

Cloud Database Trend Report

Migration toward safer, secure data management

BROUGHT TO YOU IN PARTNERSHIP WITH



Table of Contents

- 3** Highlights and Introduction
BY MELISSA HABIT
- 4** Key Research Findings
BY MATT LEGER
- 11** Leaders in Cloud Database
BY LINDSAY SMITH
- 16** Ensuring SQL Server High Availability in the Cloud
BY DAVE BIRMINGHAM
- 21** Data Safety in Cloud-Based Databases
BY GRANT FRITCHEY
- 28** Diving Deeper Into Cloud Databases

To sponsor a Trend Report:

Call: (919) 678-0300

Email: sales@devada.com

Highlights and Introduction

By [Melissa Habit](#), Publications Manager at DZone

Trends in cloud data storage continue to accelerate at a rapid pace. Now more than ever, organizations must evaluate their current and future data storage needs to find solutions that align with business goals. While cloud databases are relatively new to the scene, they show tremendous prospect in securing and managing data.


In selecting our topic for this Trend Report, we found the amount of promise and advancement in the space to be unparalleled. This report highlights DZone's original research on cloud databases and contributions from the community, as well as introduces new offerings within DZone Trend Reports.

While you may know her as your friendly Java Zone copy editor, Lindsay Smith has stepped into the role of DZone's Publications Content Manager. Among many new endeavors, she's spearheading our new strategy for Executive Insights — a series we've titled, "Leaders in Tech," which serves to complement our original research. The series focuses on the viewpoints of industry frontrunners, tech evangelists, and DZone members who share their insights into research findings and outlooks for the future.

Through the lenses of diverse experiences and skillsets, we hope "Leaders in Tech" helps our readers expand their perspectives — and even embark on new journeys toward what's yet to be discovered.

Lindsay spoke with Adam Ballai, CEO of RevOps, to discuss Cloud Databases, and he imparted some key insight into the number one thing engineers need to know about cloud DBMSs: **the use case**. Adam also touched on a profound topic from our research — cost — which is reportedly a leading challenge in moving to a cloud database solution. Other challenges noted by DZone developers include data privacy compliance and data security. These concerns are explored further by Microsoft Data Platform MVP Grant Fritchey and Technical Evangelist Dave Bermingham.

In "Data Safety in Cloud-Based Databases," Grant gives us the low-down on issues including data breaches and cloud security. He opens by acknowledging the wide-spread skepticism of online data stores, and then explains why and how cloud database providers aren't **a way** to go but **the way** to go for maximum data safety today. Dave brings us full circle in "Ensuring SQL Server High Availability in the Cloud," throughout which he addresses high availability (HA) and disaster recovery (DR) solutions in mission critical SQL Server deployments, plus murmurs of a new cloud storage trend: emerging support for high-performance cloud file shares.

Read on for a first look at the state of cloud DBMSs, reasons why organizations are adopting cloud databases, and both the benefits and challenges of database migration for engineering teams across enterprises. 

Key Research Findings

By [Matt Leger](#), Principal Research Analyst at DZone

For decades, developers have had to work overtime with on-premise databases, working every day on database maintenance and administration. This approach worked before big data, analytics, and the Internet of Things (IoT) burst onto the scene, rendering on-premise database infrastructure incapable of handling the demand of modern data storage.

The early 2010s saw the entrance of cloud databases, a new solution enabling organizations to:

- Scale up operations when needed and scale back on demand.
- Save organizational data in the event of a disaster or system crash.
- Significantly cut back on database infrastructure (a very costly endeavor).
- Off load some database maintenance to an external provider.

It was a dream come true for many IT organizations.

[According to IBM](#), *"A cloud database is a database service built and accessed through a cloud platform. It serves many of the same functions as a traditional database with the added flexibility of cloud computing. Users install software on a cloud infrastructure to implement the database."*

So, what is the current state of cloud databases and what do developers need to know over the next year? In the DZone 2020 Cloud Database survey, we asked 416 developers about their experiences with cloud databases and what the future holds for this technology. This is what we learned.

KEY TAKEAWAYS

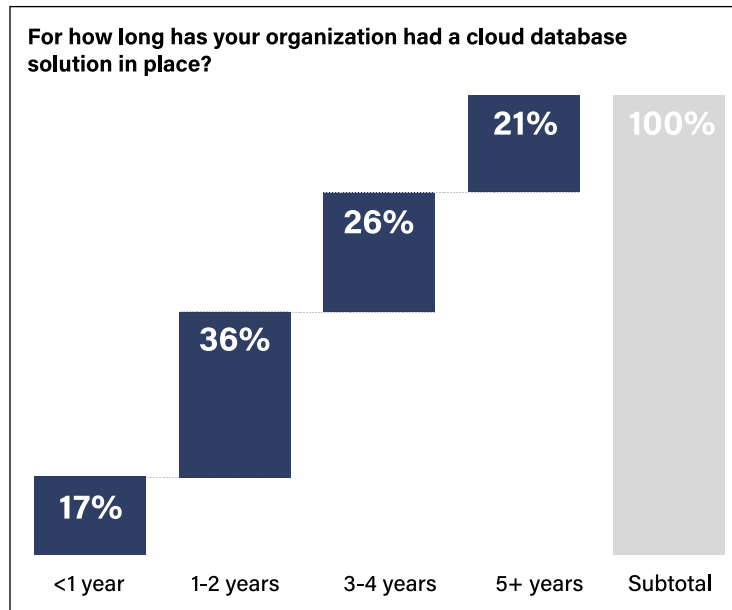
- ▶ Two-thirds of developers are now working, or moving toward working, with cloud database management systems (DBMSs) at their organization.
- ▶ In the SQL market, MySQL and PostgreSQL lead the way; in the NoSQL market, MongoDB dominates, followed by Redis.
- ▶ A majority of developers say that their organizations manage their cloud databases themselves. However, half of developers who use this method are not sure that this is the right approach.
- ▶ The leading challenges for organizations in moving to cloud DBMS were higher costs than anticipated, compliance with data privacy laws, data security, and changing mindsets to be open to cloud databases.
- ▶ Nearly half of developers believe that in the future, the use of NoSQL databases will match or even surpass cloud-based SQL databases.

The State of Cloud Database Management Systems

The days of building and maintaining on-premise database infrastructure are far from over. However, the day has come when just under two-thirds of developers are either currently working with (51%), or moving toward working with (18%), cloud DBMSs.

On the other hand, a quarter of developers say that their organizations are not moving to a cloud database solution just yet. If they change their mind, there is still time to jump on the cloud database bandwagon. The majority of developers currently using cloud databases (53%) work for organizations that are still new to using these tools (0-2 years) [see Figure 1]. Overall, there is a pretty even spread regarding the years of experience that these organizations have with cloud DBMSs. Surprisingly, these numbers are largely consistent across enterprise sizes, according to our survey.

Figure 1



NOT JUST ONE CLOUD DBMS SOLUTION, BUT TWO

Of the developers who currently use a cloud DBMS, about 40% use more than one at their organizations, and while another 26% do not yet use multiple ones, they are considering it. When asked why they use, or are considering using, multiple cloud DBMSs, half of developers said that it is because of the flexibility it affords them to use different data models based on business needs. Another quarter said that it is to optimize performance by choosing the best vendor for specific apps or use cases. We were surprised to learn that only 10% of developers said that they use multiple cloud DBMSs to avoid vendor lock-in.

Why Are Organizations Moving Their Databases to the Cloud?

Enhancing analytics capabilities (e.g., data warehousing/lakes), moving to a SaaS business model, and modernizing existing apps were the leading business purposes for migrating databases to the cloud [see Table 1]. Very few developers believe that their organizations do so in an effort to close a workforce gap (i.e., talent or skills shortage).

(see next page for table)

Table 1

For which of the following purposes did your organization move, or is considering moving its databases to the cloud?	
Analytics (i.e. data warehousing, data lakes)	52%
Moving to a SaaS model	47%
To modernize existing apps	46%
To support business services	42%
To support customer facing apps	41%
To support our efforts to become Cloud Native	37%
To store transactional data	36%
To close a gap in the workforce (either lack of skills or lack of employees)	14%
Other	5%

SQL vs. NoSQL in the Cloud

So, which type of database dominates the cloud DBMS market? About half of developers in our survey said that their organizations have migrated both types of databases to the cloud. For those that use one cloud provider, however, relational (41%) far surpassed non-relational (11%). Which vendors are the leaders in each market, you might ask? MySQL and PostgreSQL are clear winners in the SQL market [see Figure 2]. Meanwhile, MongoDB largely dominates the NoSQL market, followed by Redis [see Figure 3]:

Figure 2

Leading SQL Vendors			
MySQL	PostgreSQL	Microsoft SQL Server	Oracle Database
52%	47%	36%	35%

Figure 3

Leading NoSQL Vendors			
mongoDB	Redis	Amazon DynamoDB	Cassandra
58%	40%	24%	22%

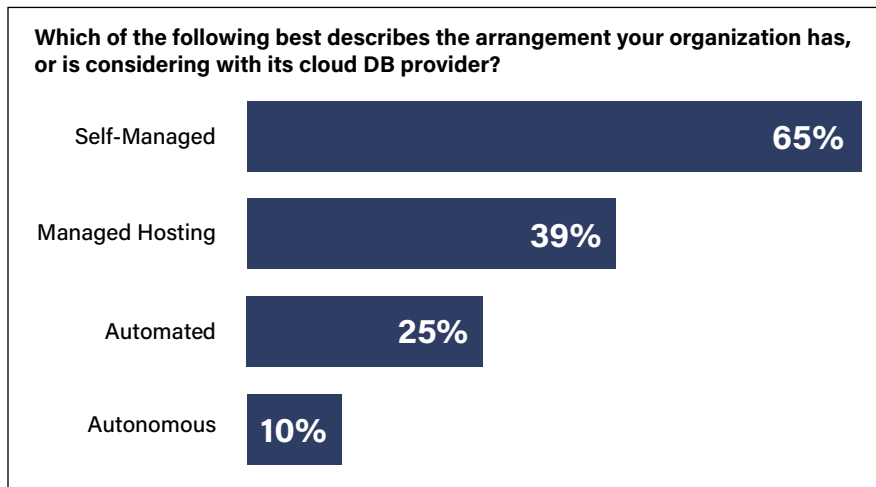
As far as the deployment model/environment of choice, more than twice as many developers say that they leverage the Database as a Service (DBaaS) model (40%) compared to those who use traditional or virtual machine image (21%). However, a third of developers say that they use a mix of both. So, with a slight edge in favor of DBaaS, there is really no clear winner yet.

Self-Managed Databases Lead the Way, but for How Long?

Perhaps what is most interesting is the breakdown between the Cloud Computing Arrangements that IT organizations have with their cloud database providers. There are four primary options: self-managed, managed hosting, automated databases, and autonomous databases. As of today, by far the leading is self-managed, suggesting that many developers are not quite ready to trust vendors with full management, maintenance, and control of their databases.

That being said, managed hosting and automated databases are making headway as organizations begin to realize the benefits from offloading some of the burden of database management, maintenance, and operations.

Figure 4



While developers may not be ready to give up control of their database, the data from our survey suggests that may change. When asked if the arrangement they have with their cloud database provider is the best option for them, less than half (46%) of developers agreed, and just as many were either not happy with their arrangement (10%) or unsure (34%) if it is the right one. Breaking down these responses by arrangement type clearly indicates that the more work developers do dealing with database related issues, the more uncertain they feel [see Table 2]. Although it appears that automated and autonomous databases are most favored, it is important to note that the number of developers using these methods today (according to our survey) is relatively small. As more organizations adopt automated and autonomous databases moving forward, time will tell if these results are representative of the DZone developer population.

Table 2

Developers who think the arrangement they have with their cloud DB provider is the best option by arrangement type				
Best Option?	Self-Managed	Managed Hosting	Automated	Autonomous
Yes	40%	57%	48%	63%
No	11%	11%	15%	11%
Unsure	38%	25%	25%	20%

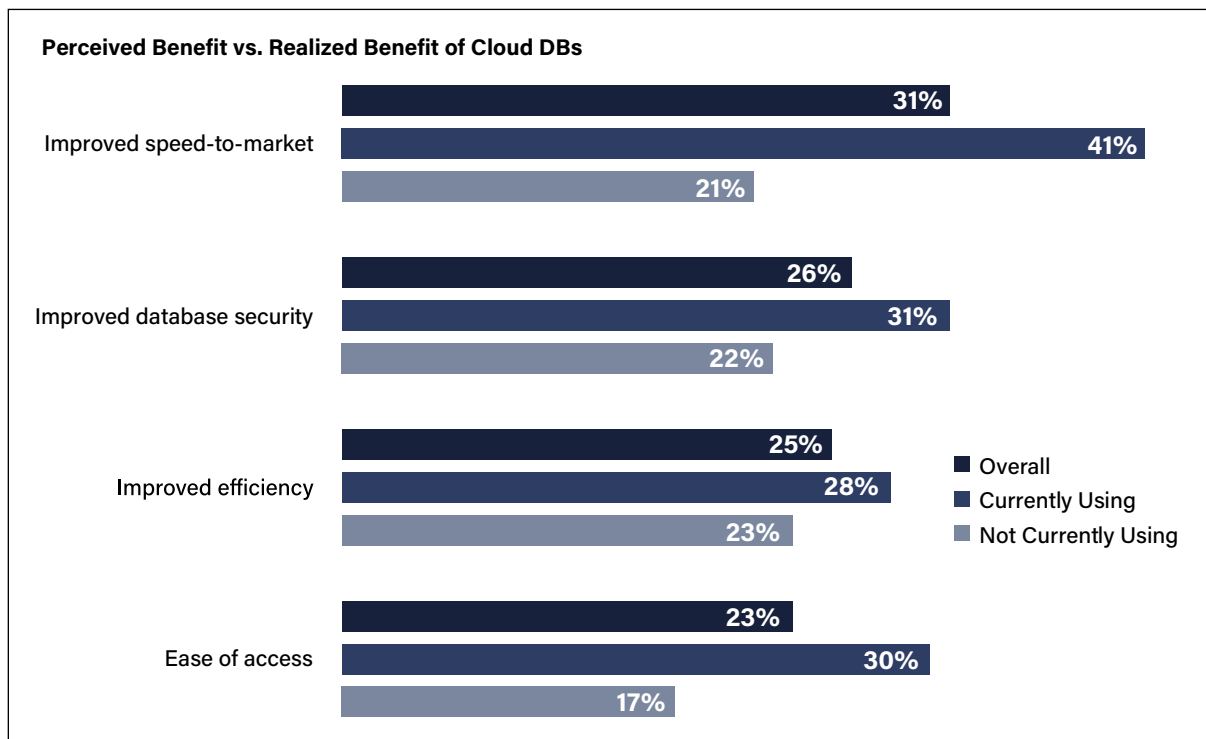
The Benefits of Cloud DBMSs to the Organization

By far, developers believe that the leading benefits of cloud databases, whether they currently use them or not, are

elasticity/scalability (74%) and data loss resilience (42%), which is unsurprising given that these were the core principles upon which cloud databases were built.

Beyond that, a third of developers overall believe that using cloud databases can improve speed to market [see Figure 5]. However, when we compared the answers of those who currently use cloud DBMS to those who do not, twice as many developers believed these tools helped increase speed to market. Seeing is truly believing with cloud DBMS. Similarly, the number of developers who believe that databases are more secure in the cloud jumped 9% when we filtered by those who currently use cloud DBMSs.

Figure 5



We also asked developers who currently use cloud DBMSs to estimate the amount of time in their workday that they saved from dealing with database-related issues since migrating over. Roughly 30% of developers said that they saved upwards of 20% of their time, and another 30% say that they saved between 21-40%. Very few saved more time than that and, unfortunately, for 16% of developers in our survey, no time was saved.

Challenges with Cloud DBMSs: Perception vs. Reality

While the movement to cloud DBMSs provides many benefits to an organization, it does not come without challenges. The first major challenge is complying with data privacy laws such as the General Data Protection Regulation (GDPR) [see Table 3]. Additionally, securing data during migration to the cloud is also a major challenge and can cause many headaches for developers. The perception of how great these challenges are, however, can differ significantly between developers who currently use a cloud DBMS and those who do not, as shown in the table below.

Those who do not use, but plan to adopt a cloud DBMS, should note that “higher costs than anticipated” was identified as a major challenge more often by developers who currently use cloud DBMSs than those who do not use them. This suggests that engineering teams should leave room in their budgets for the unexpected when migrating databases to the cloud.

Higher costs and “changing tradition mindsets” were equally as challenging as data privacy compliance and data security for those who already use these tools. Far more developers who do not currently use cloud DBMSs felt that data privacy compliance and data security were significant challenges, suggesting that these concerns are a bit overblown.

Table 3

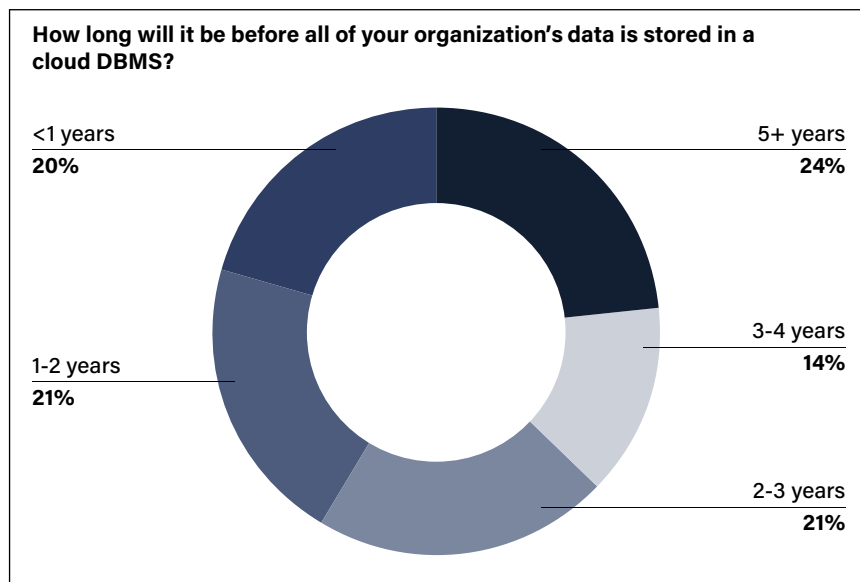
What was, or would be most challenging with migrating to a cloud DBMS?	Are you currently using a cloud DBMS?		
	Yes	No	% Difference
Compliance with data privacy laws and regulations	44%	59%	-15%
Securing the data	43%	56%	-13%
Changing traditional mindsets to be open to cloud DBs	42%	43%	-1%
Higher costs than anticipated	44%	37%	7%
Re-skilling employees	21%	21%	0%
Establishing and enforcing database governance	21%	19%	2%
Increased latency	16%	23%	-7%
Revising or re-negotiating licensing agreements	14%	18%	-4%
Data repatriation	14% ¹	8%	-5%
Other	3%	2%	1%

What's Next for Cloud Databases?

As noted above, the days of building and maintaining on-premise database infrastructure are far from over, but the use of cloud DBMSs is on the rise and beginning to dominate. In fact, more than half of developers in our survey (53%) believed that the day will come when all of their organization’s data will be stored in a cloud DBMS. About 1 in 5 could not bring themselves to make such a bold statement, unsure as of yet if the days of on-premise are doomed. The remaining respondents felt that there is no way that their organizations would ever give up all of their data to the cloud.

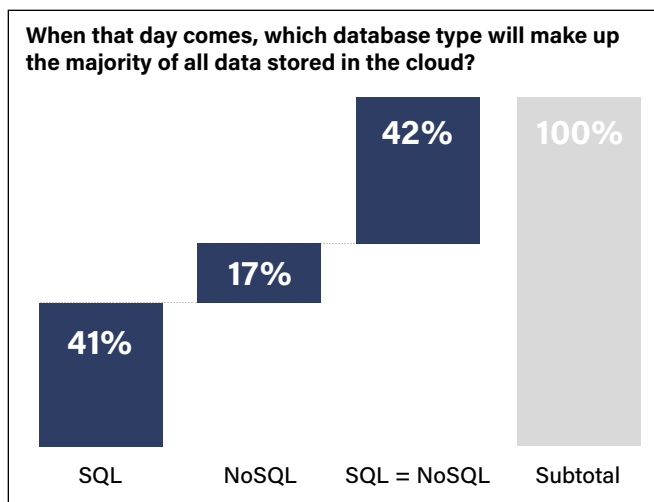
When asked to predict when the day will come that all of their organization’s data will be stored in a cloud DBMS, everyone had a different sense for when that might be. For some, the day will come sometime in 2020 — for others, the death of on-premise is 5 or more years out [see Figure 6].

Figure 6



And what about the future of cloud-based SQL or NoSQL databases? Everyone is bound to agree on what database type will dominate the cloud moving forward, right? Unfortunately, it was not that simple: 42% of developers who believe that cloud DBMSs will one day rule the land feel that the use of SQL and NoSQL databases will be equal [see Figure 7]. Roughly the same amount of developers believe that SQL will remain the dominant database type — the remaining, of course, chose NoSQL as the ultimate victor.

Figure 7



While no one agrees on this either, the data is significant. Historically, SQL/relational databases accounted for a significant majority of overall database usage. Organizations such as [Gartner and IDC have said](#) that, at least over the next few years, relational databases will continue to dominate:

"IDC forecasts that relational [databases] will still account for more than 80% of the total operational database market through 2022, and Gartner forecasts that through 2020, relational technology will continue to be used for at least 70% of new applications and projects."

Despite what the analysts at these organizations say, nearly half of developers in our survey felt differently. According to our survey, they believe that NoSQL will catch up to, or even surpass, SQL in the cloud before too long. Only time will tell whose prediction holds true. 🎲

Leaders in Cloud Database

Adam Ballai, CEO at RevOps, Discusses DZone's Key Research Findings

By [Lindsay Smith](#), Publications Content Manager at DZone

I sat down with Adam Ballai, co-founder and CEO of RevOps, to discuss findings from our research and, more broadly, the future of cloud databases. Not only was Adam able to discuss the benefits of using specific DBMS technologies, but he was able to provide a different lens — the lens of the customer.

Our research surveyed the DZone community — a community made of developers. But what good is a piece of software if it isn't meeting the needs, or demands, of its clients?

Adam was able to put this into perspective and highlight key goals across teams, whether marketing, engineering, Ops, or Sales, and why cloud databases are something that can benefit all parts of your organization.

In the transcript below, I provide key insights Adam shared with me — both about our research and his predictions for the entire database ecosystem. Let's take a look.

What are the primary reasons why organizations make the move to a cloud database management solution?

[For new companies]: "The reason why you would choose a cloud database over doing it yourself at a new company is that within five to 10 minutes, you can get a cloud database running, whereas installing something like MySQL or Postgres SQL, or the other sets of back-ends are going to require you to learn about configuration and containers and deploying that into your cloud."

[For older companies]: "As far as older types of companies, and decisions that they're making around why they would choose cloud databases, is more of a question about what their use case is. It lets us move more towards focusing more on the customer problem than staying on the infrastructure and trying to solve some of these infrastructure problems itself. And so, there's a good reason to take a look at these cloud database services out there."

What would you say are some of the biggest challenges for organizations as they make the move to a cloud database solution?

"I think, again, the biggest question is around use case. What are they trying to solve? And making sure that when they go down that model of making those decisions that they know what the one- to two-year roadmap looks like in terms of what their customers are going to opt for."

Not all data stores are understanding of trying to cover all areas of CAP, consistency and availability, and durability — information is something that they need to consider when they're putting it together. The other is around what it means from a disaster perspective. If something were to go wrong, how do these systems heal? How do they repair?

Take RedShift for example. If a node fails inside RedShift, RedShift actually deploys your data into every one of their nodes, or close to every one of their nodes. So, if something were to fail, there's another node to failover to actually collect and run the query on. And so, that's one of the big value propositions for something like that."

40% of respondents said that they are currently using multiple cloud database management solutions. Another 26% said that while they currently are not using multiple, they are considering it. So, why do you think there are so many organizations moving towards these multiple cloud database management solutions?

"I think the number one thing that people consider when it comes to a cloud database is cost. And these days, modern cloud providers — or any sort of large enterprise that's thinking about disaster recovery — in the event of something going down, they think about having a second cloud around that has that data in it, or has their infrastructure in it, so they might most likely have two or three accounts already with other cloud providers. So, when they do go shopping, they can think about the other providers — the other data stores that are out there.

It makes sense to pay attention to what's out there because if something supports a certain feature, it saves your use case's time and you can build a go-to-market faster. You're probably thinking: What's risky to the business? And time is a big risk in terms of go-to-market. So, you're going to choose those types of technologies that are going to be faster for you."

Our research actually found that only 10% of developers say that they are using an autonomous database solution. It's still very new, but where do you think the market for autonomous solutions is going and why?

"The challenge here for the market is if the technology that's used today be able to be automated the same way. Can it be automatically orchestrated, or does it require a new technology underneath the hood to actually make this simpler and automatic?


Databases move a lot slower than a cloud provider can with more stateless software, and it's also difficult for companies to move towards that direction. Because in data, what you want to think about is disaster and more of the chaos from an engineering perspective and if something were to go wrong.

Autonomy requires a lot of rigor, and you're going to find that older companies are less likely to adopt that autonomy unless they have a high amount of database administrators and where it's going be cost effective. Or, if you're a new company, and you don't have a DBA, you're probably never going to — you probably want a technology that's going to solve that because you have a real scaling challenge immediately in front of you.

But for the majority of the market that doesn't have that scaling challenge, where the software itself is today kind of takes care of what it needs to do in provision and capacity based upon a few ideas about what you do. There's not a whole lot of demand for it for systems that are very consistent on delivery. So, I think it's more of a question of: What's the market and who and where is it going to go?"

So, do you have any advice for developers moving to those new cloud database solutions?

“The most important thing that a new developer needs to learn about a new cloud database would be making sure that they know upfront the two to three most important use cases to what they’re building and how they would use that system. And, if they don’t have those answers yet, then before they decide on anything that they’re doing, they should go figure out those two things. That’s the same advice I’d give for choosing any database — regardless of the cloud provider.

And, the second piece of advice I’d give to them is measuring cost and understanding what it looks like at a small usage of their system. So, that way, they can at least have a good idea about what is the long-term decision here. If it’s going to get expensive real fast, then they need to understand how that meets up with the value of the use case that they’re delivering.” 

The Cloud is Changing How We Use Databases

By **Shane K. Johnson**, Senior Director of Product Marketing at MariaDB Corporation

It's one thing for a database to run in the cloud, many do. It's another for a database to take advantage of cloud infrastructure and services and for a database-as-a-service (DBaaS) to do it for the benefit of customers. The three big cloud service providers — Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure — all have DBaaS offerings, but they all fall short in two ways when it comes to taking full advantage of their clouds.

First, when it comes to running databases in the cloud, a cloud-native storage architecture is a must. While most databases and DBaaS offerings use block storage such as Amazon Elastic Block Storage (EBS), it's not ideal when it comes to databases with analytical workloads. For the latter, it would be more efficient and far less expensive to use object storage such as the Amazon Simple Storage Service (S3). Snowflake is a great example of this. It's a cloud-native data warehouse with disaggregate storage and compute. When running on AWS, the data is stored in S3 and queried using a separate, scalable compute — Amazon Elastic Compute Cloud (EC2).

Second, modern applications expect more from databases. They should be able to enrich transactions with analytics, interweaving transactions and analytics either within the same query or across multiple queries. However, DBaaS offerings available today support transactional and analytical workloads, but not both. For example, Amazon Relational Database Service (RDS) supports transactional workloads whereas Amazon Redshift and Snowflake support analytical workloads. However, some databases support hybrid transactional/analytical workloads by storing data in both row and columnar formats.

MariaDB Platform is the first database to implement a fully cloud-native storage architecture for hybrid transactional/analytical workloads, using block storage for transactions and object storage for analytics. And SkySQL, now available for public preview, is the first cloud-native DBaaS built from the ground up to run MariaDB Platform in the cloud — and on Kubernetes.

Learn more about MariaDB Platform: <https://mariadb.com/products/mariadb-platform/>

Try SkySQL today: <https://mariadb.com/skyview>

TREND PREDICTIONS

- ▶ Databases will run on multi-, hybrid- and inter-cloud infrastructure.
- ▶ Databases will optimize storage and query execution as they become cloud native.
- ▶ Databases will integrate with cloud services to offer new/enhanced capabilities.



THE SKY IS TRULY THE LIMIT

SKYSQL. The ultimate MariaDB Cloud is here.

EASY TO USE + POWERFUL

SkySQL is the first database-as-a-service (DBaaS) to unlock the full power of MariaDB Platform in the cloud. Engineered for mission-critical applications, SkySQL does away with common limitations and combines automation with the expertise and experience necessary to support production deployments in the cloud.

For a Tech Preview, visit:

mariadb.com/skyview

Ensuring SQL Server High Availability in the Cloud

By [Dave Bermingham](#), Technical Evangelist at SIOS Technology

Theoretically, the cloud seems tailor-made for ensuring high availability (HA) and disaster recovery (DR) solutions in mission critical SQL Server deployments. Azure, AWS, and Google have distributed, state-of-the-art data centers throughout the world. They offer a variety of SLAs that can guarantee virtual machine (VM) availability levels of 99.95% and higher.

But deploying SQL Server for HA or DR has always posed a challenge that goes beyond geographic dispersion of data centers and deep levels of hardware redundancy. Configuring your SQL Server for HA or DR involves building a Windows Server Failover Cluster (WSFC) that ensures not only the availability of different machines running SQL Server itself but also — and most importantly — the availability of storage holding the data in which SQL Server is interacting.

In a traditional WSFC, data may reside in a storage area network (SAN) or an SMB3 share accessible to all server nodes in the WSFC. But cloud storage cannot be shared in the same way as a traditional SAN. That limitation has forced organizations to use either a third-party approach that can overcome the shared storage constraints in Windows or embrace one of the two Windows-native approaches that work around the challenge of shared storage in different ways.

However, there are murmurs of a new storage trend in the cloud: emerging support for high-performance cloud file shares, which act in a manner similar to traditional on-prem file shares. The real question, though, is whether a cloud file share option, even a high performance one, presents a better path to ensuring HA and DR in the cloud.

HA and DS: The Numbers That Really Matter

When we talk about HA, we're talking about guaranteed availability of 99.99% or greater, and despite the data center build-outs described above, that's still a hard number to guarantee. It's also a number that you have to consider carefully.

You can cluster two or more VMs in separate racks in an Azure data center (in what's called an "availability set"), which you'll be guaranteed that at least one of those VMs will be available at least 99.95% of the time. Alternatively, Azure and AWS enable you to cluster VMs across *multiple* data centers (i.e. "availability zones"), which will get you an SLA guaranteeing the availability of at least one of the VMs at least 99.99% of the time.

However, those SLAs only guarantee the availability of a VM. They don't guarantee the availability of SQL Server or your SQL Server data. If a primary VM goes offline and your cluster fails over to a secondary VM, your secondary VM

needs to be running SQL Server *and* needs to be able to access the underlying database files for your operations to continue with minimal interruption. Ensuring the availability of SQL Server and the underlying data requires further configuration.

Configuring for Software Availability in the Cloud

So how do you ensure the high availability of SQL Server and your data in the cloud? If you want to rely on services that are native to Windows Server (as opposed to using a third-party product), you’ve only had two options to date. You can use Storage Spaces Direct (available in Windows Server Enterprise Edition 2016 and later) or you can create an AlwaysOn Availability Group (available in SQL Server 2012 Enterprise Edition and later).

Both approaches have advantages and disadvantages. Storage Spaces Direct (S2D) creates a *virtual* storage area network (SAN) in software that can be accessed by any of the VMs in a WSFC. This *sounds* like a cloud-based version of a traditional failover cluster. But because this cluster must be configured as an Availability *Set*, all the VMs in the cluster reside in the same data center. If the entire data center goes offline because of a disruptive event, then all of your VMs and all of your data will go offline with it. That’s why you won’t ever get more than a 99.95% availability guarantee with an S2D configuration.

The single data center requirement of S2D also eliminates the possibility of deploying a highly available SQL Server FCI that spans Availability Zones in Azure, AWS, or even Google Cloud Platform’s “Zones,” which can also span multiple data centers.

Table 1: Advantages vs. Disadvantages of Windows Server Options

	Advantages	Disadvantages
Storage Spaces Direct (S2D)	<ul style="list-style-type: none"> Virtual storage area network (SAN) Can be accessed by any of the VMs in a WSFC VMs reside in the same data center 	<ul style="list-style-type: none"> Data goes offline with disruptive event Only a 99.95% availability guarantee Cannot deploy a highly available SQL Server FCI that spans Availability Zones
AlwaysOn Availability Group (AG)	<ul style="list-style-type: none"> Can support AG replicas among geographically distinct data centers Provides services that automatically synchronize SQL Server data amongst replicas Can reach up to 99.99% availability 	<ul style="list-style-type: none"> None of the databases are protected in case of failure Cannot replicate key system databases AG has not been tested beyond 100 SQL Server databases or 10 AGs

In contrast, an AlwaysOn Availability Group (AG) can support AG replicas among geographically distinct data centers, and when configured with replicas residing in different data centers, the SLA associated with an AG climbs to 99.99%. An AG configuration doesn’t rely on shared storage the same way as a SQL Server FCI. Instead, it provides services that automatically synchronize SQL Server data amongst the replicas in the AG. If the active SQL Server instance fails, the designated replica server takes over and begins hosting databases that have been replicated to it. But this approach also has its shortcomings.

AGs replicate user-defined databases, but not key system databases — including the Master and MSDB which hold

things like agent jobs, logins, and passwords. If there's a failure that takes the primary instance of SQL Server offline, none of these databases are protected. It's also worth noting that Microsoft has not tested AlwaysOn Availability Groups beyond 100 SQL Server databases or 10 AGs, which may pose other constraints if you want to protect a large number of databases.

Enter a New Generation of Shareable Cloud Storage

For all of the reasons cited above, the news of emerging high-performance, cloud-based file shares sounds intriguing. Will they enable organizations seeking cloud-based HA and DR solutions to move beyond the constraints that Windows Server has imposed?

In the long term, the answer may be "yes." AWS indicates that organizations can use Amazon FSx to configure a WSFC today. All of the nodes in the WSFC would have access to the file share, so if the primary node goes offline and the cluster fails over to a secondary node — even in another data center — that secondary node could continue working with the SQL Server data stored in the Amazon FSx file share. Similarly, Azure indicates that organizations can configure SQL Server in an FCI using an Azure premium file share. These descriptions suggest that the elegance of the failover cluster we have always known on-prem has finally made it into the cloud.

In the near term, the answer remains "no." Today's cloud-based shared file offerings have at least one major flaw: The underlying SLAs of the cloud file share services themselves guarantee only a 99.9% availability for read and write operations, which is far lower than the 99.99% SLA that constitutes the baseline for high availability.

And there are other issues, though these may simply reflect growing pains, as noted in the table below:

Table 2: Issues in Cloud-Based Shared File Offerings and Potential Outcomes

	Disadvantages	Potential Outcomes
Microsoft	Using Azure premium files in "input- or output-intensive workloads" is discouraged.	Their utility in your use case may be limited.
	File share can reside in only one data center (in all but two regions in Azure).	If the data center goes offline, your cluster goes offline with it.
AWS	There is no mechanism in place to ensure the active SQL Server instance and the file share always reside in the same AZ.	The absence of such mechanism could introduce latencies that would compromise the performance of your SQL Server system.
	Failover of the file share from one AWS region to another is documented to take about 30 seconds.	During this time, your SQL Server FCI will go offline and may require manual intervention to bring back online.

Advances in the delivery of cloud-based file shares are probable, and one day, all cloud service providers may be able to offer SLAs that guarantee the availability of the underlying SQL Server data at a 99.99% or higher. Then, cloud-based file shares will become a viable alternative to S2D, AGs, and other third-party options that we've relied on to date. When will that happen? We shall see. 🎲

The Rise of the Converged Cloud Databases

By [Gerald Venzl](#), Master Product Manager, Oracle Corp.

As more and more developers turn to the cloud to build cloud-native applications, it is becoming increasingly clear to them that cloud is not always removing the complexities but often shifting them from on-premises infrastructure to cloud service interconnectivity and interoperations. Developers are craving to be more productive and get business logic to market faster and don't want to spend time worrying about how their cloud components can or cannot interact with each other.

Data fragmentation is the biggest threat to long-sustaining cloud applications. While the middle tier can be removed or rewritten to accommodate changing business requirements or cloud infrastructure advances, data is harder to move or reorganize, especially once it has reached the volume of many years of production workloads.

Converged cloud databases offer an alternative to specialized, single-purpose cloud databases that can only deal with one type of data. Converged databases allow developers to store and analyze many different data formats and models within one single service while providing data-centric APIs that make accessing the data just as easy and transparent as single-purpose ones. Developers don't have to worry about how the data is persisted behind the scenes or whether it will ever be consumable to other parts of the application the way the other application wants to. There is also never a risk of developers choosing the wrong cloud database service or data format at the beginning of the project.

Converged cloud databases need no complex data migration nor downtime when developers change their minds or business requirements require another approach. In the case that the data format needs to change, no new skills have to be acquired by the developer as the database cloud service remains exactly the same.

[Oracle's Autonomous Database](#) is a converged cloud database that not only offers the benefit of storing many different data models, but it also provides coherent and best-of-breed operational aspects, making it truly converged. No downtime occurs when scaling the service. The database uses AI to analyze workload behavior to tune itself in real-time to the best possible outcome for a given workload and data model. Developers don't have to worry about complex and time-intensive database tuning efforts anymore and can rely on a database that makes storing data in the cloud just easy.

Features:

- Built-in support for Relational, JSON, Graph, Spatial, XML, Key/Value
- Continuous tuning using AI for workload analysis
- Fully transparent and elastic scalability
- Cross data model analytics and SQL

Try Autonomous Database for free: www.oracle.com/cloud/free/

Join the World's Largest Developer Community



Download the latest software, tools,
and developer templates



Get exclusive access to hands-on
trainings and workshops



Grow your network with the Groundbreaker
Ambassador and Oracle ACE Programs



Publish your technical articles—and
get paid to share your expertise

ORACLE GROUNDBREAKERS developer.oracle.com
Membership Is Free | Follow Us on Social:

 [@OracleDevs](https://twitter.com/OracleDevs)

 facebook.com/OracleDevs

ORACLE®

Data Safety in Cloud-Based Databases

By [Grant Fritchey](#), Microsoft Data Platform MVP and Product Advocate at Redgate

One of the most common arguments I hear around cloud adoption is lack of faith in the safety of online data stores. Often times, individuals tasked with protecting an organization's most vital asset — its unique set of knowledge — can be reactionary when it comes to risk management. Understandably so. It can feel like the best way to keep data safe is under your direct control, in your data center, on your servers. Yet, more and more organizations are trusting corporate data to be stored with cloud vendors.

Are these people just incredibly foolish, or is there more here than meets the eye?

Causes of Data Breaches

If you research the causes of data breaches, there's not much agreement on the most common cause; however, most sites will list the following:

- Weak and/or stolen credentials
- Application vulnerabilities from old or unpatched software
- Malware
- Malicious insiders
- Employee error leading to loss
- Theft/Loss of data carrying device
- Social engineering/phishing attacks

Except for malware attacks, the majority of attack vectors can be summed up as problems associated with internal systems and processes of the organizations that suffered a data loss. Outright hacking attacks, such as through a SQL Injection attack or other types of more sophisticated attacks, frequently don't make the top 10 lists.

If you read the news, you often hear about public GitHub repositories filled with user logins and passwords, publicly accessible drives with unsecured and unencrypted data, and laptops with entire data sets either lost or stolen. In short, the risk to your data is extremely high on the local systems that you manage because of one of the reasons listed above. Yet, people see the cloud and cloud-based databases as the higher risk. Let me explain a few reasons as to why this just not true.

Types of Cloud-Based Data Stores

Before we dive in, let me take a short moment to discuss the various types of data stores because not everything "in the cloud" is created equally. In this article, I'm going to focus on Platform-as-a-Service data offerings from different

cloud vendors. Whether we're talking about AWS with the RDS databases, Azure using Azure SQL Database and CosmosDB, Google's Cloud Spanner, or any of the others, this is a different type of data store.

If we were talking to you about hosting your servers through virtual machines in the cloud and then installing a data management system from SQL Server to MongoDB to Elasticsearch, the security and management of those VMs is the responsibility of your organization, just like with your current on-premises systems. The same goes with some of the file-based data systems. Just because you've moved the location of your file from your local machine to a cloud platform, the security is still on you.

No, I'm talking about the Platform as a Service (PaaS) offerings for data management because they fundamentally change the game. You are no longer managing your VM with these. Instead, you're putting your cloud provider in charge of the fundamentals of data management, and frankly, they're going to do a better job. The reasons for this are because many of the most common attack vectors — from a lack of updates and patching on your software to misconfigured systems, poor practices, and all of the other issues listed above that can cause a data breach — are removed.

Security by Default

First, most PaaS databases are designed with security in mind. Security is a fundamental aspect of all that they do. If you're getting set up with one of the cloud vendors, not only do you have to have a secure login and password (which, let's face it, is still vulnerable to attack), but you have to white-list IP addresses through the built-in firewall. So even if you have poor choices in logins, people won't be able to get direct access to your databases because there is a firewall blocking the access — unless they can also spoof your IP addresses.

There are even data systems out there that intentionally have no security in order to make development easier. At the time of writing, Elasticsearch, by default, does not include security in its development version. It's on you to put security in place before you release it to production. Yet, none of the PaaS offerings from the various vendors suffer from this problem.

One additional note: All cloud vendors I'm familiar with have both physical and plant security in mind. They build their data centers away from flood zones and wind zones. They have secondary power sources available but are also placing the data centers near a constant source of power, two of the best and cleanest being hydro-electric and nuclear. Security starts with the physical plant.

Continuous Patching and Updates

Different cloud vendors do this in different ways, but all of them have a strong upgrade and update set of requirements. Unlike your own systems, which can fall woefully behind and expose well-known and well-documented security issues that bad actors are happy to take advantage of, the cloud-based systems are patched regularly. Some of the cloud-based systems are even patched and updated continuously. You're warned that with the SLA, your databases will be upgraded on a schedule set by the vendor, not by you.

While this can sound scary, remember that you're used to patching or updating a few hundred servers whereas they're building systems that patch and update millions at a time. They've had a lot of practice and are quite good at it subsequently.

Backups and Maintenance

One of the saddest things I read about when I read about the constant drumbeat of data breaches is how often people don't have backups in place. Sometimes, they have them, but they're old and out of date. Or, they have them,

but they don't know how to restore them. Or, worst of all, they have them, but they were stored locally, using the same security that the malware just exploited, and now, those backups are lost.

Not so with the cloud-based PaaS offerings. All that I've studied and used have some form of backup in place. Further, depending on the type of data storage we're talking about, they have the ability to perform a point-in-time recovery. This is something that many businesses struggle to get right.

In addition to the backups, most of the PaaS database management systems are also performing constant and regular maintenance of the databases under their care. Once more, depending on the type of database and the cloud vendor, they may have additional protections under the covers so that they ensure your databases are available, even in the event of data corruption (it's rare, but it happens to all of us sooner or later).

Encryption, at Rest and in Motion

Most PaaS offerings make a point of ensuring that your data is encrypted within their systems. They encrypt it when it's stored within the databases, but they also encrypt the backups. Most major vendors ensure that they're encrypting data while it's in motion within their data centers. In the event of a breach, all this added protection helps to ensure that the breach is going to be contained at the lowest possible level, rather than expose not only multiple databases but multiple organizations to a single breach.

High Availability

One last thing to point out is that most cloud vendors offer varying degrees of high availability within their systems. They typically offer some degree of built-in availability and disaster recovery at zero cost to you. However, you can pay for extra protection. So, with some additional overhead, you can ensure that not only is your data available in whatever data center you store it, but that you have automated processes that ensure it gets duplicated to secondary locations. Most of the major vendors can do this for locations all around the world.

This is a level of disaster protection that most businesses haven't even attempted, let alone successfully implemented.

Conclusion

The fact of the matter is that you may have an excellent data center with secondary power systems — security, well-maintained patching, tested security, tested backups, and a tried-and-true high availability setup. On the other hand, you may have need of these services. The most important things to think about are the common attack vectors. How many of these are increased by moving your data to cloud? Very few, if any. How many of these are decreased by moving your data to the cloud? Most of them.

So, if you're hesitant to move to the cloud because you worry data will be less safe, I strongly encourage you to look at how cloud vendors are handling security and how data breaches occur. Once you have this knowledge under your belt, you're going to worry much less about cloud-based data systems.

References:

- [Most Common Causes of Data Breaches and How You Can Spot Them](#)
- [Top 6 Causes of Data Breaches and How to Plan Against Them](#)
- [Most Common Causes of Data Breaches](#)
- [7 Major Causes of a Data Breach](#)

The Future of Your Database: A Look Back From 2025

By Jonathan Ellis, DataStax

Imagine the world of software infrastructure in 2025. Let's take a look 'back from the future' at how machine learning, native databases in the cloud, and graph capabilities combined to change the database landscape more in five years than in the previous twenty.

The most profound impact was that databases became self-driving. Machine learning and the infinite patience of AI can dial in hundreds of configuration options to achieve the best possible performance customized for each workload in a way that we as humans could never do. Initial examples such as index optimizations in the relational world and the [research of CMU professor Andy Pavlo](#) led the way to self-managing databases becoming common and reliable.

The cloud now gives any developer the ability to take an idea and 'just go' using simpler APIs like REST and GraphQL that let most of them think of the database as a black box services layer. In fact, in 2025 few developers use the term 'database' at all! These simpler, faster, infinitely scalable native cloud databases delivered with zero administration (like [those from DataStax](#)) have now eclipsed the category of managed services of legacy databases designed to do everything.

Finally, in 2025, graph technology is a must-have database feature, giving developers the ability to get more intelligence from data without writing lots of code for cross-partition queries, tree retrievals, pattern matching, and more. By now, most enterprises also realize that graph is an important feature of a scale-out database — not a database in itself — and that developers aren't well served by doing different pieces of the app in different systems connected by ad-hoc, error-prone application logic. Vendors like [DataStax](#) made this easier for users and enterprises by bringing graph capabilities into the next generation of scale-out databases like Apache Cassandra.

No matter what the future holds, it's never been more fun and exciting to work with data and databases! We look forward to following your innovations in the years to come.

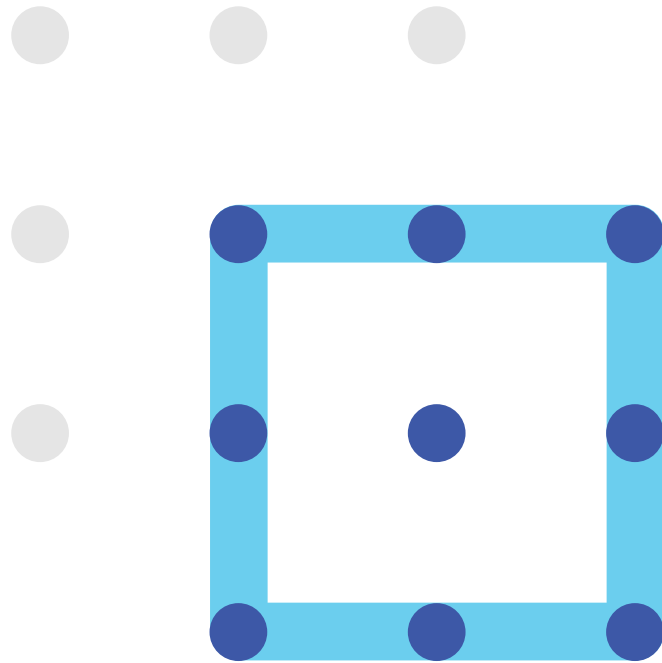
TREND PREDICTIONS

- ▶ Machine learning enables 'self-driving' databases.
- ▶ Developers adopt native cloud databases with simpler APIs.
- ▶ Graph is a powerful feature but not a database in itself.

Your relational database isn't cutting it.

Discover the most powerful way to scale with NoSQL and Apache Cassandra™

[Read More](#)



"We (ICE Data Services) compile quotes from almost every market in the world in near real-time and create synthetic products leveraging DataStax Enterprise (DSE). Our ability to deliver key, reliable, real-time data products enables our customers to continuously calculate risk, accurately price assets, and power their mission critical financial platforms."

Steve Hirsch, Chief Data Officer, ICE and NYSE



A Customer Case Study: Thorn

By [Floyd Smith](#), MemSQL

Thorn is a technology-driven not for profit, working to eliminate child sexual abuse from the Internet using machine learning and AI. Their child sex trafficking investigations tool, Spotlight, gathers information from escort sites to provide law enforcement with a tool to help find trafficked children, fast.

Spotlight is all about the fast ingest and processing of data, so the choice of database was crucial. For Spotlight, Thorn needed to meet a wide range of challenging requirements:

- From a technical point of view, the database needs to be high performance and elastically scalable, with built-in functionality to support machine learning and AI. The database needs to drive real-time comparisons between photos posted on the Internet and photos of potentially trafficked children.
- From a management point of view, the database needs to be low maintenance and easy to operate so that Thorn can focus on improving their tools instead of maintaining infrastructure. Thorn needs to support a fast-growing law enforcement user base, easily and seamlessly, with flat technical effort and management overhead.

As of now, Spotlight has helped identify over 10,000 trafficked children. Law enforcement investigation time has been cut by nearly two-thirds.

“MemSQL easily scales to support our machine learning and AI needs. It is a true case of technology being applied in a way that will make a real difference in people’s lives.”

Julie Cordua, CEO of Thorn

Product

MemSQL Helios

Cloud-native, operational database built for speed and scale

Category

Cloud Database

Release Schedule

Continuous updates

Free Option

Yes

Strengths

- Fast
- Elastically scalable
- Native ANSI SQL support
- Fully managed
- Superior TCO
- Multi-cloud
- Hybrid cloud

Notable Users

- Comcast
- Dell EMC
- Pandora
- Sony
- Uber

memsql.com

memsql.com/blog

[@MemSQL](https://twitter.com/MemSQL)



All-in-one Database. Zero upkeep.

Hello Helios.

Experience best-in-class performance, capability, and reliability from MemSQL, available instantly and on-demand in the cloud.



Distributed SQL database built for transactional and analytical workloads



Automated provisioning, configuration, and elastic scaling



Available across all major cloud providers

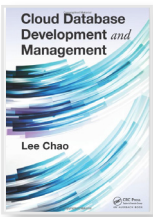
[Try It Free](#)



For more information, and to get started visit memsql.com/helios

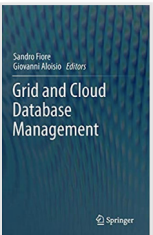
Diving Deeper Into Cloud Databases

Books



Cloud Database Development and Management by Lee Chao

This book explains how developers can take advantage of cloud environments to develop fully functioning database systems without additional investments in IT infrastructure.



Grid and Cloud Database Management

by Sandro Fiore and Giovanni Aloisio
This book, a technical reference for grid and cloud database management, provides detailed descriptions of research prototypes dealing with spatiotemporal or genomic data, which are useful for application engineers in these fields.



Hands-On Database by Steve Conger
Before you can successfully manage a cloud database, you have to understand the basics. This text uses a scenario-based approach to show readers how to build a database, walking you through each step of the process.

Refcards

Database Monitoring Thanks to DevOps, databases are managed in a very different way, and both DBAs and developers must monitor performance, security, backups, file size, and job outcomes.

Hybrid Cloud Vs. Multi-Cloud This Refcard aims to define “hybrid cloud” versus “multi-cloud” and looks at some of the management challenges in a multi-cloud world and how best to address them.

Cloud Capacity Management This Refcard dives into what it means to extend capacity management to the cloud, how it differs from traditional on-premises capacity management, and how to apply it in specific use cases.

Zones

Cloud The Cloud Zone covers the host of providers and utilities that make cloud computing possible and push the limits (and savings) with which we can deploy, store, and host applications in a flexible, elastic manner. The Cloud Zone focuses on PaaS, infrastructures, containerization, security, scalability, and hosting servers.

Database The Database Zone is DZone's portal for following the news and trends of the database ecosystems, which include relational (SQL) and nonrelational (NoSQL) solutions such as MySQL, PostgreSQL, SQL Server, NuoDB, Neo4j, MongoDB, CouchDB, Cassandra, and many others.

Performance Scalability and optimization are constant concerns for the Developer and Operations manager. The Performance Zone focuses on all things performance, covering everything from database optimization to garbage collection to tweaks to keep your code as efficient as possible.

Podcasts



Voice of the DBA This podcast was adopted by the “Voice of the DBA” (Steve Jones) for his writing and work with SQL Server.



The Cloudcast This podcasts series helps guide anyone new to Cloud Computing to understand both the technical and business aspects.



DBAle Each episode discusses topics and news in the world of SQL Server, while enjoying and chatting about beer. Cheers!



INTRODUCING THE

Database Zone

Managing the growing amount of data effectively is an ongoing concern. From handling event and streaming data to finding the best use cases for NoSQL databases, there's plenty to unpack in the database community.

Keep a pulse on the industry with topics such as:

- Best practices in optimization
- Database performance and monitoring
- Handling event and streaming data
- Advancements in database technology

Visit the Zone



TUTORIALS



CASE STUDIES



BEST PRACTICES



CODE SNIPPETS

Cloud Native Trend Report

A faster, more scalable approach to building applications

BROUGHT TO YOU IN PARTNERSHIP WITH



Table of Contents

- 3** Highlights and Introduction
BY FRANK EAVES
- 4** Key Research Findings
BY MATT LEGER
- 11** Executive Insights: Everything You Need to Know About Cloud-Native
BY LINDSAY SMITH
- 14** How Cloud-Native Development Will Create Future Business Value
BY MALVI GOYAL
- 19** Deploying Microservices and the Future of Cloud-Native
BY VIRAJ PHANSE
- 23** Diving Deeper Into Cloud-Native

To sponsor a Trend Report:

Call: (919) 678-0300

Email: sales@devada.com

Highlights and Introduction

By **Frank Eaves**, Senior Software Engineer at DZone

Here at DZone, it's always exciting to see new technologies emerge and to observe trends that spark evolution across industries. We're further inspired by the diverse groups of people and companies who join forces to innovate, create tools, and contribute to the space. Because of that, such efforts cascade down to the individual developer and have a lasting impact.

Today, these initiatives are driving the adoption of a standardized approach for developing and deploying software applications to the cloud — becoming 'cloud native.'

In the past, a large piece of software contained many modules, usually tightly coupled. New principles like CI/CD emerged and put an emphasis on continuous testing and security. As a result, testing became more efficient, the cost of software maintenance declined, and the overall complexity of the SDLC decreased.

With the growing development and availability of cloud technology also came different ways of looking at and solving software problems. These techniques and supporting tools shaped what's broadly known today as cloud-native computing.

Microservices — an approach that's taken over the software industry and its utilization of cloud resources — involves breaking applications into smaller separate pieces of software, all running independently to solve a larger problem.

Both large and small software companies are moving in this direction, migrating all architecture to the cloud and adopting microservices to manage applications. As part of this process, large software companies are breaking down their monolithic pieces of software into microservices. This is done so that they can leverage cloud resources, offloading a significant portion of software management to cloud providers.

What is perhaps more significant is that they remain competitive with the smaller companies and startups who are already using these services for more productivity among their developers and current software products and services in the market.

This report highlights key findings from our original research on these trends and addresses the impact of cloud-native technology across industries. Further are explorations of becoming cloud-native and the business advantages upon adoption.

Malvi Goyal kicks us off with a detailed look at what it takes to be "cloud-native," exploring the business value generated by cloud-native applications, including things like auto-scalability and infrastructure investments.

In Viraj Phanse's article, "Deploying Microservices and the Future of Cloud Native," he dives deeper into the role of microservices in cloud-native technology and how this will change development and deployment for the better.

According to Goyal, "the million-dollar question for today's enterprises is less about why they need cloud-native and more about how they become cloud-native!" 

Key Research Findings

By **Matt Leger**, Principal Research Analyst at DZone

Since the early 2010s, IT organizations around the world have been transforming the way they build applications using an approach known as “cloud native.” According to the [Cloud Native Computing Foundation](#),

“Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.”

Key Features of the cloud native approach:

- Applications are broken down into smaller microservices.
- Microservices are packaged in containers.
- Containers are dynamically managed in the cloud with orchestration tools.

To better understand the current state of cloud native almost a decade after its birth, we surveyed 396 software developers from DZone’s global audience.

KEY TAKEAWAYS

- ▶ 90% of developers are using, or plan to use, microservices — the leading tools are Spring Boot, Java EE, Amazon Lambda, and Prometheus.
- ▶ 90% of developers who use microservices use, or plan to use, containers to manage them — the leading tools are Kubernetes (53%) and AWS ECS (26%).
- ▶ Common challenges with building microservices include changing organizational culture, communicating between microservices, and app/service security.
- ▶ Few organizations (12%) have reached complete cloud native capabilities in all of their apps, and movement toward serverless technology is even further away from reality.
- ▶ Developers believe that hybrid and multi-cloud strategies, followed by “everything is code” and the emergence of serverless architecture, are the most important emerging cloud-native trends.

The Takeover of Microservices

Microservices appeared on the software architecture scene at a European conference in 2011. In less than a decade, adoption of microservices has snowballed so dramatically that three out of four global organizations that build software applications use the approach. And a majority of the remaining organizations are considering microservices. Only 9% are not within the market for this method of software development, of which more than half said that they cannot find an applicable use case that fits their current business needs.

So what led to the rapid adoption of microservices? Microservices help developers solve some of the basic software issues that they have faced for years — speed to market, scalability, quality, flexibility, and cost savings. Microservices also

make experimenting with different architecture configurations to achieve incremental gains in these areas easier. As the demand for technology improvement and innovation continues to grow, organizations feel increasing pressure to move at pace with the market.

Why are you using microservices?	Percent
Enable faster deployments to just one part of an app.	75%
To make apps easily accessible.	64%
Improve quality by having teams focus on just one piece of an app.	51%
Flexibility to choose different tools/languages.	44%
Improve quality by narrowing down source of failure to a particular piece of an app.	41%
To experiment with the architecture	15%

Now that we know why developers use microservices, how do they build them? For a majority of developers that we surveyed on DZone, the clear winner was Spring Boot — other tools included Java EE, Amazon Lambda, and Prometheus.

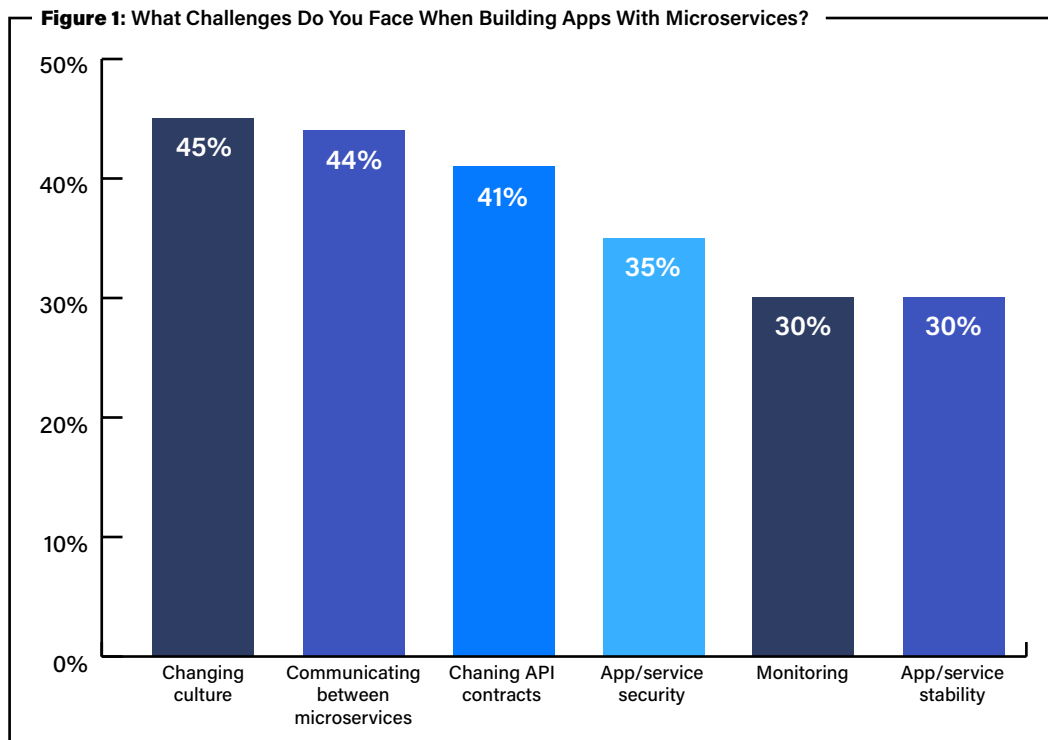
			
61%	32%	31%	22%

Perhaps most revealing about the impact of microservices, 60% of developers believe that it makes their jobs easier, which is twice the number of those that felt their jobs are not easier. The remaining group still felt unsure. So what challenges do developers experience with microservices?

Challenges With Microservices Adoption

While almost everyone uses microservices today, it does not mean that the technique comes without complication. Across the board, developers face similar challenges, though one is unrelated to technology: changing the culture to be open to microservices — a major barrier for nearly half of respondents. Communicating between microservices and changing API contracts was close behind. Security, of course, was a noteworthy challenge for more than a third of respondents (*Figure 1, see next page*).

Inherent to innovation is the need to depart from traditional ways of working. Particularly for larger, more established companies, switching from legacy architectures to microservices can take significant time and effort. In fact, when asked what challenges they face while switching from legacy architecture to microservices, one-third of developers said that the time investment alone is a major challenge. However, the biggest challenge developers face by far (64%) was determining how to break up monolithic components of existing apps. Challenges following that included overcoming tight coupling (46%), bugs (39%), and incorporating other technologies like containers (35%).



Security Concerns With Microservices

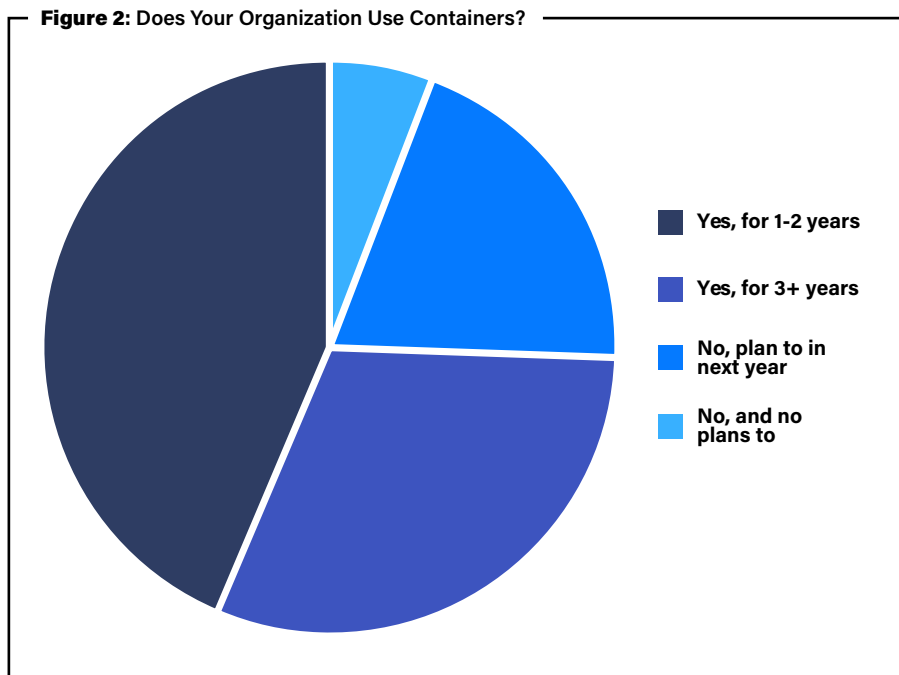
As with any change in software development and deployment, security is always an important consideration. The leading security concerns related to cloud native for developers are sensitive data exposure (56%) and API security risks (55%). Secondary concerns revolve around how security fits into day-to-day work. This result is not surprising given the constant pressure and demand developers now face to create, test, and deploy new apps on a continuous integration and continuous delivery model (CI/CD) — a critical component of being cloud native.

Which security challenges concern you most about cloud native?	Percent
Sensitive data exposure	56%
API security risks	55%
Balance between security and efficient deployment	44%
Fitting security into the DevOps process	39%
Service disruption	36%
Limited real-time visibility into hacks	27%
Broken authentication	27%

Developers use a combination of security methods to protect against these threats to microservices: JSON Web Tokens (45%), Defense in Depth (33%), OAuth 2 (29%), and User Authentication (18%) are the leading methods.

Taming the Microservices Beast With Containers

Microservices have been revolutionary in changing the way apps are developed and deployed. However, breaking down larger applications into smaller microservices adds additional layers of complexity to the process. Managing increasing numbers of microservices can become difficult if developers are not careful. This is the point at which containers come to the rescue. With the explosion of microservices have come an equally widespread use of containers to manage them. Virtually every organization uses containers — or is working to implement them — within their workflow to help manage their microservices architecture.



Development and Production of Containers

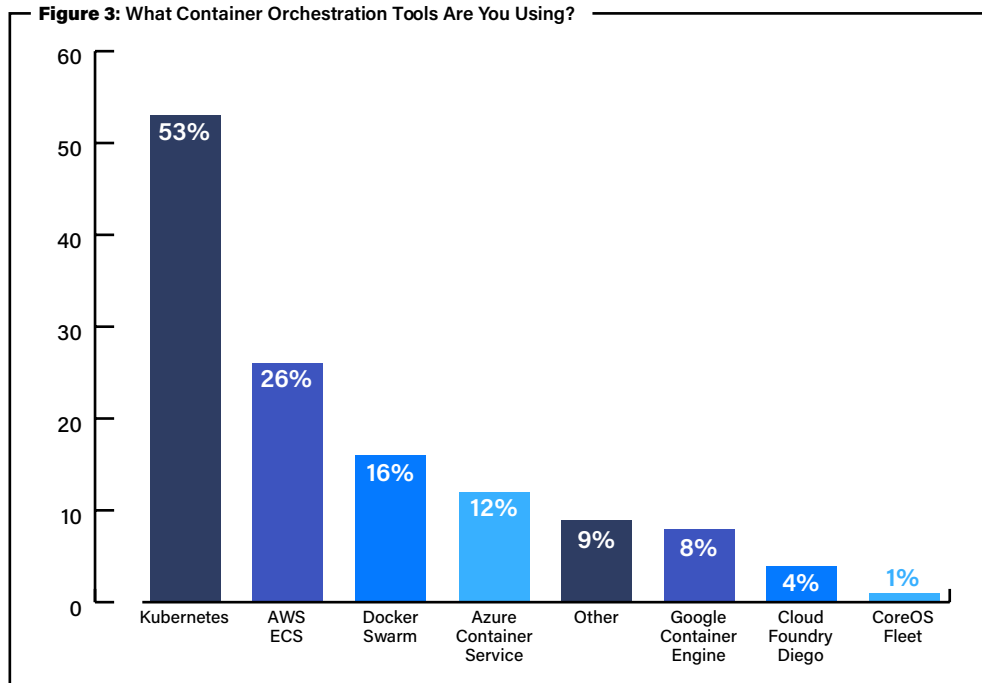
When organizations begin using containers, there are many moving parts right from the start. Of the respondents who said that their organization has used containers for the past one to two years, two-thirds use them in both development and production. By year three, the number jumps to 84%.

At the same time, the number of developers who said that their organization uses containers only in development differs significantly between years 1 and 2 (22%) and years 3 or more (8%). This demonstrates that while organizations jump right into container development and production, it can take several years to fully integrate containers into the DevOps process.

How are you using containers?	Does your organization use containers?		
	Yes, for 1-2 years	Yes, for 3+ years	% Change
In Development + Production	67%	84%	+17%
In Development	22%	8%	-14%
In Production	11%	8%	-3%

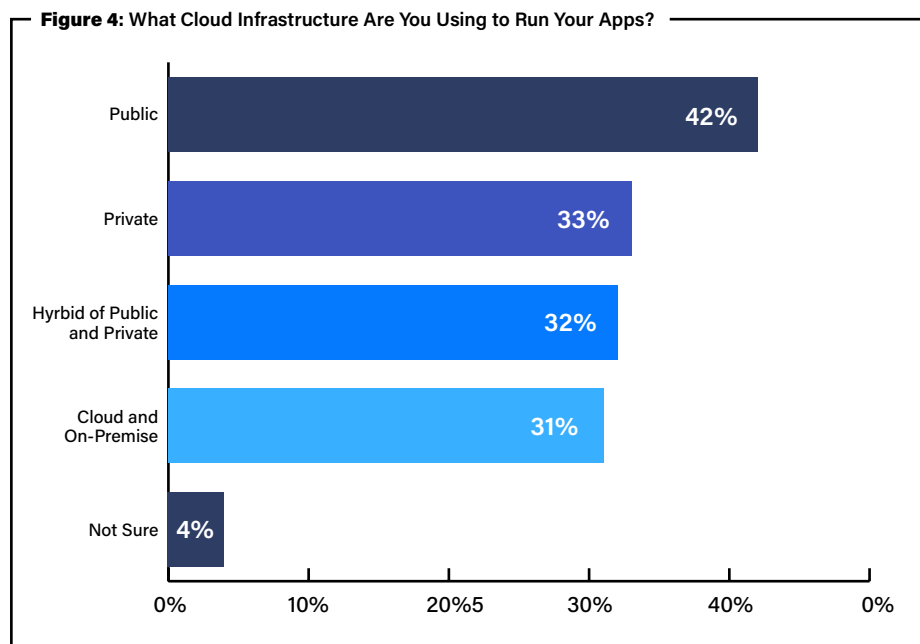
Kubernetes Dominates Container Management

Like microservices, using containers adds another layer of complexity to the software development lifecycle. While containers help to manage microservices, managing their deployment manually takes away precious time from developers. That is why 60% of organizations use orchestration tools to automate certain tasks and another 30% plan to in the next year. Container orchestration allows development teams to focus on building, deploying, and maintaining apps continuously and scaling apps more quickly. For developers, Kubernetes is the clear orchestration tool of choice (53%) followed by AWS ECS (26%).



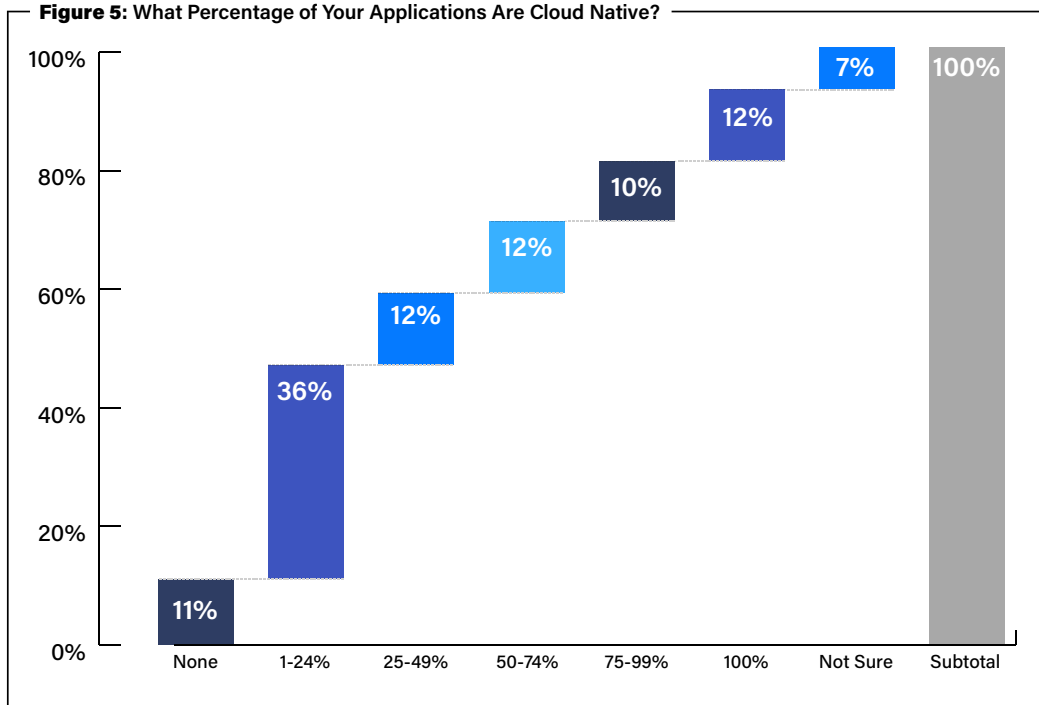
Cloud Infrastructure Models

In their efforts to become cloud native, developers said that their organizations use a mix of cloud infrastructure models. There is a slight edge in favor of public cloud, but there is not a clear dominant approach to date.

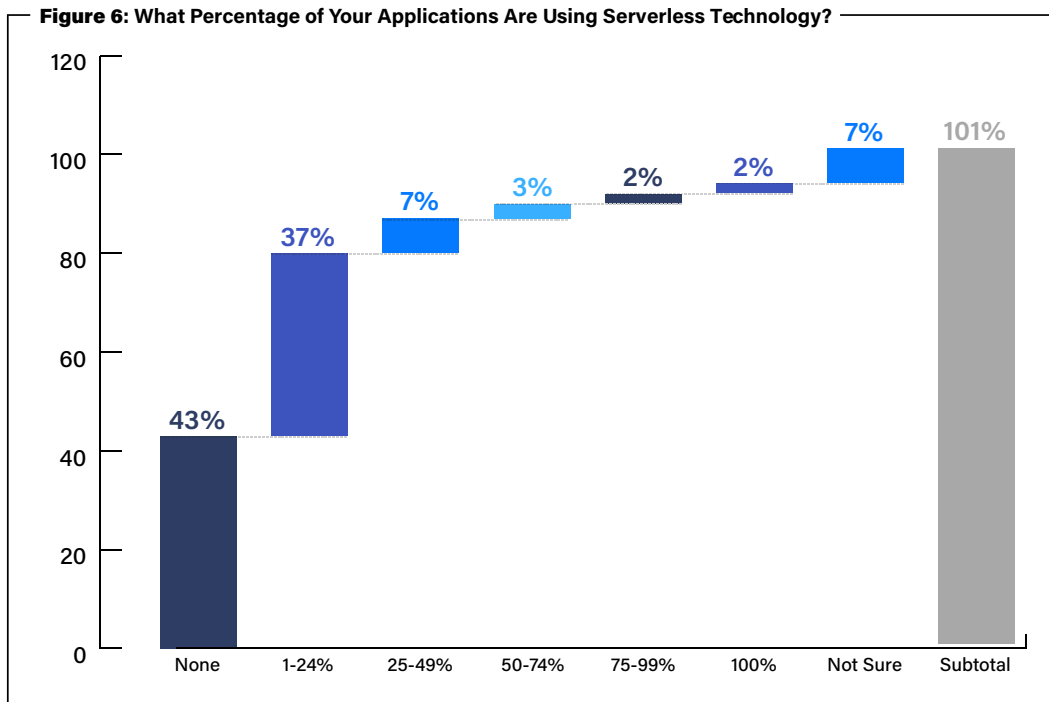


What's Next in Cloud Native?

At the end of the day, many organizations are working toward building cloud native capabilities, but very few are actually there. In fact, nearly half of all organizations represented in the survey have reached cloud native capabilities with less than one-fourth of their existing apps. Only 12% of organizations can say 100% of their apps are cloud native. For many organizations, there is still quite a ways to go toward becoming cloud native.



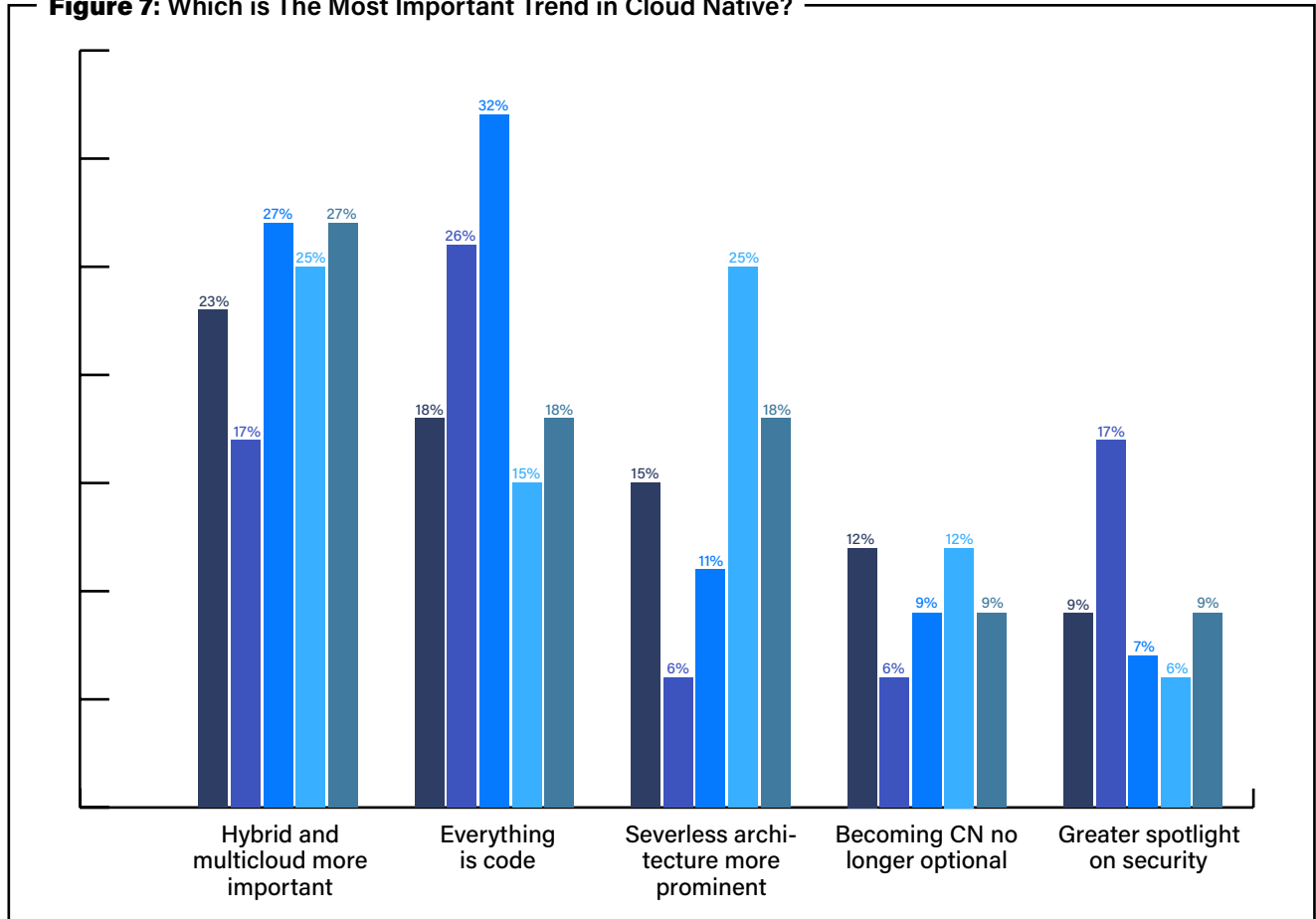
For the next big development in cloud native — using serverless technology to run apps — organizations are even further away from adoption. In fact, 43% of respondents said that none of their apps are serverless, and 37% said that less than one-fourth of their apps are. Serverless technology is still very much in its early stages of development.



There are many notable trends occurring around cloud native — which ones do developers think will be the most important? Overall, “hybrid and multi-cloud strategies will become more important” was the top prediction for developers (23%). Although “everything is code” ranked second (18%), the number of developers who agreed that this was a leading trend varied by location.

Additionally, the idea that cloud native will put a spotlight on security was far more prevalent for developers in North America (17%) compared to those anywhere else in the world. Lastly, serverless architecture was a significant trend among 25% of developers in Asia-Pac, compared to just 6% and 10% for developers in North America and EMEA, respectively. 🌐

Figure 7: Which is The Most Important Trend in Cloud Native?



Executive Insights

Everything You Need to Know About Cloud-Native

By [Lindsay Smith](#), Publications Content Manager at DZone

Questions around how to efficiently manage microservices, accelerate deployments, and make applications scalable are all answered through cloud-native technology.

Cloud-native is all about taking advantage of the cloud in every way possible. This results in faster, more efficient ways to run, develop, and deploy applications — every aspect of your application infrastructure has been adopted and implemented with the cloud in mind.

So now, instead of dealing with a monolithic architecture and massive amounts of code at a time, applications utilize microservices to create a distributed architecture that's much easier to manage.

The speed and efficiency at which cloud-native applications run allow businesses to get products and services to market faster. And being able to scale and deploy applications faster than your competitors will serve as a huge win for your organization.

To learn more about why this technology matters — to both developers and executives — we sat down with experts in the cloud-native space to discuss trends, challenges, and predictions for the upcoming year.

Moving to a Microservices Architecture and Addressing Use Case

Based on our research findings, we found that 90% of respondents currently or plan to use microservices in the next year. This means a major increase in cloud-native applications and questions around managing microservices.

The migration from a traditional monolithic architecture to microservices helped to evolve application architecture. A monolithic architecture follows the traditional model for app development where the business logic, UI, and database are all under the same umbrella, so to speak, while the introduction of microservices allowed for individual services to be managed within one application, including a separate database, business logic, etc. for individual services — all under the same UI.

When adopting microservices, the leading challenge is determining where to break up monolithic components, which is not surprising based on the challenge many organizations face when finding a valid use case. This is the result of two things: trouble understanding the market and a basic knowledge gap between microservices and what they can do for your business.

This post [“Where Microservices Are Actually Useful: Two Types of Use-Case”](#) provides thoughts on the most common use cases for adopting microservices. “The most obvious use cases are those of a CPU- or RAM-intensive part of the application. That normally goes into a separate deployment, offering an interface to the rest of the application.”

Services that consume a lot of RAM are impractical to run every time a developer starts the application, so moving to a distributed architecture can resolve some of these concerns.

Another use case for adopting microservices is the idea of multiple teams being able to work on the same product to increase flexibility and efficiency when building applications.

But microservices aren't limited to these two use cases. There are increasingly more use cases as these services become a larger piece to the cloud-native puzzle.

Expediting Day-Two Operations With the Operator Framework

Another important topic to consider when adopting cloud-native technologies are operators and operator patterns. According to [Kubernetes.io](https://kubernetes.io):

"Operators are software extensions to Kubernetes that make use of custom resources to manage applications and their components. The Operator pattern aims to capture the key aim of a human operator who is managing a service or set of services."

We talked with Director of Community Development at Red Hat, Diane Mueller, who further stressed the importance of executives learning about operators sooner rather than later.

"If there's anything to get across to executives right now, the new word that they're going to hear, if they haven't heard it already, is operators in 2020," said Mueller. "Operators take the next step beyond just installing and configuring a service or an application to managing the day-two stuff, like operations and lifecycle management!"

This includes applying a patch, reconfiguring parts of the application, automatically installing updates, completing back-ups, and more. All of these tasks were previous time-sinks for Ops teams, but now, they can all be automated through operators and operator patterns, which reduce downtime and time spent testing.

Automation and speed are major components of successful cloud-native adoption, so being able to build upon these concepts with the incorporation of operators will give your business a one-up.

Muller explains: "'Automate everything' is the key mantra of DevOps movement for some time, and cloud-native development is now a way of life for most organizations. Most organizations are building and operating cloud-native applications and services and leveraging automation practices that integrates the concepts of DevOps, continuous delivery, microservices, and containers."

Optimizing Workloads With Container Registries

Containers play a major role in moving to a cloud-native architecture. [The Linux Foundation](https://www.linuxfoundation.org) defines cloud-native and the role of containers as:

"[Cloud-native applications] use an open-source software stack to deploy applications as microservices, packaging each part into its own container and dynamically orchestrating those containers to optimize resource utilization."

As our research shows, 76% of organizations currently use containers, and 20% are not currently using containers but plan to in the next year. So, essentially, nearly every organization is working to integrate containers into their workflow and microservices management solutions.

[Container registries](#) are made up of container repositories, access control rules, indexes, and API paths. Offering continuous availability within Kubernetes, container registries allow developers to have several versions of an individual container created at various times.

According to Mueller, not only are container registries important, but she predicts more innovation around registries, scanning, and security in cloud-native technology: "Everybody needs a container registry," explained Mueller. "In terms of a container registry, you're going to see more people focusing on getting highly scalable container registries and worrying more about finding trusted hubs." Which, we will touch more on security concerns in one of the following sections.

Another advantage in employing container registries, according to Mueller, is the opportunity for optimized workloads. Mueller expressed that many companies are working to ensure that Kubernetes “plays well” with GPUs and on bare metal, which she emphasizes will be key initiatives in 2020. This will move beyond just improving workloads but also getting products to market faster and creating more stable, reliant applications.

Challenges Adopting Cloud-Native

Security

Our findings showed that 55.1% of respondents were most concerned with API security and data leaks, with concerns over finding the balance between security and DevOps processes.

With the perpetual demand to develop, test, and deploy new apps on a continuous basis, and consequences when code fails and/or is breached, these challenges are extremely real and hold major consequences when the proper procedures and precautions are not taken. Most organizations have adopted basic security hygiene ever since the Equifax hack back in 2017. Now, organizations are getting more sophisticated and strategic about security. The basic stuff, while equally as important, will simply not cut it anymore.

Knowledge and Training Gap

In addition to tackling security with a [DevSecOps](#) approach, our findings suggested there to be a lack of time for training and refactoring as well as a general lack of understanding around cloud-native development throughout organizations. Lack of time and training were cited as key reasons to not build out a microservices and/or cloud-native architecture.


According to Mueller, one of the best ways to combat knowledge and training gaps is to become active in the cloud-native community:

“It’s not the tech anymore. It’s the collaboration. That’s going to make us all successful and making sure that we have healthy collaborations, healthy connections, and we have active and vibrant communities. This is really the essential component for the continued success of the cloud-native ecosystem.”

Bottom Line

In [an interview with Pivotal about cloud-native technologies](#) and adoption, James McGlennon, Executive VP and CIO, [Liberty Mutual Insurance Group](#), said:

“One of the things we’ve learned is that if you can’t get it to market more quickly, there is no doubt that the market will have changed and no matter how well you’ve engineered it or built it or deployed it or trained your folks, it’s not going to be quite right because it’s just a little too late.”

This applies to cloud-native and microservices adoption now more than ever. Cloud-native is not going away. This approach to software development is unparalleled in terms of speed, delivery, and scalability. 

How Cloud-Native Development Will Create Future Business Value

By [Malvi Goyal](#), Product Marketing Manager at [DXchange.io](#)

Building applications in the cloud-native world is a whole different ball game. Cloud-native applications provide superior functionality that makes systems more cohesive and adapt quickly to changes in a fast-paced environment.

Per Gartner, many organizations are now focused on cloud-first strategies as they turn their attention to advancing the use of cloud services across the business. There are two approaches: migrating your existing applications to the cloud, which can be a major hassle, or making indigenous cloud-native applications. The difference between the two approaches will set world-class organizations apart from the rest.

Unlocking the true powers of the cloud — resiliency, agility, ease of implementation, and scalability — will require enterprises to adopt a cloud-native mindset. This article is aimed to look at how enterprises can implement a cloud-native deployment model and the benefits of adoption.

What Does it Take to Be Truly Cloud-Native?

Cloud-native development refers to how applications are built and deployed, not whether they sit on a public, private, or hybrid cloud. In technical terms, a cloud-native application is one born in the cloud and built as microservices packaged into containers.

IDC reports that by 2022, [90% of new apps will feature microservices architectures](#) that improve the ability to design, debug, update, and leverage third-party code; and 35% of all production apps will be cloud-native. The whole premise of using cloud-native applications is to replace the investments and manpower required to maintain an enterprise's data centers, which are not scalable and cannot adapt to the rapidly changing IT requirements of the enterprise.

So what exactly is a microservice?

Microservices architecture is a modern approach to developing software by creating multiple smaller software entities, better known as microservices. Each of these services is independent of each other and designed to perform a specific task. This independence ensures each service can be revoked or iterated without disrupting the complete application or the experience of end users. Each service is deployed in light-weight containers, which are faster to deploy and can be scaled further with automation and orchestration processes.

A cloud-native microservices architecture is scalable and faster when compared to its traditional web models and IT frameworks counterparts. Enterprises often use terms “cloud-native” and “cloud-enabled” interchangeably, but there is a huge difference between the two and their individual functionalities. A cloud-enabled application is made in a static environment on in-house servers and is merely a traditional enterprise software enabled for the cloud.

Cloud-native applications are critical for enabling a connected customer experience today and in the future. Merely having cloud-enabled applications will not work due to their scalability limitations.

Let's see how enterprises can prepare for the future with cloud-native application development.

A Strategic Approach to Adopting Cloud-Native Application Development

Although moving to cloud-native application development seems lucrative and advantageous, meticulous planning is required to carve out a roadmap for adoption. Below are important considerations for enterprises that aim to become cloud-native.

Change Management Plan

Moving IT assets to the cloud is a tedious and time-consuming process. For instance, [after being bought by Facebook](#), it took the Instagram team over a year to establish governance for the transfer of all its services from AWS to Facebook data centers. Hence, a comprehensive change management plan is the first and foremost step to encapsulate the overhaul of traditional monolithic applications and whether these need to be modernized or built from scratch.

Thorough risk assessment and mitigation plans, security and compliance issues, and smooth movement of on-prem applications to a cloud ecosystem without causing bottlenecks/downtime all need to be addressed before starting development.

Movement Towards Microservices

Microservices architectures are the backbone of cloud-native development. Applications need to be broken down into granular, single microservices, which are loosely coupled and can be developed continuously without causing disruption or downtime to the overall system.

Availability of Skilled Expertise

Cloud-native development is much more than changing programmers to DevOps. Cloud-native applications are designed to be hosted as multi-tenant instances and demand continuous delivery, automatic deployment, and support for frequent changes. These are dynamically orchestrated in containers that are actively scheduled to optimize resource utilization.

A highly skilled and trained DevOps team drives cloud-native development by building, testing, and releasing software more rapidly and reliably using automated processes for software delivery.

Synergies Between IT and Business Groups

More than a technical change, cloud-native development is a cultural change that engulfs an entire enterprise. Successful use cases for shifting to cloud-native infrastructure involve agile methodologies and collaboration between IT teams and business groups, through which developers use continuous feedback from business groups and quickly iterate updates. Clear communication involves sharing best practices and consistent feedback loops between IT teams and users to ensure business continuity and to develop applications that meet user expectations.

In a nutshell, cloud-native developments create discrete elements/services that can be rapidly deployed. Future-ready enterprises must evolve and inculcate the right culture to cope with the pace of innovation that comes with cloud-native development.

The Business Value Generated by Cloud-Native Applications


Cloud-native application development prepares enterprises to embrace the onset of digital technologies. It provides organizations with an edge to perform in a competitive business environment. With a scalable architecture, they can focus more on differentiating their underlying business proposition rather than investing in infrastructure.

A cloud-native development ecosystem adds tremendous business value through:

- 1. Auto-Scalability** – This is probably one of the most strategic benefits. Cloud-native applications are highly scalable; real-time changes can be made to a microservice without disrupting the entire application as per the deemed requirement. A real-life use-case has been shown by [IBM cloud solutions](#), which helped American Airlines move out of their existing legacy architecture and enhance the customer experience.
- 2. Infrastructure Investments** – Cloud-native application development is relatively cost-effective, as the costs are based on the licenses and the storage required on the cloud for an application. It requires no software and/or hardware upgrades or installations, curtailing down on infrastructure investments.
- 3. Maintenance as a Service** – Cloud-native applications are comprised microservices, which can be maintained easily and improved incrementally. Further, changes can be applied to microservices without interruptions or downtime. They can be updated, managed, and deployed individually.
- 4. Ease of Implementation** – The implementation of cloud-native applications is fast and efficient as no hardware or software configurations are required. The whole process of cloud-native development more accurately matches the speed and innovation that is demanded by today's changing business requirements.

The future belongs to cloud-native as it impacts the complete SDLC from all aspects — design, implementation, deployment, and operation. Oracle states that [80% of all enterprise workloads](#) will move to the cloud by 2025. As the world is envisaging digital disruption in day-to-day life, they will be opting for tailor-made cloud applications to deliver business needs and gain an edge over competitors.

Cloud-based architecture is enabling enterprises to adapt and respond quickly to business changes through continuous delivery. The power of cloud architecture lies in moving ideas to implementation in production as quickly as possible, making the most out of available business opportunities.

So the million-dollar question for today's enterprises is less about why they need cloud-native and more about how they become cloud-native! 

Ship Cloud Applications Faster

Embed security, compliance and performance into your DevOps workflow



Detect Vulnerabilities



Block Threats at Runtime



Validate Compliance



Maximize Performance and Availability



LEARN MORE AT
sysdig.com

2020 Cloud-Native Predictions

Security is the #1 Priority

By Pawan Shankar, Senior Manager, Product Marketing at Sysdig

1. A high-profile container exploit will dramatically increase interest in container security.

2020 will be the year that a vulnerability in runC, similar to CVE-2019-5736, will be exploited in a big way before it gets patched. This event will expose the shortcomings of using legacy tools to secure cloud environments. Security teams who are not as intimate with securing the cloud will begin to realize that “it’s running in a container” does not mean it is safe from threats and that legacy security tools do not fully protect them in a dynamic cloud environment.

The risk shouldn’t result in missed opportunities because it causes companies to shy away from container adoption. Rather, DevOps teams and security professionals just need to educate themselves on the actions they need to take to reduce risk exposure.

2. DevOps will hire a dedicated security engineer.

The responsibilities assigned to DevOps teams continue to grow. Similar to our first prediction, companies will evolve and create a new position, a dedicated security engineer. This position will sit within the DevOps team, but they will be responsible for managing the security of the Kubernetes platform. At least fifty percent of the Fortune 500 will have this position by the end of 2020.

3. Machine learning will automate security.

Exploitable vulnerabilities are different every time, but attackers leave a recognizable trail of indicators: high resource usage, access to specific data, execution of unusual child processes, etc. Machine learning (ML) lowers the barrier to entry for enterprises implementing cloud security by automating threat detection, policy setting, and compliance reporting. Manually setting policies introduces human error and is impossible at scale. Learning models enable security tools to protect common container images out-of-the-box.

As more tools adopt ML, writing security policies will begin to be considered a bad practice. By the end of 2020, people will be talking about integrating ML into the entire DevOps cycle, from being a part of the software development tools to being integrated throughout the entire CI/CD pipeline. At this level, ML will be able to detect attack vectors even before the software is built.

Deploying Microservices and the Future of Cloud Native

By [Viraj Phanse](#), Product Management and Strategy, Oracle

Over the years, digital transformation has forced enterprise products and platforms to evolve toward a highly scalable architecture that helps achieve simultaneous performance and availability.

In the late 90s and early 2000s, cloud systems were traditionally set up as resource-intensive, monolithic client server systems. These systems were simple to maintain and deployed via a single codebase, database, functionality, and process.

However, the simplicity of the architecture became its own nemesis.

Scaling Application Development: A Brief History

As organizations grew and business requirements became more complex, the need for large-scale, cloud-based systems became evident. The tightly coupled nature of monolithic applications did not meet these new expectations of scale and deployment.

This gave rise to Service Oriented Architecture (SOA) in the late 1990s, a precursor of microservices, whereby an application is divided into smaller components to make it modular and consumable.

Timeline of Key Events

- **2009:** The [Service Oriented Architecture Manifesto](#) was published.

SOA became widely understood and accepted as a paradigm that organizes software via distributed application components called services to help manage, maintain, and deploy internet-scale applications.

- **2012:** The word “microservices” was coined for the first time at 33rd Degree in Krakow.

This new architecture took SOA to the next level by making these services autonomous, disposable, and loosely coupled.

- **2015:** Google, Intel, IBM, VMware, and other industry leaders formed the [Cloud Native Computing Foundation \(CNCF\)](#) to direct distributed, scale-out applications development.

The CNCF led to the birth of a new SDLC paradigm, Cloud-Native Application Development, expanding the concepts of microservices and containers.

Microservices for Cloud-Native App Development

The increasing complexity of enterprise platforms and products that demand performance, scalability, security, reliability, and availability led to the popularity of a microservices-based software architecture.

In this architectural style, an application does not run as a single, monolithic codebase but is broken down into a collection of smaller services that help developers build and modify apps in a rapid, agile manner. They can also focus on developing dynamically orchestrated microservices to run and manage scalable, fault-tolerant systems responsively without worrying about the underlying cloud infrastructure, whether public, private, or hybrid.

Below are two primary modes for orchestrating microservices.

Containers

The power of microservices can be best unleashed by encapsulating them in a lightweight, consistent, self-contained, and isolated workload environment. What's known as containers in the software space offers this in a virtualized system.

At its core, a container:

- Packages code to run a service isolated from its environment (libraries, settings, systems tools, etc.).
- Helps seamless execution irrespective of changing the environment.
- Allows running services in a reliable and scalable manner.

Serverless

Serverless is another primary software architecture style that can be used as a vehicle for microservices. The serverless cloud computing model:

- Breaks down applications into individual functions that run as a service, often referred to as Function as a Service (FaaS).
- Eliminates the headache of managing or hosting underlying cloud infrastructure and hardware.
- Scales functions instantly as the underlying runtime is provisioned on a request basis and for practical purposes.

Third-party cloud vendors such as Amazon AWS and Microsoft Azure take care of server and cloud management. This helps run consumer and enterprise platforms in areas such as telecom, e-commerce, payments, and banking, which particularly need the ability to scale at speed.

Deploying Microservices

Cloud-native app development began with embracing PaaS (Platform as a Service). However, PaaS locked application developers down to a single cloud platform and did not allow sharing of resources.

This limitation led to having only two orchestration approaches favorable to cloud-native app development — containerization and serverless. These two modes of deploying microservices have their own advantages and disadvantages.

On one hand, containers give software developers more control over their environment and how services are packaged. This is extremely helpful in digital transformation initiatives involving migration of legacy applications to the cloud. However, serverless allows teams to focus more on the business capabilities and customer experience and is more applicable in digital transformation initiatives around developing and deploying new-age applications in the cloud.

Table: Comparing Application Deployment Models

	Strengths	Weaknesses	Deployment	
			Runtime Lifetime	Time to Scale
Serverless (FaaS)	<ul style="list-style-type: none"> Eliminates the effort of cloud management Increases ease of deployment and scalability, and a lower latency 	<ul style="list-style-type: none"> Vendor lock-in Expensive 	Milliseconds to seconds	Instant
Containers	<ul style="list-style-type: none"> Provides control over dev environment and how services are packaged Requires fewer resources Cost-effective 	<ul style="list-style-type: none"> Security concerns due to shared Kernel Less flexibility in OS 	Hours to days	Seconds
Platform as a Service (PaaS)	<ul style="list-style-type: none"> Cost-effective Requires minimal development 	<ul style="list-style-type: none"> Locks developers down to a single cloud platform Does not allow the sharing of resources 	N/A	N/A
Virtual Machines	<ul style="list-style-type: none"> Enhanced security due to isolated Kernel 	<ul style="list-style-type: none"> Resource intensive 	Weeks to years	Minutes
Physical Infrastructure	<ul style="list-style-type: none"> Reliability and ease of access for web services 	<ul style="list-style-type: none"> High upfront investment Lots of maintenance overhead 	Years	Days to weeks

Cloud-Native App Development and the Future of Digital Transformation

With the push to cloud-native app development, microservices, containers, and serverless have changed the IT landscape, acting as a catalyst and the base to develop applications at scale in emerging areas like edge computing, Internet of Things (IoT) and Industrial IoT (IIoT), and artificial intelligence (AI) that form the foundation of digital transformation.

Allowing computing near the edge or source of the data, edge computing typically deals with collecting and processing data from thousands of IoT devices through tools such as Docker and Microsoft’s Azure:

- Docker containers help securely distribute software to the edge and run containerized applications on a lightweight, isolated, and granularized framework.
- Azure Kubernetes Service (AKS) allows orchestration of containers on IoT edge devices.

AI requires the underlying architecture to scale in an elastic manner with functional programming paradigm central to the theme. One of the possible ways to achieve AI is by combining the powers of Amazon’s AWS Lambda and Tensorflow. Together, they can help the development and deployment of deep learning systems at scale.


Cloud-native app development has also transformed the software development management landscape. Well suited for agile software development, this new methodology has paved the way toward modern SDLC management tools and practices such as CI/CD and DevOps. It has helped leaders in traditional industries like banking and finance, retail, and manufacturing transform into software companies, and paradoxically help transform software companies into real, hard-core businesses.

Examples of such transformation include:

- **Intesa Sanpaolo** – One of Italy's largest banking groups embraced cloud-native app development to replace their incumbent technologies and systems by CI/CD, DevOps, Kubernetes orchestration to provide customers with a state-of-the-art customer experience.

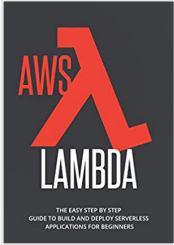
Today, the bank runs more than 3,000 applications. Of those, more than 120 are now running in production using the new microservices architecture, including two of the 10 most business critical for the bank.

- **Netflix** – Realizing that they weren't a data center operations provider, cloud-native application development helped Netflix transform from a new-age software company to a new-age distributor and creator of entertainment content.

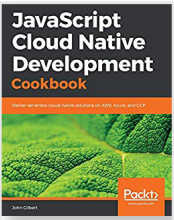
Cloud-native app development and its pillars like microservices and orchestration techniques like containerization aim to make the digital transformation initiatives of enterprises successful by helping them transform their business models into highly scalable, agile, and robust software products and platforms. 

Diving Deeper Into Cloud Native

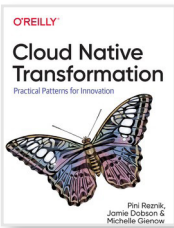
Books



AWS Lambda: The Easy Step-by-Step Guide AWS Lambda is one of the most widely used serverless architectures. This book is an excellent introduction.



JavaScript Cloud Native Development Cookbook Learn how to deliver scalable and serverless cloud-native applications on AWS, Azure, and GCP.



Cloud Native Transformation by Written for enterprises shifting away from legacy apps, this new guide shows how to move an entire organization to the cloud.

Refcards

Introduction to Serverless Monitoring Get an introduction to serverless computing and monitoring, learn how serverless can play a role in IoT and machine learning, see how monitoring and observability differ, and more.

Cloud Native Data Grids: Hazelcast IMDG With Kubernetes Take a deep dive into the concepts surrounding In-Memory Data Grids (IMDGs), including how to deploy them to Kubernetes, what data sources they use, and more.

Hybrid Cloud vs. Multi-Cloud Read how “hybrid cloud” and “multi-cloud” differ, plus learn some of the management challenges in a multi-cloud world and how best to address them.

Zones

Cloud The Cloud Zone covers the host of providers and utilities that make cloud computing possible and push the limits (and savings) with which we can deploy, store, and host applications in a flexible, elastic manner. The Cloud Zone focuses on PaaS, infrastructures, containerization, security, scalability, and hosting servers.

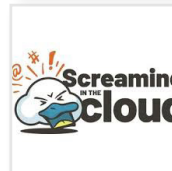
Microservices The Microservices Zone walks you through breaking down the monolith step-by-step and designing microservices architectures from scratch. It covers everything from scalability to patterns and anti-patterns and digs deeper than just containers to give you practical applications and business use cases.

Integration The Integration Zone focuses on communication architectures, message brokers, enterprise applications, ESBs, integration protocols, web services, service-oriented architecture (SOA), message-oriented middleware (MOM), and API management.

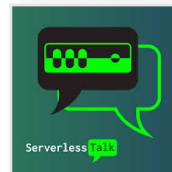
Podcasts



Serverless Chats Dive into a new topic in the serverless space each week.



Screaming in the Cloud Listen to conversations with experts on cloud computing and its diverse business use cases.



ServerlessTalk Some of the most well-regarded developers and engineers in serverless talk



INTRODUCING THE

Cloud Zone

Container technologies have exploded in popularity, leading to diverse use cases and new and unexpected challenges. Developers are seeking best practices for container performance monitoring, data security, and more.

Keep a pulse on the industry with topics such as:

- Testing with containers
- Keeping containers simple
- Monitoring container performance
- Deploying containers in your organization

Visit the Zone



TUTORIALS



CASE STUDIES



BEST PRACTICES



CODE SNIPPETS

Getting Started With Microservices

WRITTEN BY ANDY HAMPSHIRE
GLOBAL ARCHITECT AT TIBCO

CONTENTS

- > Introduction
- > What Are Microservices?
- > Why a Microservices Architecture?
- > Benefits
- > Design Patterns for Microservices
- > Operational Requirements for Microservices
- > The Future, Serverless Computing, and FaaS
- > Conclusion

Introduction

The term "microservices" describes a software architectural style that gives modern developers a way to design highly scalable, flexible applications by decomposing the application into discrete services that implement specific business functions. These services, often referred to as "loosely coupled," can then be built, deployed, and scaled independently.

The "microservices" style is linked to other trends that make this a practical approach. Things like containerization, Agile methods, DevOps culture, cloud services, and the widespread adoption — both culturally and technically — of continuous integration and continuous delivery/deployment (CI/CD) methods across the industry are making it possible to build truly modular, large-scale, service-optimized systems for both internal and commercial use.

This Refcard aims to introduce the reader to microservices and to define their key characteristics and benefits.

What Are Microservices?

Microservices are business functions that are "loosely coupled," which can then be built, deployed, and scaled independently.

Each service communicates with other services through standardized application programming interfaces (APIs), enabling the services to be written in different languages, to use different technol-

ogies and even different infrastructure. The concept differs completely from systems built as monolithic structures, where services were inextricably interlinked and could only be deployed and scaled together. However, microservices do share common goals with EAI and SOA architectures.

As each individual service has limited functionality, it is much smaller in size and complexity. The term "microservice" comes from this discrete functionality design, not from its physical size.

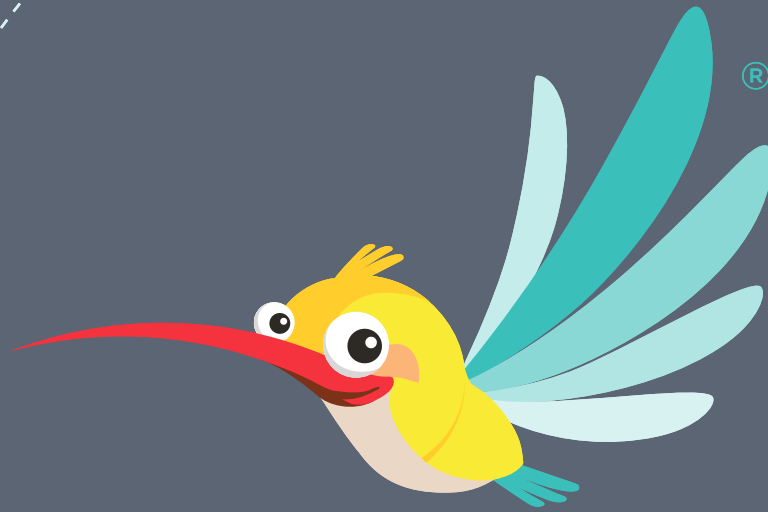


flogo.io 

Microservices Made Easy

[TRY PROJECT FLOGO](#)

An ultra-light, Go-based, open source ecosystem



Microservices Made Easy

TRY PROJECT FLOGO

*An ultra-light, Go-based,
open source ecosystem*

flogo.io 

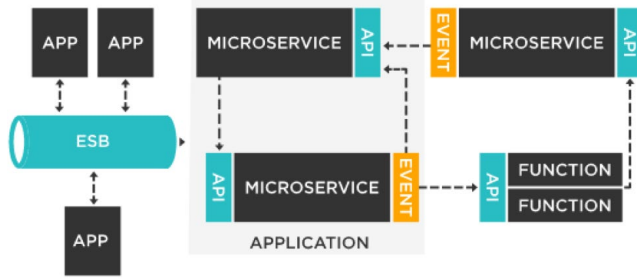


Figure 1: Evolution of Software Application Architectures

Why a Microservices Architecture?

Microservices architectures have risen in popularity because their modular characteristics lead to flexibility, scalability, and reduced development effort. Deployment flexibility, in addition to the rise of cloud-native serverless and function-as-a-service deployment options (such as AWS Lambda and Microsoft Azure Cloud Functions), have created the perfect environment for microservices to flourish in today's IT landscape. These deployment platforms enable microservices and functions to be scaled from inactivity to high volume and back again. Cloud-based services also allow businesses to pay only for the compute capacity they use.

As businesses are continuously looking to be more agile, reduce bottlenecks, and improve application delivery times, microservices architecture continues to rise in popularity. Some of the advantages of using microservices include:

- Application components can be built in different programming languages.
- Each service can be independently deployed, updated, replaced, and scaled.
- Each service is responsible for a single part of the overall functionality and executes it well.
- Use of cloud-native function-as-a-service deployment options is possible.
- "Smart endpoints and dumb pipes" — each microservice owns its domain logic and communicates with others through simple protocols.

Benefits of Microservices

INDEPENDENT SCALING

Each microservice can scale independently. As each instance of a microservice is fully independent, you have the option to deploy multiple instances of a service. This could be on the same hardware, on different machines, on cloud-based infrastructure, or any combination of these. With cloud (both private and public), the ability to scale based on demand means that the service is always able to provide the desired SLAs.

INDEPENDENT UPGRADES

Each service is deployed independently of any other services. Changes local to a service can be easily made by a developer without requiring coordination with other teams. For example, new business requirements and bug fixes can be implemented through updates to the underlying implementation. Where new versions introduce changed APIs, new versions of the service can be built and deployed alongside and service users can migrate to the new version as desired.

EASY(ER) MAINTENANCE

As code in a microservice is limited in functional scope, it should be easier to maintain — its impact on the rest of the code base is far more limited, and its codebase ultimately smaller and easier to understand. Integration testing only needs to be performed at the API level to ensure backward compatibility to other code, so testing can be focused on the business functionality. Anecdotally, this results in systems that are far better tested, as tests are better and more comprehensively targeted.

TECHNOLOGY INDEPENDENCE

Developers are now free to pick the language and tools that are best suited for their service. As interoperability is at the API level, developers are free to innovate within the confines of their service. It also means that any future rewrite of the service can utilize newer technologies, as opposed to being tied to past decisions — thus taking advantage of technological advances in the future.

FAULT AND RESOURCE ISOLATION

With any large application, finding the cause of a code issue like a memory leak or an unclosed database connection can be hard. But with microservices, this can be easier to manage, as periodic restarts of a misbehaving service only affect the users of that service. A smaller, simpler code base often makes finding errors and issues quicker and easier. This improved fault isolation also limits how much of an application a failure can affect, but with critical components, it is still important to understand the cascading impact of failures. However, as faults are isolated to a single service, they can be ultimately resolved independently and quickly.

Design Patterns for Microservices

DEFINING/DECOMPOSING SERVICES

The first big challenge in getting started with microservices is how to identify the services. Developers often think about services as technical services, much like they would when building a functional library. But microservices are much more about solving a business problem than just a technical one.

So, as we start to think about things in terms of business services, it is often tempting to think along object-oriented lines, and start

to create services like *Customer* or *Order*. But this doesn't go far enough. These OO-like services are too much like "God" services, and they need to be broken down into domains more closely aligned to business-led requirements. Getting the balance right is key, but difficult. Techniques like domain-driven design can help you get the balance right for your environment.

EVENT-DRIVEN ARCHITECTURE

Microservices architectures are renowned for being eventually consistent, given that there are multiple datastores that store state within the architecture. Individual microservices can themselves be strongly consistent, but the system as a whole may exhibit eventual consistency in parts. To account for the eventual consistency property of a microservices architecture, you should consider the use of an event-driven architecture where data changes in one microservice are propagated to interested microservices via events.

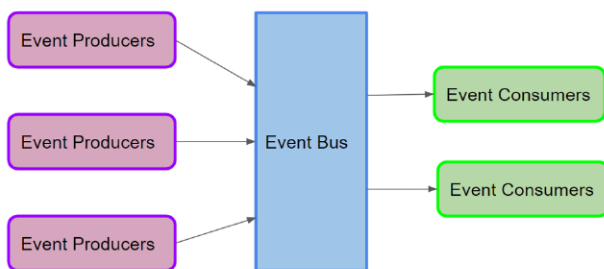


Figure 2: Typical Event Architecture

A pub-sub messaging architecture may be employed to realize the event-driven architecture. One microservice may publish events as they occur in its context, and the events would be communicated to all interested microservices, which can proceed to update their own state. Events are a means to transfer state from one service to another, so that all parts of the application reach an eventually consistent state.

DATABASE DESIGN

Unlike a monolithic application, which can be designed to use a single database, microservices should be designed to use a separate logical database for each microservice. Sharing databases is discouraged and is an anti-pattern, as it makes the database an integration point, stops services being truly independent and introduces a scaling bottleneck. The problems start because the structure of data, the granularity and scope of transactions, and the business requirements will almost certainly overlap.

To solve this conundrum, look at the options to decide what works best in your use case. A database per service scales well, but doesn't work well when multiple services need to share the same data. Sharing databases for related services is a compromise that

is often considered, but has issues with scaling, autonomy, and independence of services. Other options include [Command Query Responsibility Segregation \(CQRS\)](#) and the [Saga data pattern](#). Each has pros and cons that should be considered but are still better than the monolithic database.

API GATEWAYS TO CENTRALIZE ACCESS TO MICROSERVICES

All your microservices will be potentially used by disparate clients, ranging from mobile apps to other microservices. As these clients could be external, or in the case of a mobile client, from third-party applications or part of the same infrastructure, we have the issue of how to manage our interfaces. All services will have an API, but will all the APIs be implemented in the same way — for example, REST and JSON? For external clients, how are you going to manage access to the services? What about security?

This is where an API gateway comes in. It acts as a facade that centralizes the aforementioned concerns at the network perimeter, where the API gateway would respond to client requests over a protocol native to the calling client. The gateway can manage security demands outside the architecture, where clients can identify themselves to the API gateway through a token-authentication scheme like OAuth. It can also manage things like data format changes (XML<->JSON translation) for clients and services that are unable to handle multiple formats.

The gateway also needs to be aware of operational considerations for the services it is proxying, especially where the services are running in a service mesh. Working well with service discovery, service replication, and service migration becomes essential when services can dynamically scale.

Operational Requirements for Microservices

Microservices are not the silver bullet that will solve all architectural or infrastructure problems in your existing applications. Moving to microservices may help, but that could also be just a byproduct of refactoring your application and rewriting code to a new platform. True success requires significant investment in understanding the new technologies and taking advantage of them.

SERVICE REPLICATION

Each service needs to be able to replicate, to manage loads, and to be resilient. There should be a standard mechanism by which services can easily scale based on metadata. A container-optimized application platform such as Kubernetes, CloudFoundry, Amazon EKS, or Red Hat's OpenShift, can simplify this process by defining scaling and recovery rules.

SERVICE DISCOVERY

In a microservice world, multiple services are typically distributed in a container application platform. Service infrastructure is provided

by containers and virtual images, both for on-premise and hybrid-/cloud-orientated deployments. The services may scale up and down based on certain predefined rules, so the exact location of a service may not be known until the service is deployed and ready to be used.

The dynamic nature of a service's endpoint address is handled by service registration and discovery. Each service registers with a broker and provides more details about itself (including the endpoint address). Other consumer services then query the broker to find out the location of a service and invoke it.

There are several ways to register and query services, such as ZooKeeper, etcd, Consul, Kubernetes, Netflix Eureka, and others. Kubernetes, in particular, makes service discoverability very easy, as it assigns a virtual IP address to groups of like resources and manages the mapping of DNS entries to those grouped resources.

SERVICE OBSERVABILITY

Some of the most important aspects of managing a microservices-based architecture are service monitoring, metrics, and logging. Through metrics, you can understand how an application runs in its normal state. You can then understand what's not normal, enabling you to take proactive action if, for example, a service is consuming unexpected resources. Elasticsearch, Fluentd, and Kibana can aggregate logs from different microservices, provide a consistent visualization, and make that data available to business users. When things go wrong, distributed tracing tools like Zipkin can give you a holistic view of the end-to-end process alongside performance metrics. This can help find and solve most runtime problems.

RESILIENCE

Software failures will occur, no matter how much or how well you test. This becomes all the more important when multiple microservices are deployed to different platforms. The key concern is not "how to avoid failure" but "how to deal with failure." It's important for services to automatically take corrective action to ensure user experience is not impacted. The [Circuit Breaker](#) pattern allows you to build in resiliency — Netflix's Hystrix is a good library that implements this pattern.

DEVOPS

Continuous integration and continuous delivery/deployment (CI/CD) are very important in order for microservices-based applications to succeed. These practices ensure that bugs are identified via automated testing and human factors are removed by automated deployment pipelines.

On the CI side, the pipeline will take code from a developer's checked-in source, build it, and deploy it to a testing environment where it is tested both in isolation and in relation to service users.

On the CD side, there are two options. First is "Delivery," where the tested code is sent to the production repository ready for final deployment. Second is "Deployment," where the code is deployed automatically to the production environments.

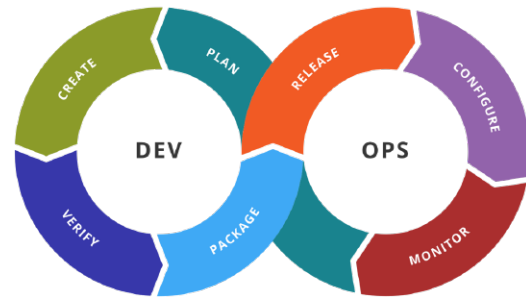


Figure 3 - The DevOps Cycle [1]

DEPLOYMENT

Being able to automate the Delivery/Deployment process is essential to a successful microservices program. But automated deployment through to production needs to be carefully managed. Regardless of whether deployment is fully automated or not, the [Blue/Green](#) deployment pattern is a good way of managing risk. In this scenario, existing services are running on the "green" system, and updated services are deployed to the "blue" system. The request routing (API gateway or service mesh) manages the switchover to the new version of the service, often in a phased manner to ensure that the service introduces no problems.

The Future, Serverless Computing, and FaaS

When people first start experimenting with microservices, they often default to using familiar techniques — for example, simple request/reply services based on RESTful APIs. The problem with this synchronous approach is that, as you have to wait for a response, the services become dependent on each other. If one service is running slower or doesn't respond, it means the service that called it will run slower or fail with a timeout. This coupling can mean losing some of the benefits of a microservices architecture, creating a more interdependent structure akin to a Request/Reply Service-Oriented Architecture (SOA) style.

If you design your services using an event-driven or Pub/Sub model, you can ensure that parts of your application continue to work, as opposed to the entire application becoming unresponsive. Take Netflix as an example. On occasion, you might notice that the "continue watching" button doesn't appear. This is because a specific service isn't available. However, that doesn't mean all of Netflix stops. Users can still browse for shows and watch previews, so other aspects of the Netflix service are still available, even though one service may not be.

Developers that fully embrace a microservices approach realize that true scalability is enabled with loose coupling and event-driven architecture. A service can be asynchronous, performing an action, broadcasting a message, and continuing on with its primary function without having to wait for a response from another service. This lends well to the adoption of Serverless and Function-as-a-Service (FaaS) platforms going forward, giving businesses easy access to "on-demand" capacity at a competitive price.

Conclusion

The microservices architectural style has well-known advantages. It can certainly help businesses evolve and innovate faster.

Consider the operational requirements of microservices carefully, in addition to the benefits, before moving to a microservices architecture — especially if you are refactoring an existing monolithic application. Better software engineering, organizational culture, and architecture will be enough to make your existing structure more agile, without having to jump to microservices. In this case, a natural migration may be a better way to proceed. But for new applications, microservices and FaaS are probably the best options available to us today.



Written by **Andy Hampshire**

Andy Hampshire has worked in the IT industry for over 30 years (been there, done that, and got quite a few t-shirts). Andy loves most forms of motorsport, so when Andy is not playing with (or working on) his cars and bikes he can often be found watching racing, at the moment mostly at Kart tracks supporting his son's racing "career". Away from work and cars, his idea of heaven is walking with his dog in the Surrey Hills where he lives, as far away from the rest of humanity as possible!



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399 919.678.0300

Copyright © 2019 Devada, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Designing Microservices With Cassandra

JEFFREY CARPENTER

DIRECTOR OF DEVELOPER ADOPTION, DATASTAX

CONTENTS

- Benefits of Microservices Architecture
- Identifying Bounded Contexts
- Identifying Services
- Designing Microservice Persistence
- Design Choices for a Java Microservice
- Deployment and Integration Considerations
- Additional Resources

Editor's Note: The following is adapted from "Cassandra: The Definitive Guide" (3rd Ed., O'Reilly), by Jeff Carpenter and Eben Hewitt, with permission from O'Reilly Media: <https://dtsx.io/oreilly>

Over the past several years, the microservice architectural style has been foundational to the discipline of cloud-native applications. As a database designed for the cloud from the ground up, Apache Cassandra is a natural fit for cloud-native applications.

In this Refcard, we explore techniques for developing Cassandra data models and designing microservices based on those models for a sample application used to manage hotel reservations.

BENEFITS OF MICROSERVICE ARCHITECTURE

We will start our discussion by referencing a subset of the basic principles of microservice architecture that are introduced in Sam Newman's book, [Building Microservices](#) (O'Reilly), an excellent source on this topic.

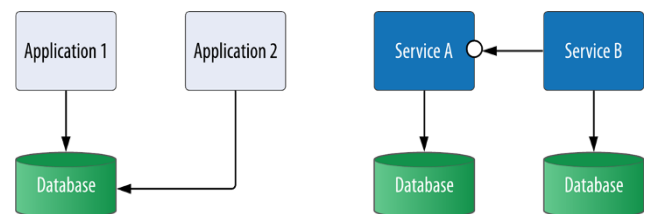
ENCAPSULATION

Encapsulation could also be phrased as "services that are focused on doing one thing well" or the "single responsibility principle." In a microservice architecture, this means that each service should manage its own data storage and not access data stores managed by other services.

By contrast, in many enterprises, the database serves as a central integration point. An application might expose interfaces to other

applications such as remote procedure call (RPC) or messaging interfaces. It's also common for one application to access another application's database directly, which violates encapsulation and produces dependencies between applications that can be difficult to isolate and debug (see Figure 1).

Figure 1: Integration by database contrasted with microservices



O'REILLY **DATASTAX**

Cassandra: The Definitive Guide, 3E

Distributed Data at Web Scale

[Get Free Ebook](#)

How to harness Cassandra's speed and flexibility with ease.



**Cassandra:
The Definitive
Guide, 3E**

[Get Free Ebook](#)

AUTONOMY

In a microservice architecture, *autonomy* refers to the ability to independently deploy each microservice without dependence on any other microservices.

This flexibility has significant advantages in allowing you to independently evolve portions of a deployed application without downtime, gradually introducing new versions of a service and minimizing the risk of these deployments.

Another implication of autonomy is that each microservice can have its own data store using the most appropriate technology for that service. We'll examine this flexibility in more detail below.

SCALABILITY

Microservice architecture provides a lot of flexibility by enabling you to run more or fewer instances of a service dynamically according to demand. This allows you to scale different aspects of an application independently.

For example, in a hotel domain there is a large disparity between *shopping* (the amount of traffic looking for hotel rooms) and *booking* (the much lower level of traffic committing to a reservation).

For this reason, you might expect to scale the services associated with hotel and inventory data to a higher degree than the services associated with storing reservations.

MICROSERVICE ARCHITECTURE FOR A HOTEL APPLICATION

To create a microservice architecture for an application, you'll need to identify services, their interfaces, and how they interact.

Although written well before microservices became popular, Eric Evans' book, *Domain-Driven Design* (Addison-Wesley Professional), has proven to be a useful reference.

One of the key principles that Evans articulates is beginning with a domain model and identifying bounded contexts. This process has become a widely recommended approach for identifying microservices.

For Cassandra, the recommended method for data modeling is to use a query-driven approach to identify tables.

For example, after identifying the key entities in a domain by creating a conceptual data model, you can analyze the anticipated workflows of the application to identify how the entities will be read by the application.

The resulting queries are used to identify the tables that will be created, which can be documented in logical and physical data models. You can see an example of how this process works in chapter five of [Cassandra: The Definitive Guide](#), which is freely available as part of the [Cassandra project documentation](#).

To define a microservice architecture, use a process that complements these data modeling processes. As you begin to identify entities as part of a conceptual data modeling phase, you can identify bounded contexts that represent groupings of related entities.

As you progress into logical data modeling, you refine the bounded contexts to identify specific services that will be responsible for each table (or group of related, denormalized tables).

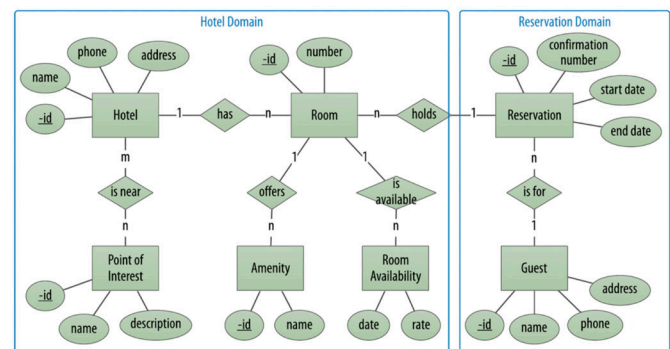
During the final stage of the design process, you confirm the design of each service, database selection, physical data models, and actual database schema.

Let's see how this high-level process works in practice for a hotel reservation application.

IDENTIFYING BOUNDED CONTEXTS

Using a conceptual data model for a hotel domain, you might choose to identify a Hotel Domain — encompassing information about hotels, rooms, and availability — and a Reservation Domain to include information about reservations and guests (see Figure 2).

Figure 2: Identifying bounded contexts for a hotel application

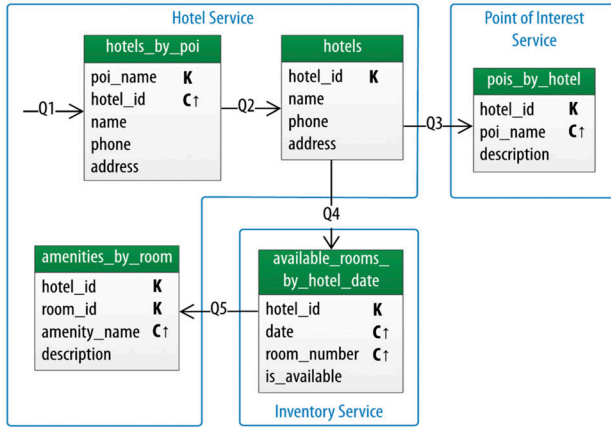


IDENTIFYING SERVICES

The next step is to formalize the bounded contexts you identified into specific services that will own specific tables within your logical data model.

For example, the Hotel Domain identified previously using the conceptual data model might decompose into separate services focused on hotels, points of interest, and inventory availability, overlaid on a logical data model (see Figure 3).

Figure 3: Identifying services for hotel data



Key design principle: Assign tables that have a high degree of correspondence to the same service.

In particular, with Cassandra, a natural approach is to assign denormalized tables representing the same basic data type to the same service.

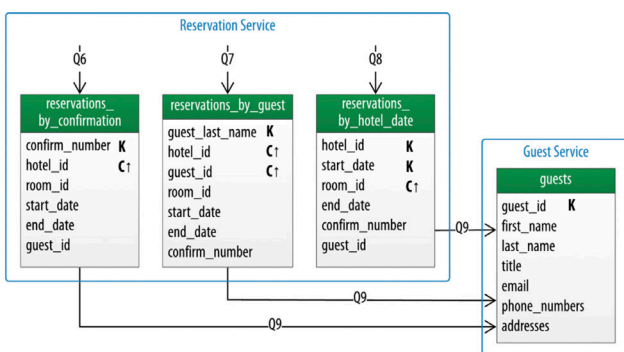
For example, the Hotel Service would manage the `hotels` and `hotels_by_poi` tables. It might also be a good candidate to manage the `amenities_by_room` table since room data would typically only be accessed in the context of a particular hotel.

Key design principle: Services should embody classic object-oriented principles of coupling and cohesion.

There should be a high degree of cohesion, or relatedness, between tables owned by a service and a low amount of coupling, or dependence, between contexts.

Using the same principles as above, examine the tables in a logical data model within the Reservation Domain. You might identify a Reservation Service and a Guest Service (see Figure 4).

Figure 4: Identifying services for reservation data



In many cases, there will be a one-to-one relationship between bounded contexts and services, although with more complex domains, there could be further decomposition into services.

DESIGNING MICROSERVICE PERSISTENCE

The final stage in the data modeling process consists of creating physical data models. This corresponds to the architectural tasks of designing services, including database-related design choices such as selecting a database and creating database schema.

POLYGLOT PERSISTENCE

One of the benefits of microservice architecture is that each service is independently deployable. This gives you the ability to select a different database for each microservice, an approach known as *polyglot persistence*.

Cassandra might not be the ideal backing store for all microservices, especially for those that do not require the scalability that Cassandra offers. Let’s examine the services that are identified in the hotel application design to determine some options for polyglot persistence (see Table 1).

Table 1: Polyglot persistence example

SERVICE	DATA CHARACTERISTICS	DATABASE OPTIONS
Hotel Service	Descriptive text about hotels and their amenities, changes infrequently	Document database (e.g., MongoDB), Cassandra, or Elasticsearch/Solr for full text search
Point of Interest Service	Geographic locations and descriptions of points of interest	Cassandra or other tabular databases supporting geospatial indexes (e.g., DataStax Enterprise)
Inventory Service	Counts of available rooms by date, large volume of reads and writes	Cassandra or other tabular databases
Reservation Service	Rooms reserved on behalf of guests, lower volume of reads and writes than inventory	Cassandra or other tabular databases
Guest Service	Guest identity and contact information, possible extension point for customer and fraud analytics systems	Cassandra, graph databases

While the initial demand on your services might not require Cassandra’s elastic scalability, you should make selections with an eye to future extensibility and scalability of the system.

RESERVATION SERVICE: A SAMPLE MICROSERVICE

Now that you have identified the candidate services for a hotel application and considered how service design might influence your Cassandra data models, it's time to examine the design of individual microservices.

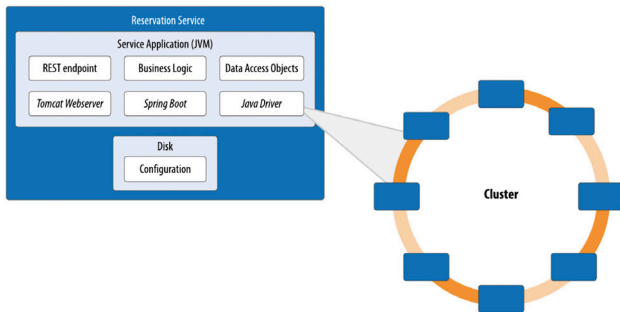
DESIGN CHOICES FOR A JAVA MICROSERVICE

Let's narrow our focus to the design of a single service — the Reservation Service, which will be responsible for reading and writing data using the tables identified above.

Figure 5 shows a candidate design for a Java implementation of the Reservation Service using popular libraries and frameworks.

This implementation uses Apache Cassandra for its data storage via the DataStax Java Driver and uses the Spring Boot project for managing the service life cycle. It exposes a RESTful API documented via Swagger.

Figure 5: Reservation Service Java design



Find the [Reservation Service Java implementation on GitHub](#).

The goal of this project is to provide a minimally functional implementation of a Java microservice using the DataStax Java Driver, which can be used as a reference or starting point for other applications.

DEPLOYMENT AND INTEGRATION CONSIDERATIONS

There are a couple of factors you should consider that are related to how the service will be deployed and integrated with other services and supporting infrastructure.

SERVICES, KEYSAPACES, AND CLUSTERS

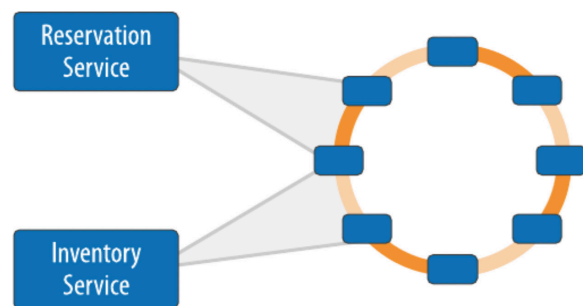
First, consider the relationship of services to keyspaces. In Cassandra, keyspaces are used to group tables with related replication and access control settings.

A good rule of thumb is to use a keyspace per service to promote encapsulation.

Cassandra's access control features allow you to create a database user per keyspace, such that each service can be easily configured to have exclusive read and write access to all of the tables in its associated keyspace.

Next, consider whether a given service will have its own dedicated Cassandra cluster or share a cluster with other services. Figure 6 depicts a shared deployment in which Reservation and Inventory Services use a shared cluster for data storage.

Figure 6: Service mapping to clusters



Companies that use both microservice architectures and Cassandra at large scale (e.g., Netflix) are known to use dedicated clusters per service. The decision of how many clusters to use will depend on the workload of each service.

A flexible approach is to use a mix of shared and dedicated clusters, in which services that have lower demand share a cluster, while services with higher demand are deployed with their own dedicated cluster.

Sharing a cluster across multiple services makes sense when the usage patterns of the services do not conflict.

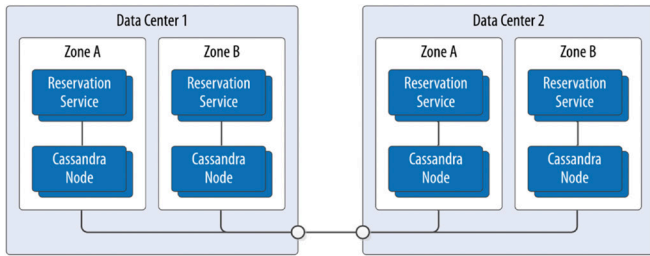
DATA CENTERS AND LOAD BALANCING

A second consideration is the selection of data centers where each service will be deployed. The corresponding cluster for a service should also have nodes in each data center to enable the fastest possible access.

Figure 7 shows a sample deployment across two data centers.

The service instances should be made aware of the name of the local data center. The keyspace used by a service will need to be configured with a number of replicas to be stored per data center, assuming the `NetworkTopologyStrategy` is the replication strategy in use.

Figure 7: Multiple data center deployment



Most of these options — such as keyspace names, database access credentials, and cluster topology — can be externalized from application code into configuration files that can be more readily changed. Even so, it's wise to begin thinking about these choices during the design phase.

INTERACTIONS BETWEEN MICROSERVICES

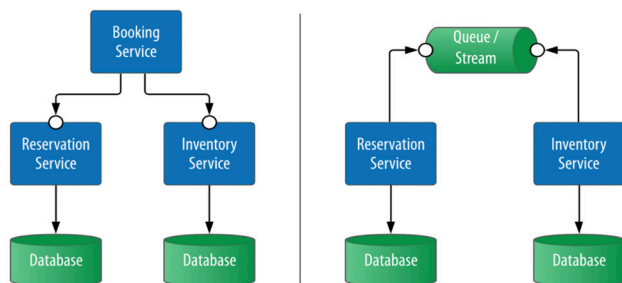
One question that arises when developing microservices that manage related types is how to maintain data consistency between the different types. If you want to maintain strict ownership of data by different microservices, how can you maintain a consistency relationship for data types owned by different services?

Cassandra does not provide a mechanism to enforce transactions across table or keyspace boundaries. This problem isn't unique to Cassandra, since you'd have a similar design challenge when you need consistency between data types managed by different services, regardless of the backing store.

Let's look at the hotel application for an example.

Given the separate services to manage inventory and reservation data, how do you ensure that the inventory records are correctly updated when a customer makes a reservation? Figure 8 shows two common approaches to this challenge.

Figure 8: Service integration patterns



The approach on the left side is to create a Booking Service to help coordinate changes to reservation and inventory data.

This is an instance of a technique known as *orchestration*, often seen in architectures that distinguish between so-called *CRUD services* (responsible for creating, reading, updating, and deleting a specific data type) and services that implement business processes.

In this example, the Reservation and Inventory Services are more CRUD services, while the Booking Service implements the business process of booking a reservation. This includes reserving inventory and possibly other activities such as notifying the customer and hotel.

An alternative approach is depicted on the right side of Figure 8, in which a message queue or streaming platform such as Apache Kafka is used to create a stream of data change events that can be consumed asynchronously by other services and applications.

For example, the Inventory Service might choose to subscribe to events related to reservations published by the Reservation Service to make corresponding adjustments to inventory.

Because there is no central entity orchestrating these changes, this approach is instead known as *choreography*.

It's important to note that both orchestration and choreography will require careful planning to address error cases such as service and infrastructure failures. Techniques and technologies to address such error cases include the following:

Distributed transaction framework

Used to coordinate changes across multiple services and databases.

This can be a good approach when strong consistency is required. [Scalar DB](#) is an interesting library for implementing distributed ACID transactions that is built using Cassandra's lightweight transactions as a locking primitive.

Distributed analytics tool (e.g., Apache Spark)

Used to check data for consistency as a background processing task.

This approach is useful as a backstop for catching data inconsistencies caused by software errors, in situations in which there is tolerance for temporary data inconsistencies.

A variant of the event-based choreography approach

Used to leverage the change data capture (CDC) feature of a database as the source of events, rather than relying on a service to reliably persist data to a database and then post an event.

This approach is typically used to guarantee highly consistent interactions at the interface between applications, although it could be used between individual services.

CONCLUSION

As you've learned, Cassandra is a natural fit for the persistence layer of a microservice-style architecture. In this Refcard, we examined techniques for identifying bounded contexts and services based on data models — and designing microservices and their interactions.

ADDITIONAL RESOURCES

- [Cassandra: The Definitive Guide \(3rd Ed.\)](#)
- [Reservation service code on GitHub](#)
- [DataStax Java Driver](#)

WRITTEN BY JEFFREY CARPENTER,

DIRECTOR OF DEVELOPER ADOPTION, DATASTAX



Jeff Carpenter works in developer relations at DataStax, where he uses his background in system architecture, microservices, and Apache Cassandra to help empower developers and operations engineers to build distributed systems that are scalable, reliable, and secure. Jeff has worked on large-scale systems in the defense and hospitality industries and is co-author of *Cassandra: The Definitive Guide*.



DZone, a Devada Media Property, is the resource software developers, engineers, and architects turn to time and again to learn new skills, solve software development problems, and share their expertise. Every day, hundreds of thousands of developers come to DZone to read about the latest technologies, methodologies, and best practices. That makes DZone the ideal place for developer marketers to build product and brand awareness and drive sales. DZone clients include some of the most innovative technology and tech-enabled companies in the world including Red Hat, Cloud Elements, Sensu, and Sauce Labs.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC 27709

888.678.0399 919.678.0300

Copyright © 2020 Devada, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means of electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Managing Microservices

CONTENTS

- > Overview
- > Microservices and FinTech: A Use Case
- > Microservices Benefits and Requirements
- > How to Manage Microservices
- > DevOps + Microservices
- > Conclusion

WRITTEN BY ARIFF KASSAM
CHIEF TECHNOLOGY OFFICER, NUODB

Overview

The promise of microservices is the ability to increase developer agility by breaking the application into smaller, more manageable components with clear interfaces. As a result, making changes to a microservice requires less coordination between components and less testing. These smaller components significantly improve the agility, scalability, and availability of applications, which offers significant benefits to developers as they seek to deploy new application features rapidly to meet customer demands.

While unquestionably adding to agility, microservices aren't for every application. When starting with microservices, it's important to remember that it's not something you can adopt in a vacuum. Microservices require significant development and delivery skills, including security built in and automated at every layer of development, a mature DevOps environment, and a high degree of standardization and automation using technology such as containers and [Kubernetes](#) for container orchestration.

While microservices can be used in any industry, right now it makes the most sense to adopt them in organizations that use Agile development methodologies and need to make changes to customer facing applications quickly. Highly regulated industries and those that use Waterfall development methodologies and have less frequent software releases, such as healthcare, government, oil and gas, and manufacturing, may not benefit from a move to microservices architectures. Other industries, such as financial services, already have an agile development

environment and a customer base that demands innovation and rapid delivery and, therefore, benefit greatly from the adoption of microservices. Industries that require services to always be available, reliable, and responsively scalable based on real-time demand benefit from a move to microservices.

Microservices and FinTech: A Use Case

Financial organizations have been facing significant competition and disruption due to the introduction, and now expectation, of technology that allows end users to rapidly make mobile payments, transfer



With NuoDB you can:

- + Scale out a container-native SQL database on demand
- + Deliver stateful apps in Kubernetes with zero downtime
- + Deploy on-prem or in public or hybrid cloud

Don't believe it? See for yourself.

Download the **FREE**
Community Edition Today

Your microservices are portable,
flexible, and scalable.

Is your database?

Learn why NuoDB is the database to build your future on:
www.nuodb.com/financial-services



NuoDB's cloud-native distributed SQL database helps enterprise organizations overcome the complex challenges faced when trying to move enterprise-grade, transactional applications to the cloud.

Learn more at nuodb.com.

money, make or request loans, conduct fundraising, and access wealth management tools. FinTech, or financial technology, encompasses any type of technology in financial services, from mobile payment apps to cryptocurrency.

FinTech originally referred to technology applied to financial institutions' back-end systems, but today includes many more consumer-focused applications, including smartphone apps to manage funds and make payments, trade stocks, exchange cryptocurrency, and make budgets. FinTech isn't just for consumers, however. It can also provide better financial services to businesses, such as expense tracking, accounting software, employee payment, and even sales tracking and invoicing.

Many banks and early FinTech companies were built on legacy systems, which they've relied on as they build new apps and interfaces, but the shift towards digital services necessitates greater flexibility and agility. Microservices are a perfect fit, as financial services organizations seek agility and scalability, but may require significant cultural and architectural shifts. A few important Agile and DevOps competencies for application delivery teams looking to make the transition to microservices include:

- Security embedded in DevOps processes.
- Continuous Integration and Continuous Delivery (CI/CD).
- Automation of Core Infrastructure and Releases.

Microservices Benefits and Requirements

PROGRAMMING LANGUAGE AGNOSTIC

Microservices do not force any specific programming model, style, or language due to their technology independence. As each microservice communicates with other services through standard channels like APIs, they don't rely on technology-related restrictions. This enables development teams to choose the programming language that works best for each microservice, as well as choosing a particular pattern or database based on what's best for their use case. Using microservices and containers, a single instance can host native code, .NET, Java, or any other programming language that is the best fit for the particular microservice.

REGULATIONS

New regulations are emerging to address the evolving digital payment economy, which aim to address questions about the use and ownership of data, operational resilience and business continuity, and increasing competition. In the European Union, the Payment Services Directive version 2, or PSD2, aims to bring banking into the open API economy, which will drive interoperability and integration through open standards. In order to meet these standards, financial services will benefit from the advantages microservices bring to APIs, integration, and open data requirements.

LOGGING AND MONITORING

Logging and monitoring go hand in hand with security, and are essential for successful microservices deployment. Since microservices architecture is highly distributed, it can be hard to determine where a failure

occurred and what caused it without logging and monitoring tools. Due to the fact that microservices break an application into many smaller components, logging generates a lot of data, so it's helpful to choose tools that help you parse or visualize the results. [Prometheus](#) is an open source application used for event monitoring and learning.

CONTINUOUS INTEGRATION (CI)/CONTINUOUS DELIVERY (CD)

There are many continuous integration build systems available that provide access to pipeline builds, such as TeamCity, Bamboo, and Jenkins. Microservices scale extremely well, accommodating increasing numbers of users and transactions and delivering new functionality rapidly. Responding in real time to demand isn't an option without an effective implementation of a CI/CD solution.

INTEGRATION

Microservices make it easier to manage and secure the API layer through isolation, scalability, and resilience. The APIs enable easy communication with both internal and external services. Well-defined APIs enable you to prevent dependencies between microservices while still enabling the exchange of data.

SECURITY

When deploying microservices, it's important to build security into your development processes. By embedding security processes and automating them, it's possible to bolster security in your applications. The platform used to deploy microservices should provide developers with options for identity and access control and authorization (such as OAuth), certificate management capabilities, automated security updates, periodic automated vulnerability and security scanning, and control over the container images available for use to limit security risks.

DATA PERSISTENCE

A microservices architecture isolates each microservice from the others, so you have a choice when addressing data storage and data persistence. This is not the place to permit open access to a single monolithic database, which traditional monolithic development environments frequently allow. If your microservices can communicate with one another through the database, you're likely to see unexpected coupling. Different microservices each own the data related to the business functionality supported by that service, and they may require different options. Some will best use a NoSQL storage system, while others require a SQL-based relational database. Regardless, each microservice should use unique credentials and access should be limited to that microservice's data. Some examples of databases that pair well with microservices are:

- [MongoDB](#), an open source NoSQL database.
- [NuoDB Community Edition \(CE\)](#), a free distributed SQL database.

CONTAINERS

Containers are the most common deployment mechanism for microservices, in part because they deliver many of the same benefits that microservices offer, such as platform independence, scalability, isolation,

resource efficiency, and speed. Because containers can easily be scaled out and back, it's relatively simple to spin up a number of instances of a specific version of your service for development, test, staging, and production environments.

DISTRIBUTED SYSTEMS

Microservices architecture enables the concept of decentralized data management, supporting distributed deployment. Microservices deployed in containers allow you to reprovision any container individually, or replace, deprecate, or add new microservices when there's a new feature to release, a vulnerability to patch, a bug to resolve, or any other change that you want to roll out rapidly. They're also simple to scale independently and responsively, so you can scale out based on demand for a particular service without scaling the entire application.

LOAD BALANCING AND RESILIENCY

In a microservices environment, load balancing functionality is typically moved into the software layer, performing the load balancing logic at the distributed edge. Microservices also provide resiliency by handling errors through retries, queuing, deadlines, and default and caching behaviors. When done correctly, microservices architecture can help you deliver self-healing applications that operate even when there are partial outages, automatically deploying new containers that allow your application to recover quickly and seamlessly.

MANAGING SERVICES

When first starting with microservices, your microservices architecture is likely to be small, and managing them seems simple. However, as the number of microservices grows, you need to start thinking about the macro-architecture of your microservices environment. There are many solutions that offer infrastructure, often referred to as service meshes. Service meshes provide a **control plane**, which sets the policy that will be enacted by the data plane through configuration files, API calls, and user interfaces, and a **data plane**, which translates, forwards, and observes every network packet that flows to and from a service instance.

- [Linkerd](#) is an open source service mesh that acts as a proxy between services and provides load balancing, circuit breaking, service discovery, dynamic request routing, HTTP proxy integration, retries and deadlines, TLS, transparent proxying, distributed tracing, and instrumentation. Protocol support includes HTTP/1.x, HTTP/2, gRPC, and anything TCP-based.
- [Istio](#) is another widely used open source service mesh and provides automatic load balancing, fault injecting, traffic shaping, timeouts, circuit breaking, mirroring, and access controls for HTTP, gRPC, WebSocket, TCP traffic, automatic metrics, logs, and traces, and infrastructure level run-time routing of messages.

How to Manage Microservices

With the rise of popularity of microservice architectures, containers are now the best deployment mechanism for microservices. As noted above, containers provide many benefits, including platform independence,

resource efficiency, speed, and isolation. They also enable flexibility and scalability for organizations deploying applications, because containers can easily be scaled out and back individually based on demand for those services.

As containers have grown in popularity, managing them at scale is the next real challenge, which created a need for container orchestration. In response to this need, Google released Kubernetes as an open source platform for automating deployment, scaling, networking, managing, and maintaining availability for container-based applications. Using microservices, containers, and container orchestration tools together can simplify running applications in the cloud, which greatly improves business agility.

Using microservices, containers, and container orchestration, developers now have on-demand access to IT resources and the architectural paradigms that help them speed up the process of both development and moving code from the dev environment to production. This significantly improves your ability to transform slowly maturing applications into adaptable containerized microservices. This is particularly true when building applications that are stateless.

Building and deploying stateless applications in containers is relatively easy. Every time you start a stateless application it has the same information that it does every time you start it, which makes them easy to scale horizontally to accommodate increased user demands (add more instances) and protect against failures (start new instances).

Many applications, however, require persistent state. That means that these applications require the ability to store data to protect against failures so that the application will not lose any data. Traditionally, stateful applications have been much harder to fit into the world of containers. While databases are the standard for managing state for applications, traditional databases have a number of issues when managed by a container orchestration solution, typically Kubernetes.

EXTERNAL PROCESS LIFECYCLE MANAGEMENT

Kubernetes automatically distributes running containers across the cluster, which is one of the advantages it provides. If a machine fails, any containers running on that system are automatically restarted on other nodes in the cluster. Kubernetes also automatically rebalances container distribution periodically; it's fairly common for containers to be stopped and then restarted on different nodes. Kubernetes controls the lifecycle management of container processes.

It's important to note that this lifecycle management is simple for stateless containers. Stateless containers can be started and stopped at any time, and stateless containers can be run on any node in the cluster. As long as you have at least one instance of the container running at any time, the service that application provides is always available.

Stateful containers aren't as flexible, partly because the state information needs to be accessible on any node to which the container can be

moved. Kubernetes recently added container-native storage solutions to allow the state to be accessed in this way. Issues related to container lifecycle management remain, such as: to migrate a container from one node to the other, Kubernetes shuts down the current container and starts a new container. It's possible that the two container instances (new and old) briefly run concurrently during this time, which means that applications could connect to either instance.

Traditional databases can't handle this scenario, because there can only be a single "active" instance of the database at any given time. All data written to the database instance being shut down is lost while the new container becomes the active one. To work in containers and the Kubernetes orchestration environment, databases must be capable of handling multiple processes running at the same time *without* any data loss.

SCALE OUT

By design, Kubernetes addresses performance issues by deploying more containers, thus enabling horizontal scale out. This horizontal scale out is simple for stateless applications. However, because traditional databases only support a single "active" process, they require scale up, not scale out. Scale up doesn't translate well to the Kubernetes environment. Kubernetes and containers are built to scale out based on demand, using as many or as few processes as necessary to handle throughput. Traditional relational databases can't spawn new Kubernetes pods; that would require a more expensive machine or necessitate that you shard your database. To work well in Kubernetes, you need a database that can scale out for both reads and writes on demand, not one limited by a single server.

CONSISTENCY

In scale out architectures, there are multiple instances of the container. For stateful containers, it's important that clients are able to connect to any instance of the container and receive a consistent view of the data. Different scale out databases have different consistency models. It's important for application developers to understand the consistency model supported by the database they are using.

For some applications, eventual consistency works. Some databases are able to achieve high availability in distributed environments using eventual consistency. With this consistency model, the application must be able to handle consistency conflicts. This may require significant application changes to support that ability. Many applications being migrated to the cloud and containers require a stricter consistency model, particularly applications handling business-critical data.

DevOps + Microservices

DevOps is an evolving philosophy, and its goals are to tightly link the development of

software and its delivery to IT Operations, thus improving the quality of the software systems as a whole. Much like microservices, a DevOps approach accomplishes this by segmenting the system into manageable

components, which are owned by teams that can resolve issues that prevent the system from operating properly.

Most DevOps advocates consider CI and CD defining attributes of DevOps. Continuous Integration allows developers to integrate changes into the source code mainline as soon as they're completed, which is easier when creating microservices because there's less testing needed when each component is built to operate independently. Likewise, Continuous Delivery allows microservices to be updated as needed.

Automating the process using CI/CD tools is also essential for successful adoption of DevSecOps, which is when security is automated and integrated within DevOps. Including (and automating) security tools into the DevOps process is essential, because there simply isn't time in a mature microservices environment for security to be an afterthought. To build an environment in which microservices and security co-exist, you must develop both a plan and a framework for development, governance, and management of microservices.

OPERATORS

Kubernetes Operators help encode the human operational logic normally required to manage services running of a Kubernetes-native application and aim to make day-to-day operations easier. Operators on application container platforms, such as [Red Hat OpenShift](#) and [Rancher](#), can help end users experience the next level of benefits from a Kubernetes-native infrastructure, with services designed to work across any cloud where Kubernetes runs. As microservices are typically delivered via containers, Operators are an important part of the deployment process for deploying stateful applications in Kubernetes.

Kubernetes Operators and operator catalogs, such as the new OpenShift OperatorHub and [OperatorHub.io](#), take complicated technical solutions and make deploying them simple. When Operators were first made public in a [2016 CoreOS blog post](#), the goal of Operators was to make the software itself include operational knowledge that previously resided outside of the Kubernetes cluster. Operators simplify that process by implementing and automating the most common Day-1 and Day-2 activities in a piece of software running inside the Kubernetes cluster.

Operators make the process of modernizing existing applications and building new applications a lot easier. While Kubernetes has made it pretty easy to manage and scale web apps, mobile backends, and API services, until recently it's been more difficult to manage stateful applications such as databases, caches, and monitoring systems. The new application domain knowledge contained in Operators makes it possible to scale, upgrade, and configure these types of applications in Kubernetes in multiple pods across the cluster.

Using the Operator Lifecycle Manager (OLM), users can subscribe to an Operator --- including individual channels, such as stable vs. beta releases, so subscribers are continuously updated to the latest version and its new capabilities.

Conclusion

Cloud-native applications built on cloud-native infrastructure make it possible to increase the velocity of software delivery, enable developers to be more agile, and allow greater application scalability. By developing and delivering applications using microservices, containers, and Kubernetes, technology innovators can deliver the agility, scalability, and availability of applications that modern businesses and consumers demand.

In this Refcard, we've reviewed how microservices require the right infrastructure and technical skills within your organization. While microservices provide many advantages, there's a lot to consider when deploying your microservices-based infrastructure. From the considerable benefits of providing a programming language agnostic framework to the ability to respond quickly to changing regulations to the tools and skills essential for effective logging, monitoring, and integration, microservices enable a new degree of flexibility that many developers and engineers will be quick to embrace.

Using microservices deployed in containers, application container platforms, Kubernetes, and Operators, you bring many powerful tools together. These solutions enable applications to scale well to accommodate increasing numbers of users and transactions and deliver new functionality rapidly, which is essential for distributed systems. As you build new applications and redesign legacy applications, microservices will serve you well, provided you select solutions that portable, flexible, and scalable.



Written by **Ariff Kassam**, Chief Technology Officer, NuoDB

Ariff is responsible for defining and driving NuoDB's product strategy. Kassam brings 20 years of database and infrastructure experience to NuoDB to help the company achieve its vision of a distributed database that can manage an organization's most valuable data while exploiting the emerging benefits of modern infrastructures such as cloud and containers.



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399 919.678.0300

Copyright © 2019 Devada, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

GitOps for Kubernetes

WRITTEN BY ANITA BUEHRLE SENIOR CONTENT LEAD, WEAVERWORKS
AND JOHN VESTER SR. ARCHITECT/PRACTICE LEAD AT CLEANSLATE TECHNOLOGY GROUP

CONTENTS

- > What DevOps Is to the Cloud, GitOps Is to Cloud-Native
- > Key Benefits of GitOps
- > What You Need for GitOps
- > How Kubernetes Handles Configuration Updates
- > Developer Tooling That Drives Ops and Engineering
- > What Does the Typical CI/CD Pipeline Look Like?
- > The GitOps Deployment Pipeline
- > GitOps Is a More Secure Way to Deploy Changes
- > Observability as a Deployment Catalyst
- > A GitOps Workflow
- > Conclusion

What DevOps Is to the Cloud, GitOps Is to Cloud-Native

Companies that want to go fast need to deploy more often and more reliably, with less overhead. GitOps, a concept coined by Weaveworks, is a fast and secure method for developers to maintain and update complex applications running in Kubernetes.

Since Kubernetes and many other cloud-native technologies are almost entirely declarative, infrastructure definitions can be kept alongside application code in Git. Keeping your entire system in Git means that your development team uses familiar Git-based workflows and pull requests to apply both application and infrastructure changes to Kubernetes.

With the entire state of your cluster kept under source control, diff tools and synchronization agents can compare what's running in production with what's under source control — and when a divergence is detected between the two, an alert can be sent, effectively creating a feedback and control loop for managing your cluster.

An Operating Model for Building Cloud-Native Applications

At its core, GitOps is defined as:

1. An operating model for Kubernetes and other cloud-native technologies, providing a set of best practices

that unify deployment, management, and monitoring for containerized clusters and applications.

2. A path toward a developer experience for managing applications where end-to-end CI/CD pipelines and Git workflows are applied to both operations and development.

CONTINUOUS DELIVERY WITH FREEDOM OF CHOICE

GitOps gives you the freedom to choose the best tools for the different parts of your CI/CD pipelines. You can select tools from the open-source ecosystem or from closed-source. Depending on your needs, you may even combine them.

Automate
Kubernetes
with GitOps.

[Download Whitepaper](#)

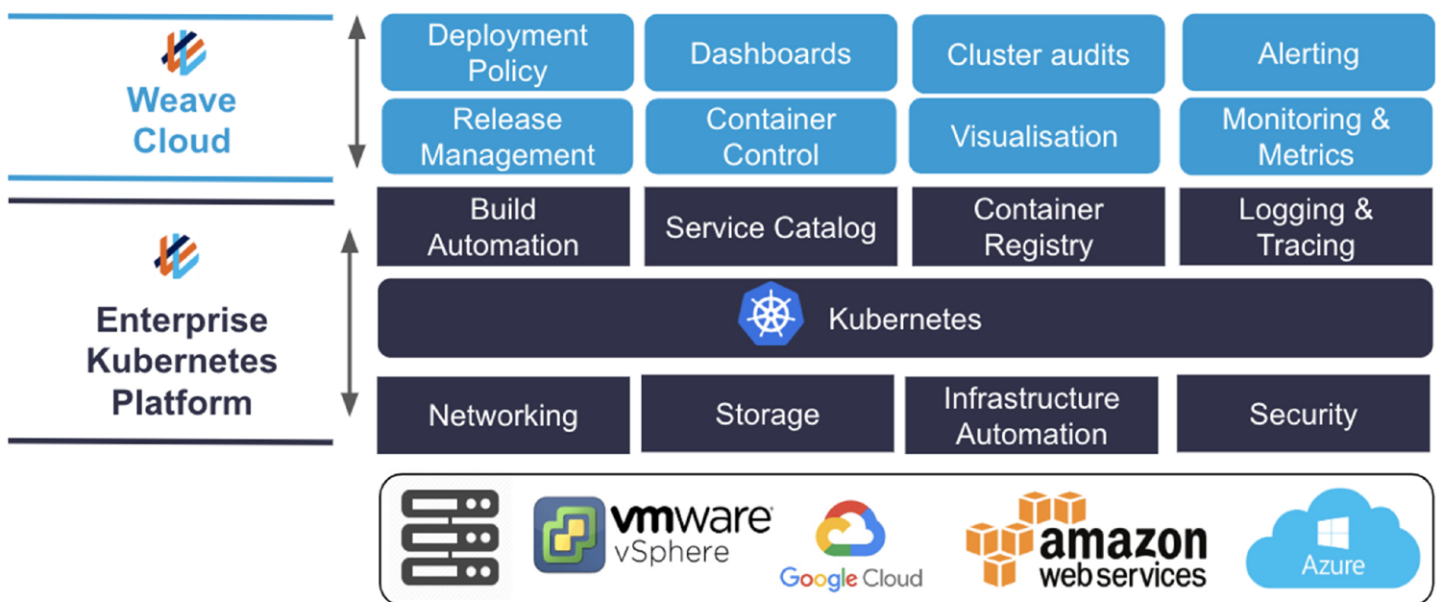
 weaveworks





Production Grade Kubernetes for Your Enterprise

Cluster configuration and management with GitOps



Scalable

Our component based approach covers storage, and networking through to monitoring and management. Run scalable infrastructure with complete confidence.

Open Source

Benefit from the latest features by using the open source upstream Kubernetes - supported by our experts. No lock-in ever.

Any cloud

Run Kubernetes wherever you need it, on VM's, bare metal or on a public cloud. Reduce complexity by using the same declarative approach across environments.

[Learn More](#)



Whatever tools you choose for your deployment and delivery pipelines, applying GitOps best practices should be an integral component of your continuous delivery process. Doing so will make building and adopting a continuous delivery culture into your organization easier.

Key Benefits of GitOps

GitOps best practices are far-reaching and can provide the following benefits.

STRONGER SECURITY GUARANTEES

Git's strong correctness and security guarantees — backed by the strong cryptography used to track and manage changes, as well as the ability to sign changes to prove authorship and origin — are key to a correct and secure definition of the cluster's desired state. If a security breach does occur, the immutable and auditable source of truth can be used to recreate a new system independent of the compromised one, reducing downtime and allowing for a much better incident response.

Also, the separation of responsibility between packaging software and releasing it to a production environment embodies the security principle of least privilege, reducing the impact of compromise and providing a smaller attack surface.

INCREASED SPEED AND PRODUCTIVITY

Continuous deployment automation with an integrated feedback and control loop speeds up your mean time to deployment (MTTD). Declarative definitions kept in Git allow developers to use familiar workflows, reducing the time it takes to spin up a new development or test environment, or to deploy new features to a cluster. Your teams can ship more changes per day, and this translates into a faster turnaround for new features and functionality to the customer.

REDUCED MEAN TIME TO RECOVERY

The amount of time it takes to recover from a cluster meltdown is also decreased with GitOps best practices. With Git's built-in capability to revert/rollback and fork, you gain stable and reproducible rollbacks. Since your entire system is described in Git, you have a single source of truth from which to recover after a cluster failure, reducing your mean time to recovery (MTTR) from hours to minutes.

IMPROVED STABILITY AND RELIABILITY

Because GitOps provides a single operating model for updating infrastructure and apps, you have consistent end-to-end workflows across your entire organization. Not only are your CI/CD pipelines all driven by pull request, your operations tasks are also fully reproducible through Git.

EASIER COMPLIANCE AND AUDITING

With Git's capability to manage your Kubernetes cluster, you gain an audit trail of who did what and when to all cluster changes outside of Kubernetes. This can be used to meet SOC 2 compliance.

Since changes are tracked and logged in a secure manner, compliance and auditing become easier. The use of software tools like `kubediff`, `terradiff`, and `ansiblediff` also allow you to compare a trusted definition of the state of the cluster with the actual running cluster, ensuring that the tracked and auditable changes match reality.

What You Need for GitOps

For those who want to implement GitOps workflows to their CI/CD pipelines, the following criteria need to be in place.

AN ENTIRE SYSTEM DECLARATIVELY DESCRIBED

Kubernetes is one of many modern cloud-native tools out there that are declarative and can be treated as code. Declarative means that configuration is guaranteed by a set of facts instead of a set of instructions. With your application's declarations versioned in Git, you have a single source of truth. Your apps can then be easily deployed and rolled back to and from Kubernetes. Even more importantly — and critical to the GitOps story — is that when disaster strikes, your cluster can also be dependably and quickly reproduced.

THE DESIRED SYSTEM STATE CANONICALLY VERSIONED IN GIT

With the declaration of your system stored in a version control system and serving as your canonical source of truth, you have a single place from which everything is derived and driven. This trivializes rollbacks, and you can use a `git revert` to go back to a previous state. With Git's excellent security guarantees, an SSH key signs commits to enforce strong security guarantees about the authorship and the code's provenance.

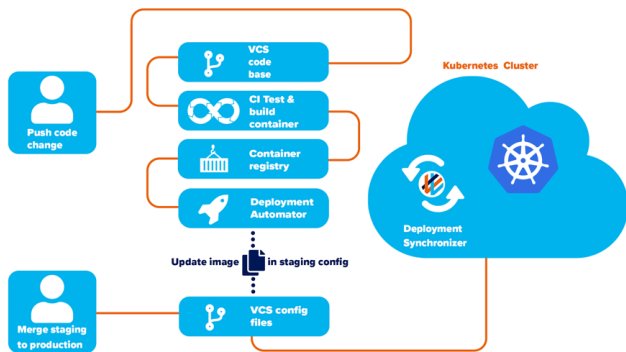
THE ABILITY TO AUTOMATICALLY APPLY APPROVED CHANGES

Once you have the declared state kept in Git, the next step is to have the ability to automatically apply any state changes to your system. What's significant about this is that you don't need specific cluster credentials to make a change to your system. With GitOps, there is a segregated environment, and the state definition lives outside of it. This allows your team to separate what they actually do from how they are going to do it.

SOFTWARE AGENTS TO ENSURE CORRECTNESS

With the state of your entire system kept under version control, you can now employ software agents to inform you whenever reality doesn't match your expectations. The use of `diff` and `sync`

tools also ensures that your entire system is really self-healing. And by self-healing, we don't mean when nodes or pods fail — those are handled by Kubernetes — but in a broader sense, like in the case of human error. In this case, software agents act as the feedback and control loop for your operations.



Deployments to Kubernetes with GitOps

How Kubernetes Handles Configuration Updates

The way Kubernetes handles deployments lends itself very well to GitOps workflows. For example, when a group of configuration updates are made by a human operator, the Kubernetes orchestrator will keep applying those changes until the cluster's state is converged to the updated configuration made by the human. The same is true for any type of Kubernetes resource.

Kubernetes deployments have the following properties that make it perfect for GitOps style deployment workflows:

- **Automation:** Kubernetes provides a built-in mechanism for automating the deployments. A Kubernetes cluster applies a set of changes in a correct order and in a timely manner.
- **Convergence:** Kubernetes will keep trying to make the update until it eventually succeeds.
- **Idempotence:** Multiple and simultaneous convergence instances will all have the same outcome.
- **Determinism:** Assuming that the cluster has adequate resources available, the updated cluster state will depend only on the desired state.

Kubernetes deployments can also be extended and automated using Kubernetes Custom Resource Definitions (CRDs) with the operator pattern. These agents can then be used to automatically detect and apply configuration changes from outside of the system when you need them, essentially creating feedback and control loops.

Developer Tooling That Drives Ops and Engineering

Introducing GitOps into your organization means that:

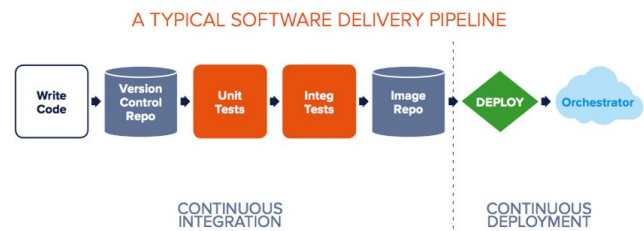
- Any developer who uses Git can start deploying new features to Kubernetes.
- The same workflows are maintained across development and operations.
- All changes can be triggered, stored, validated, and audited in Git.
- Ops changes can be made by pull request, including rollbacks.
- Ops changes can be observed and monitored.

Having everything in one place means that your operations team can use the same workflow to make infrastructure changes by creating issues and reviewing pull requests. GitOps allows you to roll back any kind of change made in your cluster. In addition to this, built-in observability enables your teams to have more autonomy — and to make more changes and experiment — without worrying about breaking the build.

What Does the Typical CI/CD Pipeline Look Like?

Most organizations that set off on their journey to continuous delivery start by automating a CI/CD pipeline. With this newly minted pipeline in place, the keen development team will furiously start writing and pushing code to Git.

In this simplified example, let's say there is a single microservice repository that bundles the microservice's code with its deployment YAML manifest files. YAML files, if you recall, are what define or declare how the microservice runs in the cluster. When the developer pushes the code to Git, a continuous integration tool kicks off unit tests that eventually build the Docker container image that gets pushed to the container registry.



With this typical CI/CD push-based pipeline, Docker container images are then deployed to the actual cluster using some sort of bespoke bash scripts or through some other method that talks directly to the Kubernetes' API.

SECURITY AND THE TYPICAL CI/CD PIPELINE

The typical CI/CD pipeline has security problems.

With this approach, your CI tooling pushes and deploys images to the cluster. For the CI system to apply the changes to a cluster,

you have to share your API credentials with the CI tooling. That means your CI tool becomes a high-value target. If someone breaks into your CI tool, they will have total control over your production cluster, even if your production cluster is highly secure.

WHAT HAPPENS WHEN YOUR CLUSTER GOES DOWN?

Additionally, what happens when you need to recreate your cluster in the case of a total meltdown? How do you restore the previous state of your application? You would have to run all of your CI jobs to rebuild everything, and then re-apply all of the workloads to the new cluster. The typical CI pipeline doesn't have its state easily recorded like when you're using GitOps.

Let's see how you can improve the typical CI/CD pipeline with GitOps.

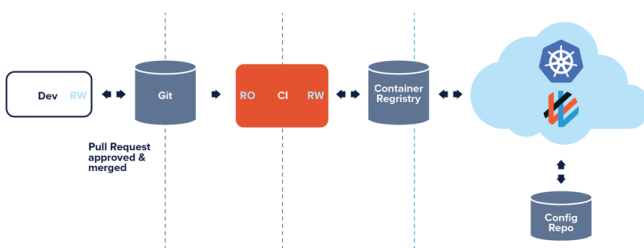
The GitOps Deployment Pipeline

GitOps implements a Kubernetes controller that listens for and synchronizes deployments to your Kubernetes cluster. The controller uses the operator pattern. This is significant on two levels:

1. It is more secure.
2. It automates complex error-prone tasks like having to manually update YAML manifests.

With the operator pattern, an agent acts on behalf of the cluster. It listens to events relating to custom resource changes, then applies those changes based on a deployment policy. The agent is responsible for synchronizing what's in *Git* with what's running *in the cluster*, providing a simple way for your team to achieve continuous deployment.

GitOps Deployment Workflow



GitOps Is a More Secure Way to Deploy Changes

The image is pulled using read-only access to the container registry. The CI tool is not granted cluster privileges, and therefore is not introducing significant security risks to your pipeline.

GITOPS SEPARATION OF PRIVILEGES

GitOps separates CI from CD. This is one reason it is a more secure method of deploying applications to Kubernetes. The table below shows how GitOps separates read/write privileges among the cluster, your CI and CD tooling, and the container repository, providing your team with a secure method of creating updates.

CI TOOLING: TEST, BUILD, SCAN, PUBLISH	CD TOOLING: RECONCILIATION BETWEEN GIT AND THE CLUSTER
Runs outside the production cluster	Runs inside the production cluster
Read access to the code repository	Read/write access to configuration repo
Read/write access to container repository	Read access to image repo
Read/write access to the continuous integration environment	Read/write access to the production cluster

GitOps separation of privileges

Observability as a Deployment Catalyst

An essential component to GitOps is feedback and control. But what is meant exactly by this? In order to have control so that developers can go faster, they need observability built into their deployment workflows. Built-in observability allows engineers to make informed decisions on real-time data. For example, when a deployment is being rolled out, a final health check can be made against your running cluster before committing to that update. Or an update that didn't go as planned can be easily rolled back to a good state.

With a feedback control loop, you can effectively answer the following questions:

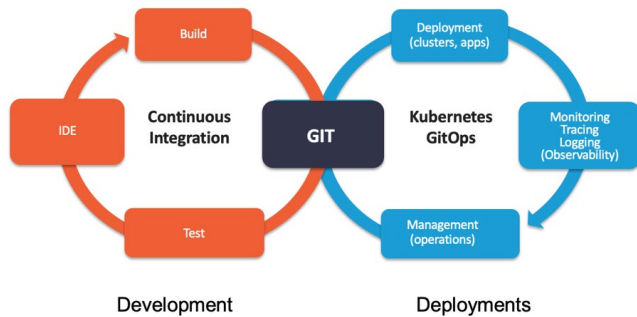
- How do I know if my deployment will succeed?
- How do I know if the live system has converged to the desired state?
- Can I be notified when this differs?
- Can I trigger a convergence between the cluster and source control?

While Git is the source of truth for the desired state of the system, observability provides a benchmark for the actual production state of the running system. GitOps takes advantage of both to manage applications.

With GitOps, divergence and convergence are achieved with a set of "diff" and "sync" tools (e.g. kubediff, terradiff, ansiblediff) that compare the intended state with actual state. Diff tools are also used to alert developers when the deployment is out of sync.

A feedback loop with diffs and built-in observability looks something like this, where observability provides feedback

for developers and operators to make key decisions about deployments and the system:



Feedback and control loop

Because about-to-be-released services or updates can be observed in real-time within the running cluster, you can deploy with confidence and deliver higher-quality features.

Observability is a principal driver of the continuous delivery cycle for Kubernetes since it describes the actual running state of the system at any given time.

After new features and fixes are merged to Git, the deployment pipeline is triggered, and once the image is ready to be released, it can be observed in real-time against the running cluster. At this point, the developer may return to the beginning of the pipeline based on this feedback, or they may deploy and release the image to the production cluster.

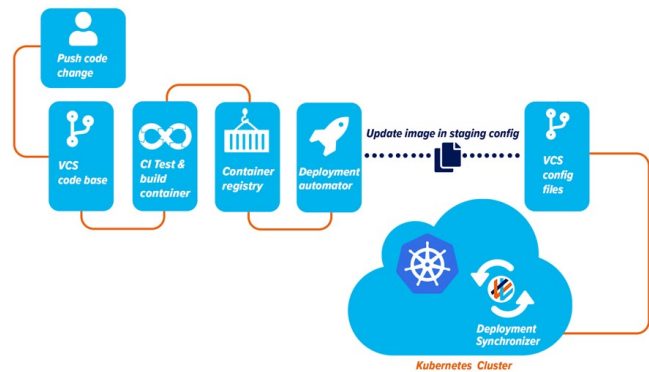
A GitOps Workflow

The GitOps core machinery is in its CI/CD tooling, with the critical piece being continuous deployment (CD) that supports Git-cluster synchronization. It is designed specifically for version-controlled systems and declarative application stacks. Every developer on your team is familiar with Git and can make pull requests. Now, they can use Git to accelerate and simplify application deployments to Kubernetes, as well.

Here is a typical developer workflow for creating or updating a new feature:

1. A pull request for a new feature is pushed to GitHub for review.
2. The code is reviewed and approved by a colleague. After the code is revised, and re-approved it is merged to Git.
3. The Git merge triggers the CI and build pipeline, runs a series of tests, and then eventually builds a new image and deposits to the new image to a registry.
4. The Deployment Automator watches the image registry, notices the image, pulls the new image from the registry, and updates its YAML in the config repo.

5. The Deployment Synchronizer detects that the cluster is out of date, pulls the changed manifests from the config repo, and deploys the new feature to production.



GitOps workflow to Kubernetes

Conclusion

GitOps is a way to do continuous delivery that implements Git as a single source of truth for declarative infrastructure and applications. As an operating model for Kubernetes and other cloud-native technologies, GitOps provides a set of best practices that unify deployment, management, and monitoring for containerized clusters and applications. With Git as the central hub of your automated CI/CD pipelines, developers can make pull requests to accelerate and simplify application and infrastructure deployments to Kubernetes.

FASTER DEVELOPMENT

Adopting GitOps best practices means that developers use familiar tools like Git to manage updates and features to Kubernetes, resulting in faster development and product turnaround times. By continuously pushing feature updates, businesses are more agile, can respond more quickly to customer demands, and are more innovative.

BETTER OPS

With GitOps, you have a complete end-to-end pipeline. Not only are your automated CI/CD pipelines all driven by pull request, your operations tasks are also fully reproducible through Git.

STRONGER SECURITY GUARANTEES

Git's strong correctness and security guarantees — backed by the strong cryptography used to track and manage changes, as well as the ability to sign changes to prove authorship and origin — are key to a correct and secure definition of desired state of the cluster. If a security breach does occur, the immutable and auditable source of truth can be used to recreate a new system independently of the compromised one, reducing downtime and allowing much better incident response.

Separation of responsibility between packaging software and releasing it to a production environment also embodies the security principle of least privilege, reducing the impact of compromise and providing a smaller attack surface.

EASIER COMPLIANCE AND AUDITING

Since changes are tracked and logged in a secure manner, compliance and auditing are made trivial. The use of comparison tools like kubediff, terradiff, and ansiblediff also allow you to compare a trusted definition of the state of the cluster with the actual running cluster, ensuring that the tracked and auditable changes match reality.



Written by **Anita Buehrle**, *Senior Content Lead, Weaveworks*

Anita has over 20 years of experience in software development. She began as a technical writer at the X Windows server company, Hummingbird (now OpenText), and later at Algorithmics, Inc., where she managed the technical documentation for their flagship product, RiskWatch, a product used by the world's leading investment banks. She's also worked several years in the wireless industry, leading QA and Product Delivery teams in testing and shipping a graphics engine and a J2EE-based content server that was acquired by Research in Motion and whose technology was used to power the first generation of color-screen graphical user interfaces for Blackberry devices. She's also developed and marketed her own mobile apps. Currently, she leads content and other market-driven initiatives at Weaveworks.



Written by **John Vester**, *Sr. Architect/Practice Lead at CleanSlate Technology Group*

John is an Information Technology professional with 25+ years expertise in application development, project management, enterprise integration, and team management. He is currently focusing on enterprise architecture/application design/continuous delivery utilizing object-oriented programming languages and frameworks. He has prior expertise with building Java-based APIs against React and Angular client frameworks, integration architecture and design, and CRM design and customization. He also has additional experience using both C# (.NET framework) and J2EE (Spring, plus various other frameworks).



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399 919.678.0300

Copyright © 2019 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.