Adam, and Tapabrata Pal. "Part of the Pipeline: Why Continuous Testing Is Essential." Presented at Velocity, Santa Clara, CA, June 23, 2016. *http://oreil.ly/2e7R1Zv*.

2 PurePerformance Cafe 002: Velocity 2016 with Adam Auerbach and Topo Pal. *http://bit.ly/2e7OdM2*.

know the cloud and orchestration and scheduling technologies. They need to be intimately familiar with the deployment pipeline, with integrating tests, and creating custom reports. The new generation of testers will take on DevOps roles or system integrator roles or even become full developers because of their new skill sets.

Guilds and Communities of Practice (CoP)
: How can training for these completely new skill sets scale for a big enterprise? Besides an online training curriculum, CoPs and guilds became central to a culture of sharing and contributing. A CoP is a semiformal team focused on a specific topic. Members from unrelated departments come together to solve problems in their area, share knowledge, and document new solutions for the broader community.

Open Spaces
: To facilitate learning on an even bigger scale, Capital One organizes meetups, Open Space events, and conferences. At Open Spaces, 100 to 200 people come together with a common problem and mingle with some experts. The events are based on the principle of self-organization for which the agenda only emerges after an initial opening circle. As Adam and Tap put it, "It's a great way to bring a bunch of people together and jumpstart their knowledge and their networks."

Conferences
: Capital One organizes the annual Software Engineering Conference SECON just for their own employees with no vendors. In 2016, more than 2,000 employees met and 146 sessions took place. All sessions were focused on new topics that are specifically relevant to the bank like microservices, Bitcoin and the blockchain, new programming languages like Go, AWS Lambda functions, and so on. There were also 54 tech booths for teams to present and discuss their work. One purpose of conferences is to create a culture of "reuse." In a typical engineering culture, heroes who create something new are awarded. In a big organization fast adoption also needs a rich culture of reuse and evolutionary improvements.

Key Takeaways

Capital One accomplished its goal to deliver high-quality software faster by investing in an advanced and continuously improving delivery pipeline. This was paired with breaking up organizational silos, creating new DevOps roles, and scaling knowledge acquisition through CoPs, Open Spaces, and Conferences.

CHAPTER 3

Beginning of Microservices

Although a lift-and-shift approach to migrating a monolithic application allows for quick migration and can save on short-term costs, the operational costs of running software that has not been optimized for the cloud will eventually overtake the cost of development. The next stage toward cloud-native is adopting a microservice architecture to take advantage of improved agility and scalability.

Embrace a Microservices Architecture

In a microservices architecture, applications are composed of small, independent services. The services are loosely coupled, communicating via an API rather than via direct method calls, as in tightly coupled monolithic application. A microservices architecture is more flexible than a monolithic application, because it allows you to independently scale, update, or even completely replace each part of the application.

It can be challenging to determine where to begin when migrating from a large legacy application toward microservices. Migrate gradually, by splitting off small parts of the monolith into separate services, rather than trying to reimplement everything all at once. The first candidates for splitting off into microservices are likely to be those parts of the application with issues that need to be addressed, such as performance or reliability issues, because it makes sense to begin by redeveloping the parts of the application that will benefit most from being migrated to the cloud.

Another challenge with splitting up monolithic applications is deciding on the granularity for the new services—just how small should each service be? Too large and you'll be dealing with several monoliths instead of just the one. Too small and managing them will become a nightmare. Services should be split up so that they each implement a single business capability. You can apply the Domain-Driven Design (DDD) technique of context mapping to identify bounded contexts (conceptual boundaries) within a business domain and the relationship between them. From this, you can derive the microservices and the connections between them.

Microservices can be stateful or stateless. Stateless services do not need to persist any state, whereas stateful services persist state; for example, to a database, file system, or key-value store. Stateless services are often preferred in a microservices architecture, because it is easy to scale stateless services by adding more instances. However, some parts of an application necessarily need to persist state, so these parts should be separated from the stateless parts into stateful services. Scaling stateful services requires a little more coordination; however, stateful services can make use of cloud data stores or make use of persistent volumes within a container environment.

Containers

As mentioned earlier, container environments are particularly well suited to microservices.

Containers are a form of operating system virtualization. Unlike hypervisor-based virtualization (i.e., traditional VMs), which each contain a separate copy of the operating system, system binaries, and libraries as well at the application code, containers running on the same host run on the same shared operating system kernel. Containers are a lightweight environment for running the processes associated with each service, with process isolation so that each service runs independently from the others.

Each container encapsulates everything that the service needs to run—the application runtime, configuration, libraries, and the application code for the microservice itself. The idea behind containers is that you can build them once and then run them anywhere.

Almost 60 percent of the respondents to the Cloud Platform Survey reported that they are not running containers in production (Figure 3-1). However, more than half (61 percent) of those not currently running container-based environments in production are planning to (Figure 3-2) adopt containers within the next five years.

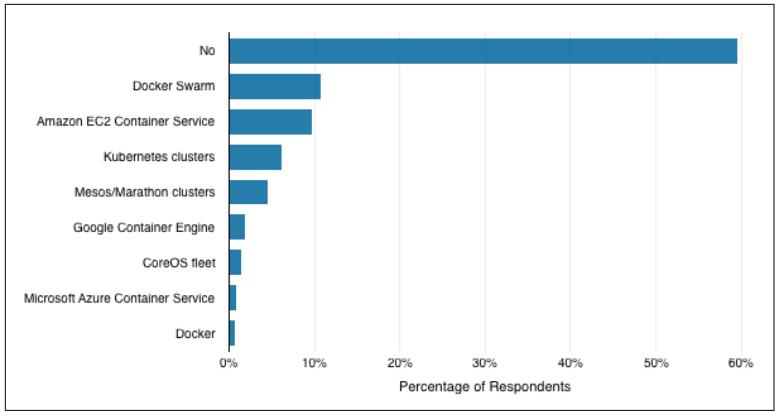


Figure 3-1. Are you running container-based environments in production?

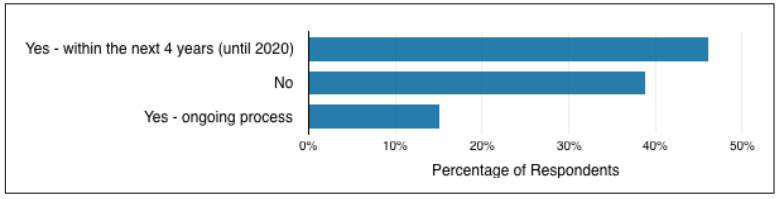


Figure 3-2. Are you planning to run a container-based environment in production?

Considering just those respondents using OpenStack, 77 percent of the survey respondents currently using that tool indicated that they plan to adopt containers in their OpenStack cluster. Forty-one percent anticipate introducing autoscaling of resources (Figure 3-3).

Figure 3-4 shows the top motivations for running containers for these 178 respondents. Common motivations for the majority of the respondents included flexible scaling of services (65 percent of respondents), resource efficiency (53 percent), and migrating to microservices (51 percent).

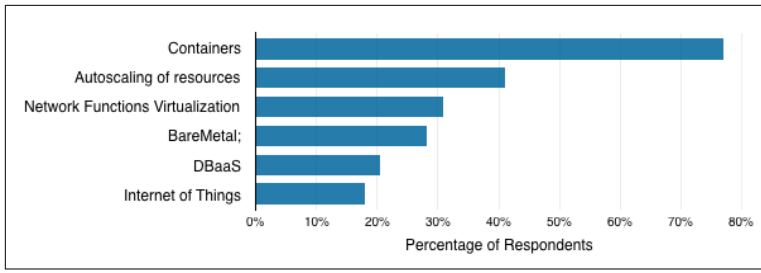


Figure 3-3. Which new/emerging technologies are you planning to use in your OpenStack cluster?

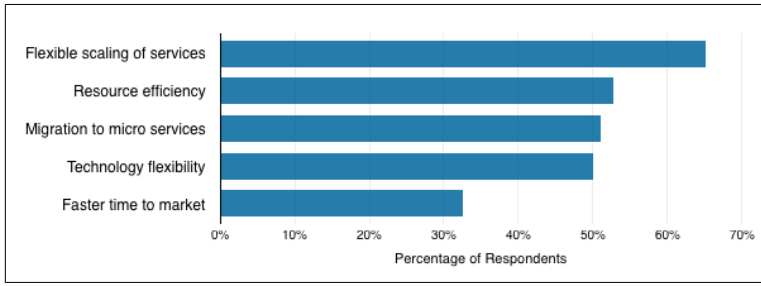


Figure 3-4. Motivations for planning to run container-based environments in production

Forty-five percent of the 178 respondents planning to adopt containers indicated that they will adopt Docker. Other popular container technologies include EC2 Container Service, runc, rkt, and Mesoscontainer.

The traditional Platform as a Service (PaaS) approach for configuring application environments is via buildpacks. Originally designed for Heroku, buildpacks provide runtime support for applications running on the PaaS, for example by including scripts that download dependencies and configure the application to communicate with services. However, you can also launch PaaS containers from images; for example, Docker containers are launched from images created from Dockerfiles.

PaaS

Container technologies enable the journey toward a microservices architecture. Containers are light-weight environments for running microservices. Moving applications into containers, with continuous

service delivery, standardizes the way that applications are delivered. Rather than IaaS running a monolithic application with the technology stack for the application largely unchanged, an entire new technology stack is required for the microservices—PaaS is adopted. PaaS provides platform abstraction to make it easier for development teams to deploy applications, allowing them to focus on development and deployment of their applications rather than DevOps.

Only 21 percent of survey respondents are currently running PaaS in production, compared to 74 percent running IaaS (Figure 3-5).

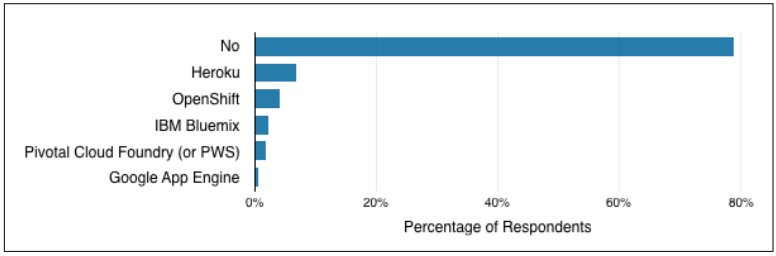


Figure 3-5. Are you running PaaS environments in production?

However, 34 percent of those not currently using PaaS in production indicated that they do intend to adopt PaaS. The primary motivation reported by those 132 respondents planning to adopt PaaS was streamlining development and deployment, reported by 66 percent of respondents. Other top motivations included resource efficiency, ease of use, establishing company standards and security (Figure 3-6).

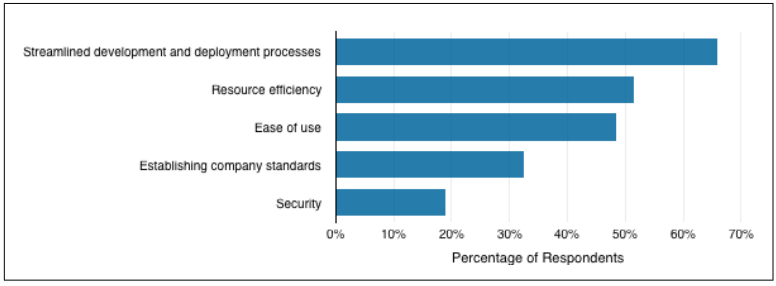


Figure 3-6. Motivations for adopting PaaS

Many PaaS providers are also Containers as a Service (CaaS) providers. Containers have become the industry standard for implementing microservices on both public and private cloud platforms.

Of the 104 respondents who are running PaaS in production, 59 percent run their PaaS on a public cloud (Figure 3-7).

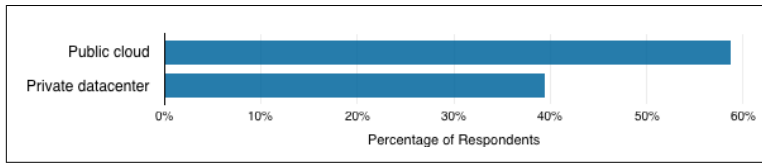


Figure 3-7. Where do you run your PaaS environment?

Adopting containers and a PaaS helps to streamline the release process, resulting in faster development iterations to push code to production. However, as more services are split off into separate containers, to make the most of containers they need to be orchestrated to coordinate applications spanning multiple containers. To make efficient use of cloud resources and for improved resilience, load balancing should be in place to distribute workloads across multiple service instances, and health management should be a priority for automated failover and self-healing.

Polyglot Architectures

With microservices, services communicate via high-level APIs, and so unlike a monolithic application, services implementing each part of the application do not need to use the same technology stack. Development teams have the autonomy to make architecture decisions that affect only their own services and APIs. They can employ different technologies for different purposes in the architecture instead of having to conform to use a technology that doesn't fit. For example, a team might implement a messaging queue service using Node.js and a NoSQL cloud data store, whereas a team responsible for a geolocation service might implement it using Python.

Independent Release Cycles

Splitting a monolith into smaller decoupled services also means that development teams are no longer as dependent on one another. Teams might choose to release their services on a schedule that suits them. However, this flexibility comes at the price of additional complexity: when each team is operating with independent release cycles, it becomes more difficult to keep track of exactly what services and which version(s) of each service are live at any given time. Communication processes need to be established to ensure that

DevOps and other development teams are aware of the current state of the environment and are notified of any changes. Automated testing should also be adopted within a Continuous Integration/Continuous Delivery (CI/CD) pipeline, to ensure that teams are notified as soon as possible if any changes to services on which they depend have resulted in issues.

Monitoring

Microservices architectures by nature introduce new challenges to cloud computing and monitoring:

Granularity and locality
: It is an architectural challenge to define the scope and boundaries of individual microservices within an application. Monitoring can help to identify tightly coupled or chatty services by looking at the frequency of service calls based on user or system behavior. Architects might then decide to combine two services into one or use platform mechanisms to guarantee colocation (for example Kubernetes pods).

Impacts of remote function calls
: In-memory function in monoliths turn into remote service calls in the cloud, where the payload needs to include actual data versus only in-memory object references. This raises a number of issues that depend heavily on the locality and chattiness of a service: how large is the payload value compared to the in-memory reference? How much resources are used for marshalling/unmarshalling and for encoding? How to deal with large responses? Should there be caching or pagination? Is this done locally or remotely? Monitoring provides valuable empirical baselines and trend data on all these questions.

Network monitoring
: Network monitoring gets into the limelight as calls between microservices usually traverse the network. Software-Defined Networks (SDNs) and overlay networks become more important with PaaS and dynamic deployments. Although maintenance and administration requirements for the underlying physical network components (cables, routers, and so on) decline, virtual networks need more attention because they come with network and computing overhead.

Polyglot technologies
: Monitoring solutions need to be able to cover polyglot technologies as microservice developers move away from a one-language-fits-all approach. They need to trace transactions across different technologies like, for example, a mobile front-end, a Node.js API gateway, a Java or .NET backend, and a MongoDB database.

Container monitoring
: With the move toward container technologies, monitoring needs to become "container-aware"; that is, it needs to cover and monitor containers and the service inside automatically. Because containers are started dynamically—for example, by orchestration tools like Kubernetes—static configuration of monitoring agents is no longer feasible.

Platform-aware monitoring
: Furthermore, monitoring solutions need the capabilities to distinguish between the performance of the application itself and the performance of the dynamic infrastructure. For example, microservice calls over the network have latency. However, the control plane (e.g., Kubernetes Master) also uses the network. This could be discarded as background noise, but it is there and can have an impact. In general, cloud platform technologies are still in their infancy and emerging technologies need to be monitored very closely because they have the potential for catastrophic failures.

Monitoring as communications tool
: Microservices are typically developed and operated by many independent teams with different agendas. Using an integrated monitoring solution that covers the engineering process from development through operation, not only allows for better problem analysis but has significant benefits for establishing and nurturing an agile DevOps culture and for providing business executives with the high-level views they need.

Cultural Impact on the Organization

Just as adopting microservices involves breaking applications into smaller parts, migrating to microservices architecture also involves breaking down silos within the organization—opening up communication, sharing responsibility, and fostering collaboration. Accord-

ing to Conway's Law, organizations produce application designs that mimic the organization's communication structure. When moving toward microservices, some businesses are applying an "inverse Conway maneuver"—reorganizing their organizational structure into small teams with lightweight communication overheads.

> Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure.
>
> —Melvin Conway

Decentralized governance is desirable for microservices architectures—autonomous teams have the flexibility to release according to their own schedule and to select technologies that fit their services best. The extension of this strategy is that teams take on responsibility for all aspects of their code, not just for shipping it—this is the "you build it, you run it" approach. Rather than having a DevOps team (another silo to be broken down), teams develop a DevOps mindset and shift toward cross-functional teams, in which team members have a mix of expertise. However, although DevOps might be being absorbed into the development teams, there is still a need for Ops, to support development teams and manage and provide the platform and tools that enable them to automate their CI/CD and self-service their deployments.

Although communication between teams is reduced, teams still need to know their microservice consumers (e.g., other development teams), and ensure that their service APIs continue to meet their needs while having the flexibility to update their service and APIs. Consumer-driven contracts can be used to document the expectations of each consumer by autogenerating the documentation from their code where possible so that it does not grow stale, and real examples of how other teams are using the APIs can also be added to test suites.

There might also be the need for some minimal centralized governance: as the number of teams increases and the technology stack becomes more diverse, the microservices environment can become more unpredictable and more complex, and the skills required to manage microservices-based applications become more demanding. Just because a team wants to use a particular technology does not mean that the Ops team or platform will be willing to support it.

Adopting containers is one strategy for dealing with this complexity. Containers are designed to be built once and run anywhere, so they allow the development team to focus on taking care of its apps, and Ops doesn't need to know what is inside of the containers to keep them running.

Challenges Companies Face Adopting Microservices

The biggest challenges identified by survey respondents with PaaS in production include integrating with (legacy) systems, mastering complexity of application dependencies, technology maturity, and scaling application instances (Figure 3-8).

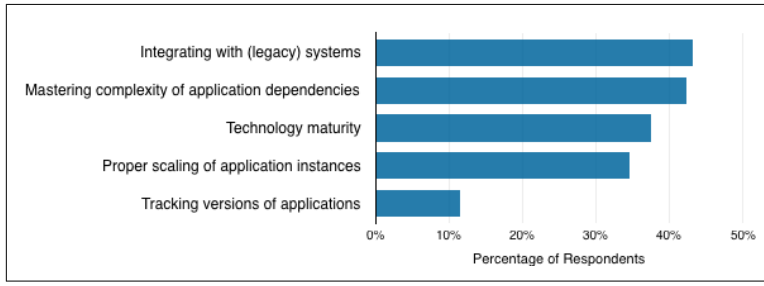


Figure 3-8. What are the biggest challenges with your PaaS environment?

Specifically looking at those respondents using container-based environments, the biggest challenge reported by the 198 respondents using containers was technology maturity (Figure 3-9). Container technologies themselves continue to evolve, with container management and orchestration toolchains in particular changing rapidly and sometimes in ways that are not compatible with other container systems. Efforts by groups like the Open Container Initiative to describe and standardize behavior of container technologies might help to ease this concern as container technologies mature.

Once again, mastering complexity of dependencies was a concern, identified by 44 percent of the respondents using containers. With organizational changes associated with adopting microservices leading to teams adopting a DevOps mindset, teams can independently deploy, which also increases complexity of the microservices envi-

ronment. This change in mindset was a concern for 41 percent of respondents.

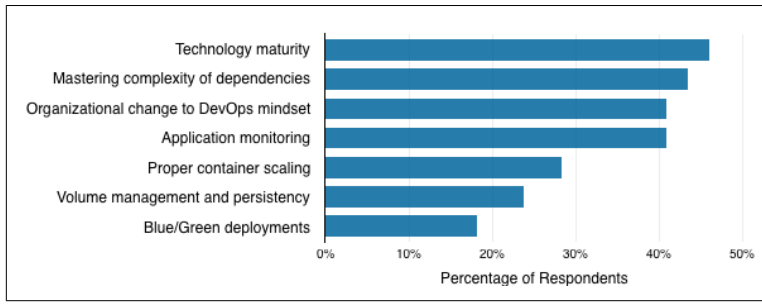


Figure 3-9. What are the biggest challenges with your container-based environment?

The other top concern was application monitoring, (41 percent of respondents). With multiple development teams working faster and application complexity and dependencies increasing as applications scale, application monitoring—service-level monitoring in particular—is essential to ensure that code and service quality remain high. Other common concerns included container scaling, volume management, and *blue/green* deployments.

Blue/green deployment is a strategy for minimizing downtime and reducing risk when updating applications. Blue/green deployments are possible when the platform runs two parallel production environments (named blue and green for ease of reference). Only one environment is active at any given time (say, blue), while the other (green) remains idle. A router sits in front of the two environments, directing traffic to the active production environment (blue).

Having two almost identical environments helps to minimize downtime and reduce risk when performing rolling updates. You can release new featuresinto the inactive green environment first. With this release strategy, the inactive (green) environment acts as a staging environment, allowing final testing of features to be performed in an environment which is identical to the blue production environment, thus eliminating the need for a separate preproduction environment. When ready, the route to the production environment is flipped at the router so that production traffic will begin to be directed to the green environment.

The now inactive blue environment can be kept in its existing state for a short period of time, to make it possible to rapidly switch back if anything goes wrong during or shortly after the update.

PaaS offers different mechanisms to create two logical environments on the fly. In fact, the blue/green environments coexist in the production environment and upgrades are rolled out by replacing instance by instance with newer versions. Although the same basics are applied, the entire procedure is controlled by code and can therefore be executed automatically, without human intervention. Furthermore the rolling ugprade mechanism is not only limited to services and containers (e.g., Kubernetes), but is also applicable for VMs (e.g., AWS Cloudforms).

Conclusion

Adopting a microservices architecture allows businesses to streamline delivery of stable, scalable applications that make more efficient use of cloud resources. Shifting from a monolith application toward microservices is an incremental process that requires and results in cultural change as well as architectural change as teams become smaller and more autonomous, with polyglot architectures and independent release cycles becoming more common.

- Adopting containers helps deal with increased complexity, and when combined with CI/CD introduced in the first stage of cloud-native evolution, results in faster development iterations to push high-quality services to production.
- Identifying and mastering complexity of application dependencies remain key challenges during this phase of cloud-native evolution.
- Monitoring can help to better understand dependencies while applications are being migrated to a microservices architecture, and to ensure that applications continue to operate effectively throughout the process.

Case Study: Prep Sportswear

Prep Sportswear is a Seattle-based textile printing and ecommerce company. It produces apparel with custom logos and mascots for thousands of US schools, colleges, sports clubs, the military, and the

government. Its online marketplace allows customers to design and purchase products related to their favorite schools or organizations.

In 2014, Preps Sportswear had a serious scaling and quality problem. On peak volume days like Black Friday or Cyber Monday, it took up to 22 hours just to process the incoming orders before the production and fulfillment process even started. On top of that, the company faced regular breakdowns which required the lead software engineer in Seattle to be on call around the clock. During one season in particular, the software engineer's phone rang 17 days in a row at 3 o'clock in the morning because something was wrong with the platform and the production facility came to a halt!

This was detrimental to the goal of shipping an online order within four days. Preps Sportswear did not have preprinted products on the shelf. Clearly something had to be done to make the software platform faster, more scalable, and less error prone. In 2015, the engineering team decided to move to a cloud-based technology stack with microservices. This process involved dealing with technical debt in their existing monolithic application, restructuring the development team to implement an Agile approach, and adopting new bug-tracking and CD tools to integrate with the existing technology stack:

Monoliths as technical debt
: The developers began with a custom-built monolith with 4.5 million lines of code, all written from the ground up by an external vendor. It included the ecommerce platform, the Enterprise Resource Planning (ERP) system, manufacturing device integration, and design pattern manipulation. And it represented a huge technical debt because all of the original developers had left, documentation was poor, and some of the inline comments were in Russian.

Agile teams
: To put the new vision into action, Prep Sportswear needed to lay the groundwork by restructuring the development team and implementing an Agile DevOps approach to software engineering. The new team is very slim, with only 10 engineers in software development, 1 in DevOps, and 2 in operations.

Technology stack
: The old bug tracking tool was replaced by JIRA, and TeamCity became the team's CD technology. It integrated nicely with its

technology stack. The application is written in C# and .NET. For the cloud infrastructure, the team chose OpenStack.

Converting the large monolithic application into a modern, scalable microservice architecture turned out to be tricky and it had to be done one step at a time. Here are two examples of typical challenges:

- The first standalone service was a new feature; an inventory system. Traditionally, Prep Sportswear had produced just-in-time, but it turned out that the fulfillment process would be faster with a small stock and a smart inventory system. The main challenge was the touchpoint with the rest of the monolith. At first, tracking inventory against orders worked just as intended. But there was another part of the system outside of the new service that allowed customers to cancel an order and replace it with a new request. This didn't happen often, but it introduced serious errors into the new service.
- A second example was pricing as part of a new purchasing cart service. The price of a shirt, for example, depended on whether logos needed to be printed with ink, embroidered, or appliquéd. For embroidery works, the legacy system called the design module to retrieve the exact count of stitches needed, which in turn determined the price of the merchandise. This needed to be done for every item in the cart and it slowed down the service tremendously.

In the monolith, the modules were so entangled that it was difficult to break out individual functionalities. The standard solution to segregate the existing data was technically not feasible. Rather, a set of practices helped to deal with the technical debt of the legacy system:

- Any design of a new microservice and its boundaries started with a comprehensive end-to-end analysis of business processes and user practices.
- A solid and fast CD system was crucial to deploy resolutions to unforeseen problems fast and reliably.
- An outdated application monitoring tool was replaced with a modern platform that allowed the team to oversee the entire software production pipeline, perform deep root-cause analysis, and even helped with proactive service scaling by using trend data.

Key Takeaways

The transition from a monolith to microservices is challenging because new services can break when connected with the legacy system. However, at Prep Sportswear the overall system became significantly more stable and scalable. During the last peak shopping period the system broke only once. The order fulfillment time went down from six to three days and the team is already working on the next goal—same day shipping.

CHAPTER 4

Dynamic Microservices

After basic microservices are in place, businesses can begin to take advantage of cloud features. A dynamic microservices architecture allows rapid scaling-up or scaling-down, as well as deployment of services across datacenters or across cloud platforms. Resilience mechanisms provided by the cloud platform or built into the microservices themselves allow for self-healing systems. Finally, networks and datacenters become software defined, providing businesses with even more flexibility and agility for rapidly deploying applications and infrastructure.

Scale with Dynamic Microservices

In the earlier stages of cloud-native evolution, capacity management was about ensuring that each virtual server had enough memory, CPU, storage, and so on. Autoscaling allows a business to scale the storage, network, and compute resources used (e.g., by launching or shutting down instances) based on customizable conditions.

However, autoscaling on a cloud instance–level is slow—too slow for a microservices architecture for which dynamic scaling needs to occur within minutes or seconds. In a dynamic microservices environment, rather than scaling cloud instances, autoscaling occurs at the microservice level. For example for a service with low-traffic, only two instances might run, and be scaled up at load-peak time to seven instances. After the load-peak, the challenge is to scale the service down again; for example, back down to two running instances.

In a monolithic application, there is little need for orchestration and scheduling; however, as the application begins to be split up into separate services, and those services are deployed dynamically and at scale, potentially across multiple datacenters and cloud platforms, it no longer becomes possible to hardwire connections between the various services that make up the application. Microservices allow businesses to scale their applications rapidly, but as the architecture becomes more complex, scheduling and orchestration increase in importance.

Service Discovery and Orchestration

Service discovery is a prerequisite of a scalable microservices environment because it allows microservices to avoid creating hardwired connections but instead for instances of services to be discovered at runtime. A service registry keeps track of active instances of particular services—distributed key-value stores such as etcd and Consul are frequently used for this purpose.

As services move into containers, container orchestration tools such as Kubernetes coordinate how services are arranged and managed at the container level. Orchestration tools often provide built-in service automation and registration services. An API Gateway can be deployed to unify individual microservices into customized APIs for each client. The API Gateway discovers the available services via service discovery and is also responsible for security features; for example, HTTP throttling, caching, filtering wrong methods, authentication, and so on.

Just as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Containers as a Service (CaaS) provide increasing layers of abstraction over physical servers, the idea of liquid infrastructure applies virtualization to the infrastructure. Physical infrastructure like datacenters and networks are abstracted by means of Software-Defined Datacenters (SDDCs) and Software-Defined Networks (SDNs), to enable truly scalable environments.

Health Management

The increased complexity of the infrastructure means that microservice platforms need to be smarter about dealing with (and preferably, avoiding) failures, and ensuring that there is enough redundancy and that they remain resilient.

Fault-tolerance is the the ability of the application to continue operating after a failure of one or more components. However, it is better to avoid failures by detecting unhealthy instances and shutting them down before they fail. Thus, the importance of monitoring and health management rises in a dynamic microservices environment.

Monitoring

In highly dynamic cloud environments, nothing is static anymore. Everything moves, scales up or down dependent on the load at any given moment and eventually dies—all at the same time. In addition to the platform orchestration and scheduling layer, services often come with their own built-in resiliency (e.g., Netflix OSS circuit breaker). Every service might have different versions running because they are released independently. And every version usually runs in a distributed environment. Hence, monitoring solutions for cloud environments must be dynamic and intelligent, and include the following characteristics:

Autodiscovery and instrumentation
: In these advanced scenarios, static monitoring is futile—you will never be able to keep up! Rather, monitoring systems need to discover and identify new services automatically as well as inject their monitoring agents on the fly.

System health management
: Advanced monitoring solutions become *system health management tools*, which go far beyond the detection of problems within and individual service or container. They are capable of identifying dependencies and incompatibilities between services. This requires transaction-aware data collection for which metrics from multiple ephemeral and moving services can be mapped to a particular transaction or user action.

Artificial intelligence
: Machine learning approaches are required to distinguish, for example, a killed container as a routine load balancing measure from a state change that actually affects a real user. All this requires a tight integration with individual cloud technologies and all application components.

Predictive monitoring
: The future will bring monitoring solutions that will be able to predict upcoming resource bottlenecks based on empirical evidence and make suggestions on how to improve applications and architectures. Moving toward a closed-loop feedback system, monitoring data will be used as input to the cloud orchestration and scheduling mechanisms and allow a new level of dynamic control based on the health and constraints of the entire system.

Enabling Technologies

As the environment becomes more dynamic, the way that services are scaled also needs to become more dynamic to match.

Load Balancing, Autoscaling, and Health Management

Dynamic load balancing involves monitoring the system in real time and distributing work to nodes in response. With traditional static load balancing, after work has been assigned to a node, it can't be redistributed, regardless of whether the performance of that node or availability of other nodes changes over time. Dynamic load-balancing helps to address this limitation, leading to better performance; however, the downside is that it is more complex.

Autoscaling based on metrics such as CPU, memory, and network doesn't work for transactional apps because these often depend on third-party services, service calls, or databases, with transactions that belong to a session and usually have state in shared storage. Instead, scaling based on the current and predicted load within a given timeframe is required.

The underlying platform typically enables health management for deployed microservices. Based on these health checks, you can apply failover mechanisms for failing service instances (i.e., containers), and so the platform allows for running "self-healing systems." Beyond platform health management capabilities, the microservices might also come with built-in resilience. For instance, a microservice might implement the Netflix OSS components—open source libraries and frameworks for building microservices at scale —to automate scaling cloud instances and reacting to potential service outages. The Hystrix fault-tolerance library enables built-in "circuit breakers" that trip when failures reach a threshold. The Hystrix cir-

cuit makes service calls more resilient by keeping track of each endpoint's status. If Hystrix detects timeouts, it reports that the service is unavailable, so that subsequent requests don't run into the same timeouts, thus preventing cascading failures across the complete microservice environment.

Container Management

Container management tools assist with managing containerized apps deployed across environments (Figure 4-1). Of the 139 respondents to this question in the Cloud Platform Survey, 44 percent don't use a container management layer. The most widely adopted management layer technologies were Mesosphere (19 percent of respondents to this question) and Docker Universal Control Pane (15 percent). Rancher was also used. Let's take a look at these tools:

Mesosphere
: Mesosphere is a datacenter-scale operating system that uses Marathon orchestrator. It also supports Kubernetes or Docker Swarm.

Docker Universal Control Pane (UCP)
: This is Docker's commercial cluster management solution built on top of Docker Swarm.

Rancher
: Racher is an open source platform for managing containers, supporting Kubernetes, Mesos, or Docker Swarm.

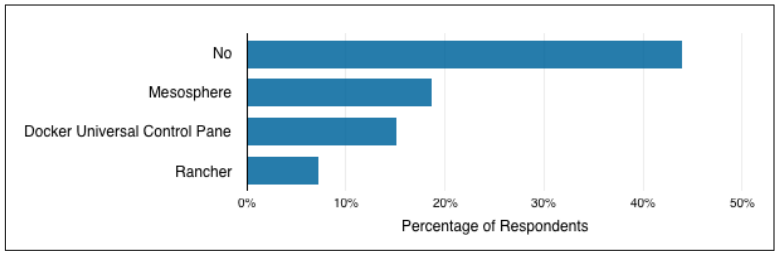


Figure 4-1. Top container management layer technologies

SDN

Traditional physical networks are not agile. Scalable cloud applications need to be able to provision and orchestrate networks on demand, just like they can provision compute resources like servers

and storage. Dynamically created instances, services, and physical nodes need to be able to communicate with one another, applying security restrictions and network isolation dynamically on a workload level. This is the premise of SDN: with SDN, the network is abstracted and programmable, so it can be dynamically adjusted in real-time.

Hybrid SDN allows traditional networks and SDN technologies to operate within the same environment. For example, the OpenFlow standard allows hybrid switches—an SDN controller will make forwarding decisions for some traffic (e.g., matching a filter for certain types of packets only) and the rest are handled via traditional switching.

Forty-seven percent of 138 survey respondents to this question are not using SDNs (Figure 4-2). Most of the SDN technologies used by survey respondents support connecting containers across multiple hosts.

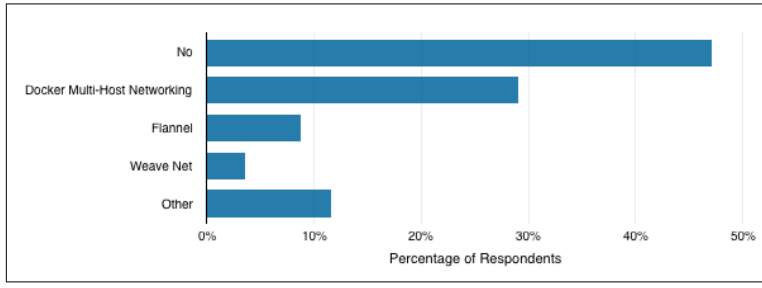


Figure 4-2. SDN Adoption

Overlay Networks

In general, an overlay network is a network built on top of another network. In the context of SDN, overlay networks are virtual networks created over the top of a physical network. Overlay networks offer connectivity between workloads on different hosts by establishing usually unencrypted tunnels between the workloads. This is accomplished by use of an encapsulation protocol (e.g., VXLAN and GRE) on the physical network. The workloads use virtual network interfaces to connect to the NIC of the host.

Docker's Multi-Host Networking was officially released with Docker 1.9 in November 2015. It is based on SocketPlane's SDN technology. Docker's original address mapping functionality was very rudimentary and did not support connecting containers across multiple hosts, so other solutions including WeaveNet, Flannel, and Project Calico were developed in the interim to address its limitations. Despite its relative newness compared to the other options, Docker Multi-Host Networking was the most popular SDN technology in use by respondents to the Cloud Platform Survey (Figure 4-2)—29 percent of the respondents to this question are using it. Docker Multi-Host Networking creates an overlay network to connect containers running on multiple hosts. The overlay network is created by using the Virtual Extensible LAN (VXLAN) encapsulation protocol.

A distributed key-value store (i.e., a store that allows data to be shared across a cluster of machines) is typically used to keep track of the network state including endpoints and IP addresses for multi-host networks, for example, Docker's Multi-Host Networking supports using Consul, etcd, or ZooKeeper for this purpose.

Flannel (previously known as Rudder), is also designed for connecting Linux-based containers. It is compatible with CoreOS (for SDN between VMs) as well as Docker containers. Similar to Docker Multi-Host Networking, Flannel uses a distributed key-value store (etcd) to record the mappings between addresses assigned to containers by their hosts, and addresses on the overlay network. Flannel supports VXLAN overlay networks, but also provides the option to use a UDP backend to encapsulate the packets as well as host-gw, and drivers for AWS and GCE. The VXLAN mode of operation is the fastest option because of the Linux kernel's built-in support for VxLAN and support of NIC drivers for segmentation offload.

Weave Net works with Docker, Kubernetes, Amazon ECS, Mesos and Marathon. Orchestration solutions like Kubernetes rely on each container in a cluster having a unique IP address. So, with Weave, like Flannel, each container has an IP address, and isolation is supported through subnets. Unlike Docker Networking, Flannel, and Calico, Weave Net does not require a cluster store like etcd when using the weavemesh driver. Weave runs a micro-DNS server at each node to allow service discovery.

Another SDN technology that some survey participants use is Project Calico. It differs from the other solutions in the respect that it is a pure Layer 3 (i.e., Network layer) approach. It can be used with any kind of workload: containers, VMs, or bare metal. It aims to be simpler and to have better performance than SDN approaches that rely on overlay networks. Overlay networks use encapsulation protocols, and in complex environments there might be multiple levels of packet encapsulation and network address translation. This introduces computing overhead for de-encapsulation and less room for data per network packet because the encapsulation headers take up several bytes per packet. For example, encapsulating a Layer 2 (Data Link Layer) frame in UDP uses an additional 50 bytes. To avoid this overhead, Calico uses flat IP networking with virtual routers in each node, and uses the Border Gateway Protocol (BGP) to advertise the routes to the containers or VMs on each host. Calico allows for policy based networking, so that you can containers into schemas for isolation purposes, providing a more flexible approach than the CIDR isolation supported by Weave, Flannel, and Docker Networking, with which containers can be isolated only based on their IP address subnets.

SDDC

An SDDC is a dynamic and elastic datacenter, for which all of the infrastructure is virtualized and available as a service. The key concepts of the SDDC are server virtualization (i.e., compute), storage virtualization, and network virtualization (through SDNs). The end result is a truly "liquid infrastructure" in which all aspects of the SDDC can be automated; for example, for load-based datacenter scaling.

Conclusion

Service discovery, orchestration, and a liquid infrastructure are the backbone of a scalable, dynamic microservices architecture.

- For cloud-native applications, everything is virtualized—including the computer, storage, and network infrastructure.
- As the environment becomes too complex to manage manually, it becomes increasingly important to take advantage of automa-

ted tools and management layers to perform health management and monitoring to maintain a resilient, dynamic system.

Case Study: YaaS—Hybris as a Service

Hybris, a subsidiary of SAP, offers one of the industry's leading ecommerce, customer engagement, and product content management systems. The existing Hybris Commerce Suite is the workhorse of the company. However, management realized that future ecommerce solutions needed to be more scalable, faster in implementing innovations, and more customer centered.

In early 2015, Brian Walker, Hybris and SAP chief strategy officer, introduced YaaS—Hybris-as-a-Service.[1] In a nutshell, YaaS is a *microservices marketplace* in the public cloud where a consumer (typically a retailer) can subscribe to individual capabilities like the product catalog or the checkout process, whereas billing is based on actual usage. For SAP developers, on the other hand, it is a platform for publishing their own microservices.

The development of YaaS was driven by a vision with four core elements:

Cloud first
: Scaling is a priority.

Retain development speed
: Adding new features should not become increasingly difficult; the same should hold true with testing and maintenance.

Autonomy
: Reduce dependencies in the code and dependencies between teams.

Community
: Share extensions within our development community.

A core team of about 50 engineers is in charge of developing YaaS. In addition, a number of globally distributed teams are responsible for developing and operating individual microservices. Key

1 YaaS stands for SAP Hybris-as-a-Service on SAP HANA Cloud Platform. Because the logo of Hybris is the letter "Y" the acronym becomes Y-a-a-S.

approaches and challenges during the development and operation of the YaaS microservices include the following:

Technology stack
: YaaS uses a standard IaaS and CloudFoundry as PaaS. The microservices on top can include any technologies the individual development teams choose, as long as the services will run on the given platform and exposes a RESTful API. This is the perfect architecture for process improvements and for scaling services individually. It enables high-speed feature delivery and independence of development teams.

Autonomous teams and ownership
: The teams are radically independent from one another and chose their own technologies including programing languages. They are responsible for their own code, deployment, and operations. A microservice team picks its own CI/CD pipeline whether it is Jenkins, Bamboo, or TeamCity. Configuring dynamic scaling as well as built-in resilience measures (like NetflixOSS technologies) also fall into the purview of the development teams. They are fully responsible for balancing performance versus cost.

Time-to-market
: This radical decoupling of both the microservices themselves and the teams creating them dramatically increased speed of innovation and time-to-market. It takes only a couple days from feature idea, to code, to deployment. Only the nonfunctional aspects like documentation and security controls take a bit longer.

Managing independent teams
: Rather than long and frequent meetings (like scrum of scrums) the teams are managed by objectives and key results (OKR). The core team organized a kick-off meeting and presented five top-level development goals. Then, the microservice teams took two weeks to define the scope of their services and created a roadmap. When that was accepted, the teams worked on their own until the next follow-up meeting half a year later. Every six weeks all stakeholders and interested parties are invited to a combined demo, to see the overall progress.

Challenges

The main challenge in the beginning was to slim down the scope of each microservice. Because most engineers came from the world of big monoliths, the first microservices were hopelessly over-engineered and too broad in scope. Such services would not scale, and it took a while to establish a new mindset. Another challenge was to define a common API standard. It took more than 100 hours of tedious discussions to gain consensus on response codes and best practices, but it was time well spent.

Key Takeaways

The digital economy is about efficiency in software architecture.[2] A dynamically scalable microservice architecture goes hand in hand with radical organizational changes toward autonomous teams who own a service end to end.

2 Stubbe, Andrea, and Philippe Souidi. "Microservices—a game changer for organizations." Presented at API: World 2016, San Jose.

CHAPTER 5
Summary and Conclusions

Becoming cloud-native is about agile delivery of reliable and scalable applications. Businesses migrating to the cloud typically do so in three stages:

- The first stage involves migrating existing applications to virtualized infrastructure—initially to Infrastructure as a Service (IaaS) with a lift-and-shift approach and implementation of an automated Continuous Integration/Continuous Delivery (CI/CD) pipeline to speed up the release cycle.
- In phase two, monolithic applications begin to move toward microservices architectures with services running in containers on Platform as a Service (PaaS).
- In Phase 3, businesses begin to make more efficient use of cloud technologies by shifting toward dynamic microservices.

A dynamic microservices architecture enables businesses to rapidly scale applications on demand and improves their capability to recover from failure; however, as a consequence, application environments become more complex and hence more difficult to manage and understand.

At each step along this journey, the importance of scheduling, orchestration, autoscaling and monitoring increases. Taking advantage of tooling to automate these processes will assist in effectively managing dynamic microservice environments.

APPENDIX A
Survey Respondent Demographics

About half (49 percent) of the respondents to the Cloud Platform Survey work in the IT industry (Figure A-1).

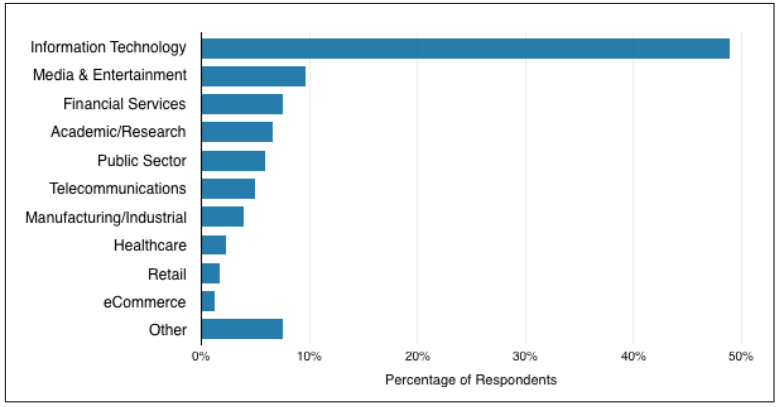


Figure A-1. Industry

The majority of survey responses (51 percent) came from people located in North America (Figure A-2), with 35 percent from Europe.

Figure A-3 shows respondents came from a range of company sizes (in number of employees). Twenty-two percent are from companies with 9 employees or fewer, 24 percent from companies with 10 to 99 employees, 20 percent from 100 to 999, 18 percent from companies of 1,000 to 9,999, and 9 percent from companies with 10,000 to 99,999 employees.

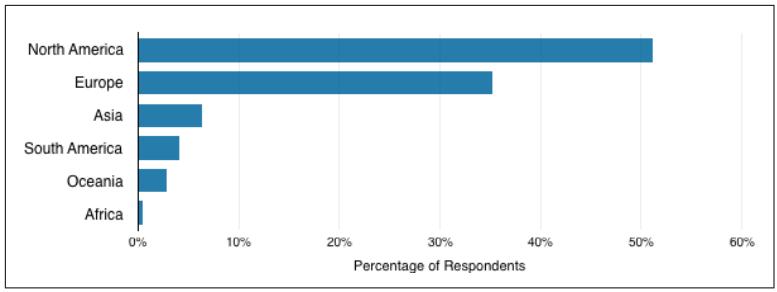


Figure A-2. Geographic region

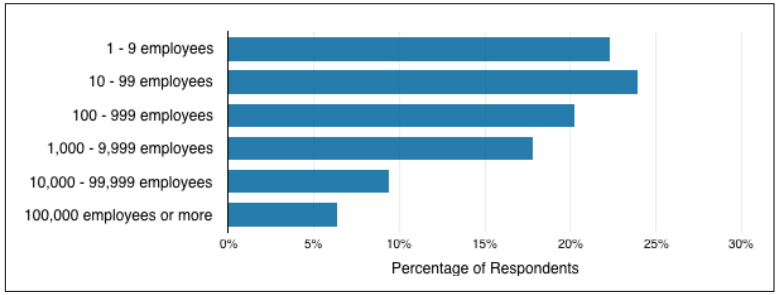


Figure A-3. Company size

Respondents identified as being software developers (27 percent), software/cloud architects (22 percent), or in IT operations roles (17 percent) (Figure A-4).

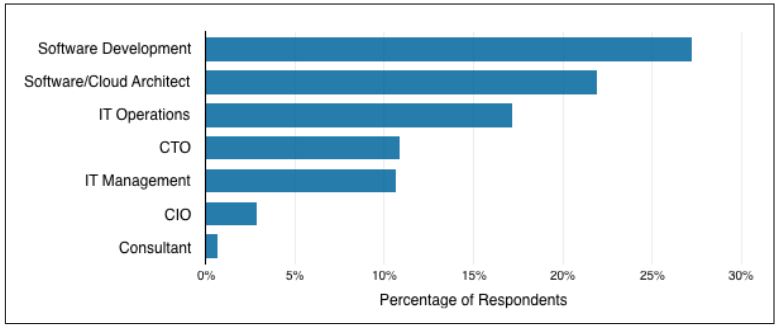


Figure A-4. Respondent's role in company

The top roles responsible for infrastructure and platform issues at the respondents' companies included IT operations (53 percent) and DevOps (41 percent) (Figure A-5), with only a handful (less than 2 percent) of respondents nominating the development team as being responsible.

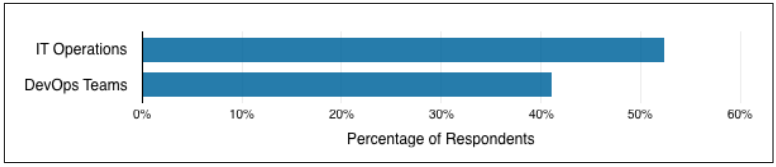


Figure A-5. Who is responsible for infrastructure and platform issues in your company?

APPENDIX B

Case Study: Banco de Crédito del Perú

Banco de Crédito del Perú (BCP) celebrated its 125th anniversary in 2014. Giafranco Ferrari, VP of retail, and the executive team used this occasion to look ahead into the next decades and decided to take a bold step toward becoming a digital business.[1] BCP's digital transition is representative of many similar efforts not only in the banking industry, but also in the insurance, retail, healthcare, software, production, and government sectors.

BCP realized that the majority of its fast-growing customer base were young digital natives who expected to interact with their bank on any device at any time and have the same experience as with leading digital services like Netflix, Facebook, Twitter, and the like. This was in stark contrast to a world where banking was done in person by walking into an office and talking to a clerk. Now the transactions take place in the the digital space. And the electronic systems of the past, which were used by the bank employees, needed to be rebuilt from scratch to serve customers directly.

1 *http://bit.ly/2fd37EQ*

The digital transition team in Lima prepared for nine months to start its first project of a new era. Here are the main actions the team took:

Define scope and digitization
: The team began by identifying the clients' needs and their pain points, ran some economics, and then decided that the digital opening of savings accounts was the right scope.

Implement Agile methods
: The team began building small teams that adopted Agile methods, in contrast to the waterfall methods previously used.

Involving customers and stakeholders
: Customers were involved throughout the development process, as was every department in the bank. For example, the legal department was part of the process from the very beginning, instead of asking them to apply regulative frameworks after the software was already built.

Migrate to the cloud
: The development team created a mobile application with a backend running in the Amazon cloud and a first version of a continuous delivery pipeline with integrated automated tests. Fast release cycles: The first working release that could be tested with actual clients was completed in only four months! Creating a new product from design to completion in such short time was unthinkable before this effort.

Key takeaways

Francesca Raffo, head of digital, points out that the two main ingredients for the project were the digitization process and culture change. And the key success factor was top-level management support because the new approach changed "the DNA of the organization."

About the Authors

Alois Mayr is Technology Lead for Cloud and Containers with the Dynatrace Innovation Lab. He works on bringing full-stack monitoring to cloud and PaaS platforms such as Docker and Cloud Foundry. In his role as technology lead, he works very closely with R&D and customers and supports them with their journeys to those platforms. He is also a blogger and speaker at conferences and meetups, where he shares lessons learned and other user stories with cloud platforms. Before joining Dynatrace, he was a researcher focused on software quality measurements and evaluation.

Peter Putz is the Operations Lead of the Dynatrace Innovation Lab. He and his team spearhead technology research and business development efforts for next generation Digital Performance Monitoring solutions. Peter holds a PhD in social and economic sciences from the Johannes Kepler University Linz, Austria. He has managed, developed, and operated intelligent enterprise systems for more than 15 years. Before joining Dynatrace, he was a computer and management scientist with the NASA Ames Research Center and the Xerox Palo Alto Research Center (PARC).

Dirk Wallerstorfer is Technology Lead for OpenStack and SDN at Dynatrace. He has 10+ years of deep, hands-on experience in networking, security, and software engineering. Dirk spends his days researching new and emerging technologies around virtualization of infrastructure and application environments. He is passionate about making things fast and easy and likes to share his experiences through blog posts, and during his speaking engagements at conferences and meetups. Before joining Dynatrace, he built up and led a quality management team and worked as a software engineer.

Anna Gerber is a full-stack developer with more than 15 years of experience in the university sector. A senior developer at the Institute for Social Science Research at The University of Queensland, Australia, and Leximancer Pty Ltd, Anna was formerly a technical project manager at UQ ITEE eResearch specializing in Digital Humanities, and research scientist at the Distributed System Technology Centre (DSTC). In her spare time, Anna enjoys tinkering with and teaching about soft circuits, 3D printing, and JavaScript robotics.