What is Cloud Native?

Cloud native is the software approach of building, deploying, and managing modern applications in cloud computing environments. Modern companies want to build highly scalable, flexible, and resilient applications that they can update quickly to meet customer demands. To do so, they use modern tools and techniques that inherently support application development on cloud infrastructure. These cloud-native technologies support fast and frequent changes to applications without impacting service delivery, providing adopters with an innovative, competitive advantage.

How does a cloud-native approach benefit businesses?

Organizations gain competitive advantages in various ways when they build cloud-native software applications.

Increase efficiency

Cloud-native development brings along agile practices like DevOps and continuous delivery (CD). Developers use automated tools, cloud services, and modern design culture to build scalable applications rapidly.

Reduce cost

By adopting the cloud-native approach, companies don't have to invest in the procurement and maintenance of costly physical infrastructure. This results in long-term savings in operational expenditure. The cost savings of building cloud-native solutions might also benefit your clients.

Ensure availability

Cloud-native technology allows companies to build resilient and highly available applications. Feature updates do not cause downtime and companies can scale up app resources during peak seasons to provide a positive customer experience.

What are cloud-native applications?

Cloud-native applications are software programs that consist of multiple small, interdependent services called microservices. Traditionally, developers built monolithic applications with a single block structure containing all the required functionalities. By using the cloud-native approach, software developers break the functionalities into smaller microservices. This makes cloud-native applications more agile as these microservices work independently and take minimal computing resources to run.

Cloud-native applications compared to traditional enterprise applications

Traditional enterprise applications were built using less flexible software development methods. Developers typically worked on a large batch of software functionalities before releasing them for testing. As such, traditional enterprise applications took longer to deploy and were not scalable.

On the other hand, cloud-native applications use a collaborative approach and are highly scalable on different platforms. Developers use software tools to heavily automate building, testing, and deploying procedures in cloud-native applications. You can set up, deploy, or duplicate microservices in an instant, an action that's not possible with traditional applications.

What is the CNCF?

The Cloud Native Computing Foundation (CNCF) is an open-source foundation that helps organizations kick start their cloud-native journey. Established in 2015, the CNCF supports the open-source community in developing critical cloud-native components, including Kubernetes. Amazon is a member of CNCF.

What is cloud-native application architecture?

The cloud-native architecture combines software components that development teams use to build and run scalable cloud-native applications. The CNCF lists immutable infrastructure, microservices, declarative APIs, containers, and service meshes as the technological blocks of cloud-native architecture.

Immutable infrastructure

Immutable infrastructure means that the servers for hosting cloud-native applications remain unchanged after deployment. If the application requires more computing resources, the old server is discarded, and the app is moved to a new high-performance server. By avoiding manual upgrades, immutable infrastructure makes cloud-native deployment a predictable process.

Microservices

Microservices are small, independent software components that collectively perform as complete cloud-native software. Each microservice focuses on a small, specific problem. Microservices are loosely coupled, which means that they are independent software components that communicate with each other. Developers make changes to the application by working on individual microservices. That way, the application continues to function even if one microservice fails.

API

Application Programming Interface (API) is a method that two or more software programs use to exchange information. Cloud-native systems use APIs to bring the loosely coupled microservices together. API tells you what data the microservice wants and what results it can give you, instead of specifying the steps to achieve the outcome.

Service mesh

Service mesh is a software layer in the cloud infrastructure that manages the communication between multiple microservices. Developers use the service mesh to introduce additional functions without writing new code in the application.

Containers

Containers are the smallest compute unit in a cloud-native application. They are software components that pack the microservice code and other required files in cloud-native systems. By containerizing the microservices, cloud-native applications run independently of the underlying operating system and hardware. This means that software developers can deploy cloud-native applications on premises, on cloud infrastructure, or on hybrid clouds. Developers use containers for packaging the microservices with their respective dependencies, such as the resource files, libraries, and scripts that the main application requires to run.

Benefits of containers

Some benefits of containers include:

- You use fewer computing resources than conventional application deployment
- You can deploy them almost instantly
- You can scale the cloud computing resources your application requires more efficiently

What is cloud-native application development?

Cloud-native application development describes how and where developers build and deploy cloud-native applications. A cultural shift is important for cloud-native development. Developers adopt specific software practices to decrease the software delivery timeline and deliver accurate features that meet changing user expectations. We give some common cloud-native development practices below.

Continuous integration

Continuous integration (CI) is a software practice in which developers integrate changes into a shared code base frequently and without errors. Small, frequent changes make development more efficient because you can identify and troubleshoot issues faster. CI tools automatically assess the code quality for every change so that development teams can add new features with greater confidence.

Continuous delivery

Continuous delivery (CD) is a software practice that supports cloud-native development. With CD, development teams ensure that the microservices are always ready to be deployed to the cloud. They use software automation tools to reduce risk when making changes, such as introducing new features and fixing bugs on applications. CI and CD work together for efficient software delivery.

DevOps

DevOps is a software culture that improves the collaboration of development and operations teams. It is a design philosophy that aligns with the cloud-native model. DevOps practices allow organizations to speed up the software development lifecycle. Developers and operation engineers use DevOps tools to automate cloud-native development.

Serverless

Serverless computing is a cloud-native model where the cloud provider fully manages the underlying server infrastructure. Developers use serverless computing because the cloud infrastructure automatically scales and configures to meet application requirements. Developers only pay for the resources the application uses. The serverless architecture automatically removes compute resources when the app stops running.

What are the benefits of cloud-native application development?

Faster development

Developers use the cloud-native approach to reduce development time and achieve better quality applications. Instead of relying on specific hardware infrastructure, developers build ready-to-deploy containerized applications with DevOps practices. This allows developers to respond to changes quickly. For example, they can make several daily updates without shutting down the app.

Platform independence

By building and deploying applications in the cloud, developers are assured of the consistency and reliability of the operating environment. They don't have to worry about hardware incompatibility because the cloud provider takes care of it. Therefore, developers can focus on delivering values in the app instead of setting up the underlying infrastructure.

Cost-effective operations

You only pay for the resources your application actually uses. For example, if your user traffic spikes only during certain times of the year, you pay additional charges only for that time period. You do not have to provision extra resources that sit idle for most of the year.

What is cloud-native stack?

Cloud-native stack describes the layers of cloud-native technologies that developers use to build, manage, and run cloud-native applications. They are categorized as follows.

Infrastructure layer

The infrastructure layer is the foundation of the cloud-native stack. It consists of operating systems, storage, network, and other computing resources managed by third-party cloud providers.

Provisioning layer

The provisioning layer consists of cloud services that allocate and configure the cloud environment.

Runtime layer

The runtime layer provides cloud-native technologies for containers to function. This comprises cloud data storage, networking capability, and a container runtime such as containerd.

Orchestration and management layer

Orchestration and management are responsible for integrating the various cloud components so that they function as a single unit. It is similar to how an operating system works in traditional computing. Developers use orchestration tools like Kubernetes to deploy, manage, and scale cloud applications on different machines.

Application definition and development layer

This cloud-native stack layer consists of software technologies for building cloud-native applications. For example, developers use cloud technologies like database, messaging, container images, and continuous integration (CI) and continuous delivery (CD) tools to build cloud applications.

Observability and analysis tools

Observability and analysis tools monitor, evaluate, and improve the system health of cloud applications. Developers use tools to monitor metrics like CPU usage, memory, and latency to ensure there is no disruption to the app's service quality.

What is cloud computing?

Cloud computing refers to software infrastructure hosted on an external data center and made available to users on a pay-per-use basis. Companies don't have to pay for expensive servers and maintain them. Instead, they can use on-demand cloud-native services such as storage, database, and analytics from a cloud provider.

Cloud computing compared to cloud native

Cloud computing is the resources, infrastructure, and tools provided on-demand by cloud vendors. Meanwhile, cloud native is an approach that builds and runs software programs with the cloud computing model.

What is cloud-enabled?

Cloud-enabled applications are legacy enterprise applications that were running on an on-premises data center but have been modified to run on the cloud. This involves changing part of the software module to migrate the application to cloud servers. You can thus use the application from a browser while retaining its original features.

Cloud native compared to cloud enabled

The term cloud native refers to an application that was designed to reside in the cloud from the start. Cloud native involves cloud technologies like microservices, container orchestrators, and auto scaling. A cloud-enabled application doesn't have the flexibility, resiliency, or scalability of its cloud-native counterpart. This is because cloud-enabled applications retain their monolithic structure even though they have moved to the cloud.

he impact of cloud computing on industry and end users would be difficult to overstate: many aspects of everyday life have been transformed by the omnipresence of software that runs on cloud networks. By leveraging cloud computing, startups and businesses are able to optimize costs and increase their offerings without purchasing and managing the hardware and software themselves. Independent developers are empowered to launch globally-available apps and online services. Researchers can share and analyze data at scales once reserved only for highly-funded projects. And internet users can quickly access software and storage to create, share, and store digital media in quantities that extend far beyond the computing capacity of their personal devices.

Despite the growing presence of cloud computing, its details remain obscure to many. What exactly is the cloud, how does one use it, and what are its benefits for businesses, developers, researchers, government, healthcare practitioners, and students? In this conceptual article, we’ll provide a general overview of cloud computing, its history, delivery models, offerings, and risks.

In this article, you will gain an understanding of how the cloud can help support business, research, education, and community infrastructure and how to get started using the cloud for your own projects.

What is Cloud Computing?

Cloud computing is the delivery of computing resources *as a service*, meaning that the resources are owned and managed by the cloud provider rather than the end user. Those resources may include anything from browser-based software applications (such as Tik Tok or Netflix), third party data storage for photos and other digital media (such as iCloud or Dropbox), or third-party servers used to support the computing infrastructure of a business, research, or personal project.

Before the broad proliferation of cloud computing, businesses and general computer users typically had to buy and maintain the software and hardware that they wished to use. With the growing availability of cloud-based applications, storage, services, and machines, businesses and consumers now have access to a wealth of on-demand computing resources as internet-accessed services. Shifting from on-premise software and hardware to networked remote and distributed resources means cloud users no longer have to invest the labor, capital, or expertise required for buying and maintaining these computing resources themselves. This unprecedented access to computing resources has given rise to a new wave of cloud-based businesses, changed IT practices across industries, and transformed many everyday computer-assisted practices. With the cloud, individuals can now work with colleagues over video meetings and other collaborative platforms, access entertainment and educational content on demand, communicate with household appliances, hail a cab with a mobile device, and rent a vacation room in someone's house.

Defining Cloud Computing

The National Institute of Standards and Technology (NIST), a non-regulatory agency of the United States Department of Commerce with a mission to advance innovation, defines cloud computing as:

> a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned

and released with minimal management effort or service provider interaction.

NIST lists the following as the five essential characteristics of cloud computing:

- On-demand self-service: Cloud resources can be accessed or provisioned without human interaction. With this model, consumers can gain immediate access to cloud services upon signup. Organizations can also create mechanisms for allowing employees, customers, or partners to access internal cloud services on demand according to predetermined logics without needing to go through IT services.
- Broad network access: Users can access cloud services and resources through any device and in any networked location provided that they have permission.
- Resource pooling: Cloud provider resources are shared by multiple tenants while keeping the data of individual clients hidden from other clients.
- Rapid elasticity: Unlike on-premise hardware and software, cloud computing resources can be rapidly increased, decreased, or otherwise modified based on the cloud user's changing needs.
- Measured service: Usage of cloud resources is metered so that businesses and other cloud users need only pay for the resources they use in any given billing cycle.

These characteristics offer a wide variety of transformative opportunities for businesses and individuals alike, which we'll discuss later in the section Benefits of Cloud Computing. To gain some additional context, let's briefly review the emergence of cloud computing.

History of Cloud Computing

Many aspects of cloud computing can be traced as far back as the 1950s, when universities and companies rented out computation time on mainframe computers. At the time, renting was one of the only ways to access computing resources as computing technology was too large and expensive to be owned or managed by individuals. By the 1960s, computer scientists like John McCarthy of Stanford University and J.C.R Licklider of The U.S. Department of Defense Advanced Research Projects Agency (ARPA) began proposing ideas that anticipated some of the major features of cloud computing today, such as the conceptualization of computing as a public utility and the possibility of a network of computers that would allow people to access data and programs from anywhere in the world.

Cloud computing, however, didn't become a mainstream reality and a popular term until the first decade of the 21st century. This decade saw the launch of cloud services like Amazon's Elastic Compute (EC2) and Simple Storage Service (S3) in 2006, Heroku in 2007, Google Cloud Platform in 2008, Alibaba Cloud in 2009, Windows Azure (now Microsoft Azure) in 2010, IBM's SmartCloud in 2011, and DigitalOcean in 2011. These services allowed existing businesses to optimize costs by migrating their in-house IT infrastructure to cloud-based resources and provided independent developers and small developer teams resources for creating and deploying apps. Cloud-based applications, known as *Software as a Service (SaaS)* — which we'll discuss in greater detail in the Cloud Delivery Models section — also became popular during this time period. Unlike on-premise software, or software that users need to physically install and maintain on their machines, SaaS increased the availability of applications by allowing users to access them from a variety of devices on demand.

Some of these cloud-based applications — such as Google's productivity apps (Gmail, Drive, and Docs) and Microsoft 365 (a cloud-based version of the Microsoft Office Suite) — were offered by the same companies that launched cloud infrastructure services, while other pre-existing software products, such as Adobe Creative Cloud, were launched as cloud-based applications using the services of cloud providers. New SaaS products and businesses also emerged based on the novel opportunities of these cloud providers, such as Netflix's streaming services in 2007, the music platform Spotify in 2008, the file-hosting service Dropbox in 2009, the video conferencing service Zoom in 2012, and the communication tool Slack in 2013. Today, cloud-based IT infrastructure and cloud-based applications have become a popular choice for both businesses and individual users and their market share is expected to grow.

Cloud Delivery Models

Cloud resources are provided in a variety of different delivery models that offer customers different levels of support and flexibility.

Infrastructure as a Service (IaaS)

IaaS is the on-demand delivery of computing infrastructure, including operating systems, networking, storage, and other infrastructural components. Acting much like a virtual equivalent to physical servers, IaaS relieves cloud users of the need to buy and maintain physical servers while also providing the flexibility to scale and pay for resources as needed. IaaS is a popular option for businesses that wish to leverage the advantages of the cloud and have system administrators who can oversee the installation, configuration, and management of operating systems, development tools, and other underlying infrastructure that they wish to use. However, IaaS is also used by developers, researchers, and others who wish to customize the underlying infrastructure of their computing environment. Given its flexibility, IaaS can support everything from a company's computing infrastructure to web hosting to big data analysis.

Platform as a Service (PaaS)

PaaS provides a computing platform where the underlying infrastructure (such as the operating system and other software) is installed, configured, and maintained by the provider, allowing users to focus their efforts on developing and deploying apps in a tested and standardized environment. PaaS is commonly used by software developers and developer teams as it cuts down on the complexity of setting up and maintaining computer infrastructure, while also supporting collaboration among distributed teams. PaaS can be a good choice for developers who don't have the need to customize their underlying infrastructure, or those who want to focus their attention on development rather than DevOps and system administration.

Software as a Service (SaaS)

SaaS providers are cloud-based applications that users access on demand from the internet without needing to install or maintain the software. Examples include GitHub, Google Docs, Slack, and Adobe Creative Cloud. SaaS applications are popular among businesses and general users given that they're often easy to adopt, accessible from any device, and have free, premium, and enterprise versions of their applications. Like PaaS, SaaS abstracts away the underlying infrastructure of the software application so that users are only exposed to the interface they interact with.

Cloud Environments

Cloud services are available as public or private resources, each of which serves different needs.

Public Cloud

The *public cloud* refers to cloud services (such as virtual machines, storage, or applications) offered publicly by a commercial provider to businesses and individuals. Public cloud resources are hosted on the commercial provider's hardware, which users access through the internet. They are not always suitable for organizations in highly-regulated industries, such as healthcare or finance, as public cloud environments may not comply with industry regulations regarding customer data.

Private Cloud

The *private cloud* refers to cloud services that are owned and managed by the organization that uses them and available only to the organization's employees and customers. Private clouds allow organizations to exert greater control over their computing environment and their stored data, which can be necessary for organizations in highly-regulated industries. Private clouds are sometimes seen as more secure than public clouds as they are accessed through private networks and enable the organization to directly oversee their cloud security. Public cloud providers sometimes provide their services as applications that can be installed on private clouds, allowing organizations to keep their infrastructure and data on premise while taking advantage of the public cloud's latest innovations.

Hybrid Cloud and Multicloud

Many organizations use a *hybrid cloud* environment which combines public and private cloud resources to support the organization's computing needs while maintaining compliance with industry regulation. *Multicloud* environments are also common, which entail the use of more than one public cloud provider (for example, combining Amazon Web Services and DigitalOcean).

Benefits of Cloud Computing

Cloud computing offers a variety of benefits to individuals, businesses, developers, and other organizations. These benefits vary according to the cloud users goals and activities.

For Business and Industry

Prior to the proliferation of cloud computing, most businesses and organizations needed to purchase and maintain the software and hardware that supported their computing activities. As cloud computing resources became available, many businesses began using them to store data, provide enterprise software, and deploy online products and services. Some of these cloud-based adoptions and innovations are industry-specific. In healthcare, many providers use cloud services that are specifically designed to store and share patient data or communicate with patients. In academia, educators and researchers use cloud-based teaching and research apps. But there are also a large number of general cloud-based tools that have been adopted across industries, such as apps for productivity, messaging, expense management, video conferencing, project management, newsletters, surveys, customer relations management, identity management, and scheduling. The rapid growth of cloud-based business apps and infrastructure shows that the cloud isn't just changing business IT strategy: it's a booming business in its own right.

Cloud-based technologies offer businesses several key advantages. First, they can help optimize IT costs. As businesses shift towards renting computing resources, they no longer have to invest as much in purchasing and maintaining on-premise IT infrastructure. Cloud computing is also enormously flexible, allowing businesses to rapidly scale (and only pay for) the computing resources they actually use. Cost, however, is not the only consideration that drives cloud adoption in business. Cloud-based technologies can help make internal IT processes more efficient as they can be accessed on demand by employees without needing to go through IT approval processes. Cloud-based apps can improve collaboration across a business as they allow for real-time communication and data sharing.

For Independent Developers

Computing resources that were once only affordable to large companies and organizations are now available on demand through an internet connection and at a fraction of their previous cost. In effect, independent developers can rapidly deploy and experiment with cloud-based apps. Cloud-based apps for sharing code (such as GitHub) have also made it easier for developers to build upon and collaborate on open source software projects. Additionally, cloud-based educational platforms and interactive coding tutorials have expanded access to developer education, enabling individuals without formal technical training to learn to code in their own time.

Altogether, these cloud-based computing and educational resources have helped lower the barriers to learning developer skills and deploying cloud-based apps. Formal training, company support, and massive amounts of startup capital are no longer necessary for individuals to experiment with creating and deploying apps, allowing for more individuals to participate in cloud development, compete with established industry players, and create and share apps as side projects.

For Researchers

As machine learning methods become increasingly important in scientific research, cloud computing has become essential to many scientific fields, including astronomy, physics, genomics, and artificial intelligence. The massive amount of data collected and analyzed in machine learning and other data-intensive research projects often require computing resources that scale beyond the capacity of hardware owned by an individual researcher or provisioned by the university. Cloud computing allows researchers to access (and only pay for) computing resources as their workloads require and allows for real-time collaboration with research partners across the globe. Without commercial cloud providers, a majority of academic machine learning research would be limited to individuals with access to university-provisioned, high-powered computing resources.

For Educators and Students

Cloud computing has also provided students with tools for supplementing their education and opportunities to put their technical skills into practice as they learn. Cloud-based apps for sharing, teaching, and collaborating on code and data (such as GitHub and Jupyter Notebooks) enable students to learn technical skills in a hands-on manner by studying, deploying, and contributing to open source software and research projects relevant to their field or professional aspirations. And just like independent developers, students are able to use cloud computing resources to share their code and apps with the public and reap the satisfaction of understanding the real-world application of their skills.

Students, researchers, and educators can also take advantage of cloud computing resources to support personalized academic infrastructure and practice greater control over their computing environments. Some academics prefer this approach as it lets them pick which applications they use, customize the functionality and design of these tools, and limit or prohibit the collection of data. There are also a growing number of cloud-based applications developed specifically for academic purposes that supplement or provide alternatives to traditional academic IT offerings. Voyant Tools offers students and researchers a code-free method for providing textual analysis on documents of their choosing and The HathiTrust provides access to its digital collection of millions of volumes. Reclaim Hosting, Commons in a Box, the Modern Language Humanities Commons, and Manifold offer educational, publishing, and networking tools designed specifically for academic communities.

For Community Infrastructure

Some individuals and communities choose to install and manage their own cloud-based software to serve community needs and values, customize functionality, protect user data, and have more control over their computing environment. Open source software, such as social media tools like Mastodon, video conferencing software like Jitsi, collaborative text editors like Etherpad, and web chat tools like Rocket Chat, provide alternatives to SaaS platforms that often limit user's control, privacy, and oversight over their computing environment. While often requiring more administrative work than SaaS applications or social media platforms, some communities prefer these options given ethical concerns about the use of personal data and company practices with popular platforms and SaaS applications.

Risks, Costs, and Ethics in Cloud Computing

Though the cloud offers many benefits, it also comes with its own set of risks, costs, and ethical questions that should be considered. Some of these issues are relevant to all cloud users, while others are more applicable to businesses and organizations that use the cloud to store customers' data:

Considerations for all cloud users:

- Security: Cloud resources can have additional security vulnerabilities (compared to traditional on-premise data centers) given their use of APIs, cloud-based credentials, and on-demand services that make it easier for attackers to obtain unauthorized access. Find out what measures the cloud service provider takes to secure customer data from theft and other attacks and what practices or additional services customers can implement to safeguard their data.
- Data loss: Just as with physically-owned or managed devices, cloud services can permanently lose stored data due to physical disasters, bugs, unintended syncing, user-generated errors, or other unforeseen issues. When implementing cloud services, find out what backup services the provider offers and be aware that these may not be automatically or freely provided. You may also choose to run backups yourself.
- Data persistence: There are times when cloud users may want to ensure the deletion of personal data they've given to cloud service providers. However, the processes for deleting data on cloud resources and verifying that deletion can be time-consuming, complicated, or even impossible. Before you give cloud

providers access to your data, find out what their policies are for deleting it in case you want to remove the data later.

- Costs: Though the cloud can provide computing services at a fraction of the cost of owning them, expenses for cloud services can quickly ramp up with usage. When signing up for a cloud service, check the billing details to learn how services are metered and whether you can set caps or notifications when usage goes beyond your desired limits. It is also worth researching how billing details are communicated, as the billing methods of some providers are not always easy to understand.
- Vendor lock-in: Users of proprietary cloud services may be at more risk for vendor lock in, or the condition in which it becomes difficult or impossible to change providers once computing operations are structured to fit a closed, proprietary system. Using open source cloud solutions can help alleviate this risk as its open standards make it easier to migrate computing operations from one provider to another. However, cloud users should be aware that any migration will take work, planning, and expertise.
- Company use of data: Cloud service providers may use data to understand customer use of their product, sell or personalize ads, train machine learning algorithms, or even sell customer data to outside entities. If you have concerns about how your or your organization’s data is used, make sure to find out the service provider’s policies regarding their use of it.
- Company ethic: Given the vast power some cloud service providers have over world affairs, cloud users may want to consider the ethics of the company that their business is supporting. Reviewing company practices with regard to topics such as data collection, advertising, hate speech, politics, misinformation, the environment, and labor may help a cloud user choose a provider that best reflects their personal values.
- Loss of user control and visibility: The use of third-party computing resources makes it difficult or impossible for cloud users to have full visibility and control over their computing environments, which can create a variety of technical and trust concerns. Some of these technical concerns can be helped through the use of monitoring and analytics tools which allow cloud users to stay updated on their infrastructure’s performance, allowing users to respond quickly when problems arise. Trust concerns — such as those related to a company’s use of personal data — can be addressed by reviewing the company’s customer data policies and public forms of analysis about its data practices.

Additional Business Considerations:

- Regulation: Some industries — such as healthcare, finance, and education — have strict regulations regarding the storage and use of customer data and may prohibit the storage of customer data in public clouds. Cloud users in these industries often need to adopt a hybrid cloud approach and other customized IT solutions in order to comply with regulations regarding customer data. In addition to industry regulations, organizations also need to comply with data protection and privacy laws of the location where their service is accessed. For example, cloud providers serving customers in the European Union must comply with the General Data Protection Regulation (GDPR).
- Complexity: Migrating an organization's computing resources to the cloud can be an extremely complex endeavor, requiring in-depth planning, governance structures, and continuous oversight to avoid incompatibilities, data loss, and cost optimization. Though the cloud can help organizations cut costs on computing infrastructure, they will still need IT experts to direct and manage infrastructure.

Cloud Computing can be defined as the practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer. Companies offering such kinds of cloud computing services are called *cloud providers* and typically charge for cloud computing services based on usage. Grids and clusters are the foundations for cloud computing.

Types of Cloud Computing

Most cloud computing services fall into five broad categories:

1. Software as a service (SaaS)
2. Platform as a service (PaaS)
3. Infrastructure as a service (IaaS)
4. Anything/Everything as a service (XaaS)
5. Function as a Service (FaaS)

These are sometimes called the **cloud computing stack** because they are built on top of one another. Knowing what they are and how they are different, makes it easier to accomplish your goals. These abstraction layers can also be viewed as a **layered architecture** where services of a higher layer can be composed of services of the underlying layer i.e, SaaS can provide Infrastructure.

Software as a Service(SaaS)

Software-as-a-Service (SaaS) is a way of delivering services and applications over the Internet. Instead of installing and maintaining software, we simply access it via the Internet, freeing ourselves from the complex software and hardware management. It removes the need to install and run applications on our own computers or in the data centers eliminating the expenses of hardware as well as software maintenance.
SaaS provides a complete software solution that you purchase on a **pay-as-you-go** basis from a cloud service provider. Most SaaS applications can be run directly from a web browser without any downloads or installations required. The SaaS applications are sometimes called **Web-based software, on-demand software, or hosted software.**

Advantages of SaaS

1. **Cost-Effective:** Pay only for what you use.
2. **Reduced time:** Users can run most SaaS apps directly from their web browser without needing to download and install any software. This reduces the time spent in installation and configuration and can reduce the issues that can get in the way of the software deployment.
3. **Accessibility:** We can Access app data from anywhere.
4. **Automatic updates:** Rather than purchasing new software, customers rely on a SaaS provider to automatically perform the updates.
5. **Scalability:** It allows the users to access the services and features on-demand.

The various companies providing *Software as a service* are Cloud9 Analytics, Salesforce.com, Cloud Switch, Microsoft Office 365, Big Commerce, Eloqua, dropBox, and Cloud Tran.

Disadvantages of Saas :

1. **Limited customization**: SaaS solutions are typically not as customizable as on-premises software, meaning that users may have to work within the constraints of the SaaS provider's platform and may not be able to tailor the software to their specific needs.
2. **Dependence on internet connectivity**: SaaS solutions are typically cloud-based, which means that they require a stable internet connection to function properly. This can be problematic for users in areas with poor connectivity or for those who need to access the software in offline environments.
3. **Security concerns:** SaaS providers are responsible for maintaining the security of the data stored on their servers, but there is still a risk of data breaches or other security incidents.
4. **Limited control over data:** SaaS providers may have access to a user's data, which can be a concern for organizations that need to maintain strict control over their data for regulatory or other reasons.

Platform as a Service

PaaS is a category of cloud computing that provides a platform and environment to allow developers to build applications and services over the internet. PaaS services are hosted in the cloud and accessed by users simply via their web

browser.
A PaaS provider hosts the hardware and software on its own infrastructure. As a result, PaaS frees users from having to install in-house hardware and software to develop or run a new application. Thus, the development and deployment of the application take place **independent of the hardware**.
The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment. To make it simple, take the example of an annual day function, you will have two options either to create a venue or to rent a venue but the function is the same.

Advantages of PaaS:

1. **Simple and convenient for users:** It provides much of the infrastructure and other IT services, which users can access anywhere via a web browser.
2. **Cost-Effective:** It charges for the services provided on a per-use basis thus eliminating the expenses one may have for on-premises hardware and software.
3. **Efficiently managing the lifecycle:** It is designed to support the complete web application lifecycle: building, testing, deploying, managing, and updating.
4. **Efficiency:** It allows for higher-level programming with reduced complexity thus, the overall development of the application can be more effective.

The various companies providing *Platform as a service* are Amazon Web services Elastic Beanstalk, Salesforce, Windows Azure, Google App Engine, cloud Bees and IBM smart cloud.

Disadvantages of Paas:

1. **Limited control over infrastructure:** PaaS providers typically manage the underlying infrastructure and take care of maintenance and updates, but this can also mean that users have less control over the environment and may not be able to make certain customizations.
2. **Dependence on the provider**: Users are dependent on the PaaS provider for the availability, scalability, and reliability of the platform, which can be a risk if the provider experiences outages or other issues.
3. **Limited flexibility:** PaaS solutions may not be able to accommodate certain types of workloads or applications, which can limit the value of the solution for certain organizations.

Infrastructure as a Service

Infrastructure as a service (IaaS) is a service model that delivers computer infrastructure on an outsourced basis to support various operations. Typically IaaS is a service where infrastructure is provided as outsourcing to enterprises such as networking equipment, devices, database, and web servers.
It is also known as **Hardware as a Service (HaaS).** IaaS customers pay on a per-user basis, typically by the hour, week, or month. Some providers also charge customers based on the amount of virtual machine space they use.
It simply provides the underlying operating systems, security, networking, and servers for developing such applications, and services, and deploying development tools, databases, etc.

Advantages of IaaS:

1. **Cost-Effective:** Eliminates capital expense and reduces ongoing cost and IaaS customers pay on a per-user basis, typically by the hour, week, or month.
2. **Website hosting:** Running websites using IaaS can be less expensive than traditional web hosting.
3. **Security:** The IaaS Cloud Provider may provide better security than your existing software.
4. **Maintenance:** There is no need to manage the underlying data center or the introduction of new releases of the development or underlying software. This is all handled by the IaaS Cloud Provider.

The various companies providing *Infrastructure as a service* are Amazon web services, Bluestack, IBM, Openstack, Rackspace, and Vmware.

Disadvantages of laaS :

1. **Limited control over infrastructure:** IaaS providers typically manage the underlying infrastructure and take care of maintenance and updates, but this can also mean that users have less control over the environment and may not be able to make certain customizations.
2. **Security concerns**: Users are responsible for securing their own data and applications, which can be a significant undertaking.
3. **Limited access:** Cloud computing may not be accessible in certain regions and countries due to legal policies.

Anything as a Service

It is also known as Everything as a Service. Most of the cloud service providers nowadays offer anything as a service that is a compilation of all of the above services including some additional services.

Advantages of XaaS:

1. **Scalability:** XaaS solutions can be easily scaled up or down to meet the changing needs of an organization.
2. **Flexibility:** XaaS solutions can be used to provide a wide range of services, such as storage, databases, networking, and software, which can be customized to meet the specific needs of an organization.
3. **Cost-effectiveness**: XaaS solutions can be more cost-effective than traditional on-premises solutions, as organizations only pay for the services.

Disadvantages of XaaS:

1. **Dependence on the provider:** Users are dependent on the XaaS provider for the availability, scalability, and reliability of the service, which can be a risk if the provider experiences outages or other issues.
2. **Limited flexibility**: XaaS solutions may not be able to accommodate certain types of workloads or applications, which can limit the value of the solution for certain organizations.

3. **Limited integration:** XaaS solutions may not be able to integrate with existing systems and data sources, which can limit the value of the solution for certain organizations.

Function as a Service :

FaaS is a type of cloud computing service. It provides a platform for its users or customers to develop, compute, run and deploy the code or entire application as functions. It allows the user to entirely develop the code and update it at any time without worrying about the maintenance of the underlying infrastructure. The developed code can be executed with response to the specific event. It is also **as same as PaaS**.

FaaS is an event-driven execution model. It is implemented in the serverless container. When the application is developed completely, the user will now trigger the event to execute the code. Now, the triggered event makes response and activates the servers to execute it. The servers are nothing but the Linux servers or any other servers which is managed by the vendor completely. Customer does not have clue about any servers which is why they do not need to maintain the server hence it is **serverless architecture.**

Both PaaS and FaaS are providing the same functionality but there is still some differentiation in terms of Scalability and Cost.

FaaS, provides auto-scaling up and scaling down depending upon the demand. PaaS also provides scalability but here users have to configure the scaling parameter depending upon the demand.

In FaaS, users only have to pay for the number of execution time happened. In PaaS, users have to pay for the amount based on pay-as-you-go price regardless of how much or less they use.

Advantages of FaaS :

- **Highly Scalable:** Auto scaling is done by the provider depending upon the demand.
- **Cost-Effective:** Pay only for the number of events executed.
- **Code Simplification:** FaaS allows the users to upload the entire application all at once. It allows you to write code for independent functions or similar to those functions.
- Maintenance of code is enough and no need to worry about the servers.
- Functions can be written in any programming language.
- Less control over the system.

The various companies providing Function as a Service are Amazon Web Services – Firecracker, Google – Kubernetes, Oracle – Fn, Apache OpenWhisk – IBM, OpenFaaS,

Disadvantages of FaaS :

1. **Cold start latency**: Since FaaS functions are event-triggered, the first request to a new function may experience increased latency as the function container is created and initialized.
2. **Limited control over infrastructure:** FaaS providers typically manage the underlying infrastructure and take care of maintenance and updates, but this can also mean that users have less control over the environment and may not be able to make certain customizations.
3. **Security concerns:** Users are responsible for securing their own data and applications, which can be a significant undertaking.

4. **Limited scalability**: FaaS functions may not be able to handle high traffic or large number of requests.

Enterprise cloud software applications can help businesses improve their workflows by giving them access to best-in-class enterprise resource planning (ERP) tools. Such cloud software is easier to deploy, manage, and budget than an on-premises solution.

However, cloud solutions are not always perfectly understood—even by companies that eagerly seek to deploy them. This often leads to inefficiencies that reduce the ROI businesses receive from implementing new enterprise software in the cloud.

What is the cloud? What do companies need to have to use the cloud *effectively* so they can create a competitive advantage? And, what are the benefits of using cloud applications versus on-premises applications?

What is an Enterprise Cloud Application?

The term “cloud services” can cover a wide variety of tools and resources that are accessed remotely. Some of the major categories include:

- **Software as a Service.** When an enterprise application is delivered by accessing remote servers hosted by a cloud service provider (CSP).
- **Platform as a Service.** When enterprises develop their own software but run it on the CSP’s platform.
- **Infrastructure as a Service.** When enterprises access computing resources remotely to reduce costs while developing and maintaining their own platforms.

Enterprise cloud applications fall under the category of software-as-a-service (SaaS), offering companies access to a fully-developed enterprise management solution that can be configured to meet the company's needs. While these enterprise applications are designed to minimize the amount of work a business will need to put in to use them, there are still a few things that companies need to successfully implement such cloud applications to the fullest.

Requirements of Cloud Applications

There are several things that businesses need to have to get the most out of any new enterprise cloud application. This list includes, but is not limited to:

1. **A Comprehensive Implementation Plan.** The viability of cloud solutions is often predicated on whether they are implemented correctly. Having an implementation strategy in place that accounts for critical success factors is foundational for getting the most out of a cloud-based enterprise software.
2. **Configuration of Cloud Applications for Enterprise Workflows.** While most enterprise cloud software developers will try their best to create a one size fits all solution, there are too many variables between companies for any one software to be perfect for all users. So, an important element for successful cloud adoption is to configure the enterprise software to conform to the needs of the company's internal processes. This helps to better integrate the cloud solution with the business' existing workflows, smoothing out the transition for employees.
3. **Ongoing Management Support of Cloud Solution Adoption.** Employees often take their cues from their leaders. Because of this fact, it is crucial for the leadership in an enterprise to demonstrate their support for the adoption of new enterprise cloud solutions. Otherwise, employees may not take the new software seriously and neglect to make it part of their revised process workflows.
4. **A Strong Business Case for the Enterprise Cloud Software.** Simply adopting a new enterprise cloud software solution because it's there is not a sound business strategy. When preparing to implement any new enterprise application, there has to be a thorough understanding of how the solution will be used and what its business benefits are. This can be critical for justifying the adoption of the new solution to investors or the company's leadership.

This list is not, by any means, completely comprehensive of all the things needed for a successful enterprise cloud implementation. Depending on the solution and the business plan for using it, there can be hundreds of technical or process requirements for the solution. This is why conducting a cloud migration assessment is important.

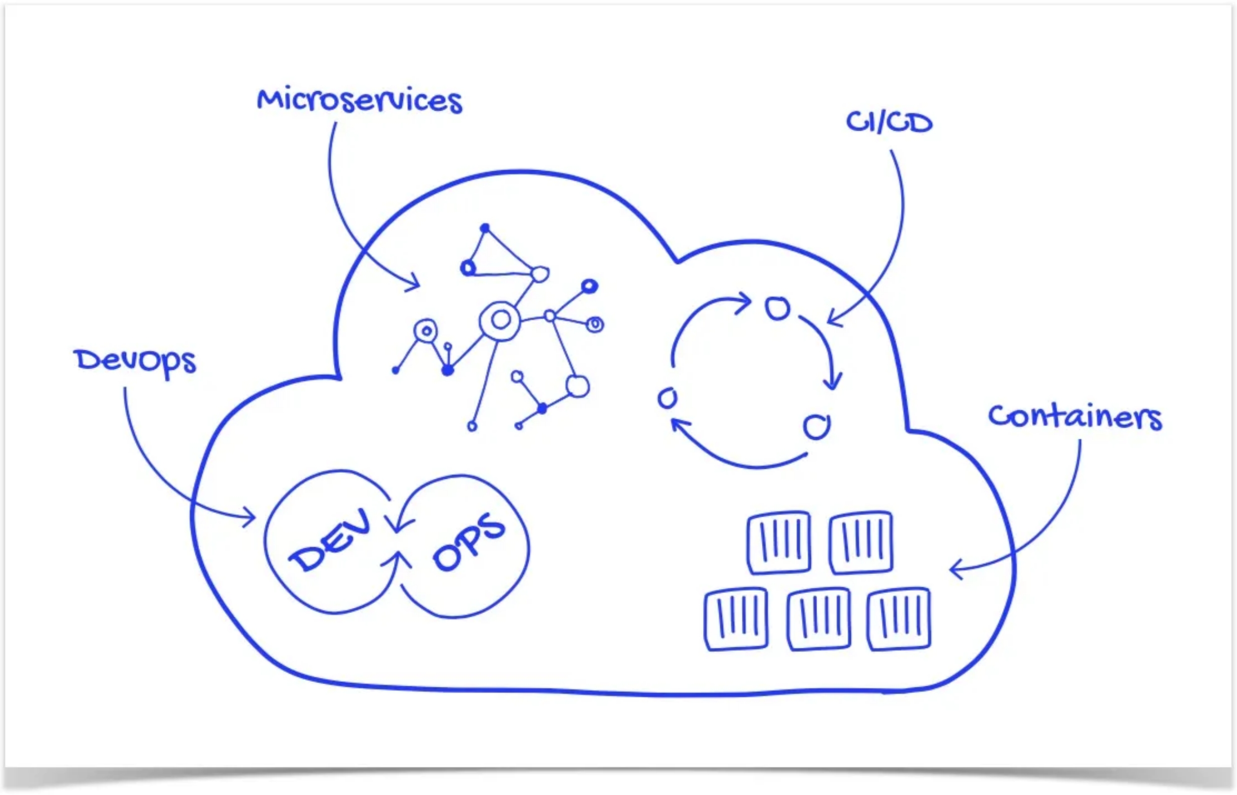
Benefits of Enterprise Cloud Solutions

So, what are the benefits of enterprise cloud software versus using an on-premises solution? Some of the benefits of using cloud services to deliver enterprise applications include:

- **Improved Scalability.** With a cloud service, adding or removing capacity to meet changing resource demands is easy. When requirements are at their peak (such as during the holiday rush), the CSP only needs to allocate more resources to match. When requirements die down (such as a post-holiday slump), the CSP can scale back resource allocation. This allows enterprises to pay for only the resources they need, when they need them. Compare this to an on-premises solution, where the infrastructure upgrades needed to accommodate peak use are permanent, causing invested money to go to waste.
- **Reduced Maintenance Costs.** When using an enterprise cloud application, the business does not have to maintain its own servers, data centers, etc. This helps to minimize IT maintenance costs by offsetting them onto the CSP. The CSP, in turn, is able to divide their own costs for upgrades, maintenance, and security amongst numerous customers.
- **Centralized Data Management.** Cloud enterprise application services allow the data required for the application to be stored in a centralized location. This makes implementing remote work easier and more secure since employees working from afar don't have to connect to the enterprise's main databases or servers. Putting these files in the cloud makes them easier to share with team

members, and reduces the risk of such files being accidentally deleted forever when a user’s computer is lost or damaged.

Introduction



Cloud native is an approach for building applications as micro-services and running them on a containerised and dynamically orchestrated platforms that fully exploits the advantages of the cloud computing model. Cloud-native is about how applications are created and deployed, not where. Such technologies empower organisations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. These applications are built from the ground up, designed as loosely coupled systems, optimised for cloud scale and performance, use managed services and take advantage of continuous delivery to achieve reliability and faster time to market. The overall objective is to improve speed, scalability and, finally, margin.

Speed — Companies of all sizes now see a strategic advantage in being able to move quickly and get ideas to market fast. By this, we mean moving from months to get an idea into production to days or even hours. Part of achieving this is a cultural shift within a business, transitioning from big bang projects to more incremental improvements. At its heart, a Cloud Native strategy is about handling technical risk. In the past, our standard approach to avoiding danger was to move slowly and carefully. The Cloud Native approach is about moving quickly by taking small, reversible and low-risk steps.

Scalability — As businesses grow, it becomes strategically necessary to support more users, in more locations, with a broader range of devices, while maintaining responsiveness, managing costs and not falling over

Margin — In the new world of cloud infrastructure, the strategic goal is to be to pay for additional resources only as needed — as new customers come online. Spending moves from up-front CAPEX (buying new machines in anticipation of success) to OPEX (paying for additional servers on-demand).

Role Of CNCF

Cloud Native Computing Foundation is an open source software foundation housed in the Linux Foundation and includes big names such as Google, IBM, Intel, Box, Cisco, and VMware etc, dedicated to making cloud-native computing universal and sustainable. Cloud native computing uses 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 utilisation. According to the FAQ on why CNCF is needed — Companies are realising that they need to be a software company, even if they are not in the software business. For example, Airbnb is revolutionising the hospitality industry and more traditional hotels are struggling to compete. Cloud native allows IT and software to move faster. Adopting cloud-native technologies and practices enables companies to create software in-house, allows business people to closely partner with IT people, keep up with competitors and deliver better services to their customers.

Cloud Native Design Principles

Designed As Loosely Coupled Microservices

Microservice is an approach to develop a single application as a suite of small services, each running in their own process and communicating using lightweight protocols like HTTP. These services are built around business capabilities and are independently deployable by fully automated deployment machinery.

Developed With Best-of-breed Languages And Frameworks

Each service of a cloud-native application is developed using the language and framework best suited for the functionality. Cloud-native applications are polyglot. Services use a variety of languages, runtimes and frameworks. For example, developers may build a real-time streaming service based on WebSockets, developed in Node.js, while choosing Python for building a machine learning based service and choosing spring-boot for exposing the REST APIs. The fine-grained approach to developing microservices lets them choose the best language and framework for a specific job.

Centred Around APIs For Interaction And Collaboration

Cloud-native services use lightweight APIs that are based on protocols such as representational state transfer (REST) to expose their functionality. Internal services communicate with each other using binary protocols like Thrift, Protobuff, GRPC etc for better performance

Stateless And Massively Scalable

A cloud-native app stores its state in a database or some other external entity so instances can come and go. Any instance can process a request. They are not tied to the underlying infrastructure which allows the app to run in a highly distributed manner and still maintain its state independent of the elastic nature of the underlying infrastructure. From a scalability perspective, the architecture is as simple as just by adding commodity server nodes to the cluster, it should be possible to scale the application

Resiliency At The Core Of the Architecture

According to Murphy's law — "Anything that can fail will fail". When we apply this to software systems, In a distributed system, failures will happen. Hardware can fail. The network can have transient failures. Rarely, an entire service or region may experience a disruption, but even those must be planned for. Resiliency is the ability of a system to recover from failures and continue to function. It's not about *avoiding* failures, but responding to failures in a way that avoids downtime or data loss. The goal of resiliency is to return the application to a fully functioning state following a failure. Resiliency offers the following:

- High Availability — the ability of the application to continue running in a healthy state, without significant downtime
- Disaster Recovery — the ability of the application to recover from rare but major incidents: non-transient, wide-scale failures, such as service disruption that affects an entire region

One of the main ways to make an application resilient is through redundancy. HA and DR are implemented using multi node clusters, Multi region deployments, data replication, no single point of failure, continuous monitoring etc.

Following are some of the strategies for implementing resiliency:

- Retry transient failures — Transient failures can be caused by momentary loss of network connectivity, a dropped database connection, or a timeout when a service is busy. Often, a transient failure can be resolved simply by retrying the request
- Load balance across instances — Implement cluster everywhere. Stateless applications should be able to scale by adding more nodes to the cluster
- Degrade gracefully — If a service fails and there is no failover path, the application may be able to degrade gracefully while still providing an acceptable user experience
- Throttle high-volume tenants/users — Sometimes a small number of users create excessive load. That can have an impact on other users, reducing the overall availability of your application

- Use a circuit breaker — The circuit breaker pattern can prevent an application from repeatedly trying an operation that is likely to fail. The circuit breaker wraps calls to a service and tracks the number of recent failures. If the failure count exceeds a threshold, the circuit breaker starts returning an error code without calling the service
- Apply compensating transactions. A compensating transaction is a transaction that undoes the effects of another completed transaction. In a distributed system, it can be very difficult to achieve strong transactional consistency. Compensating transactions are a way to achieve consistency by using a series of smaller, individual transactions that can be undone at each step.

Testing for resiliency — Normally resiliency testing cannot be done the same way that you test application functionality (by running unit tests, integration tests and so on). Instead, you must test how the end-to-end workload performs under failure conditions which only occur intermittently. For example: inject failures by crashing processes, expired certificates, make dependent services unavailable etc. Frameworks like chaos monkey can be used for such chaos testing.

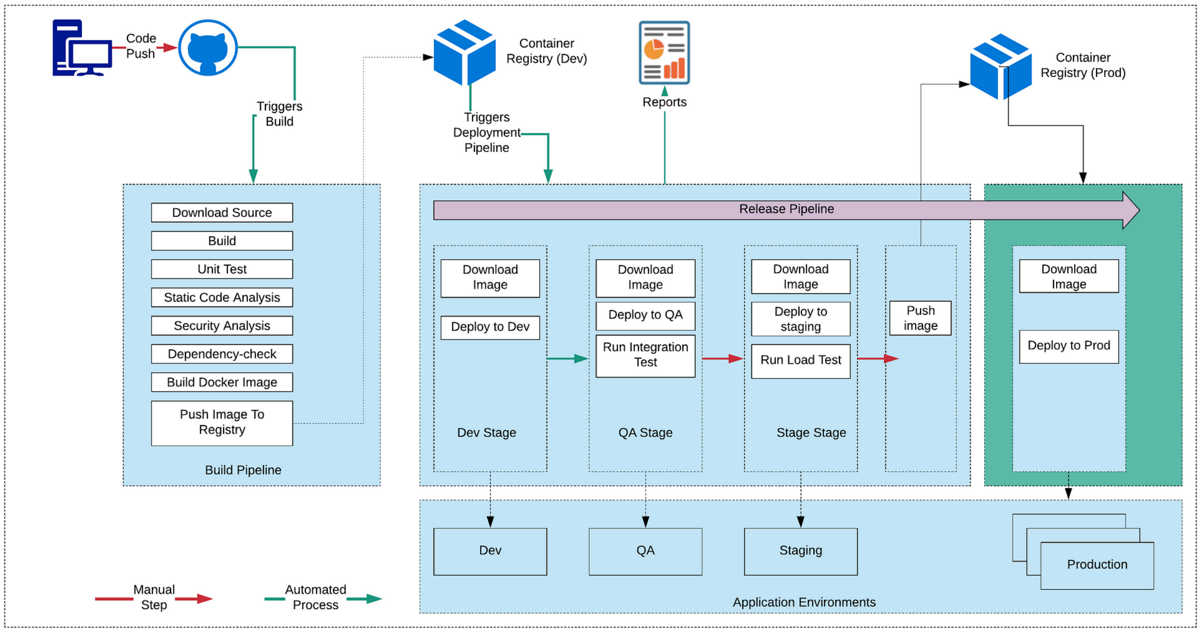
Packaged As Lightweight Containers And Orchestrated

Containers make it possible to isolate applications into small, lightweight execution environments that share the operating system kernel. Typically measured in megabytes, containers use far fewer resources than virtual machines and start up almost immediately. Docker has become the standard for container technology. The biggest advantage they offer is portability.

Cloud-native applications are deployed using Kubernetes which is an open source platform designed for automating deployment, scaling, and management of containerised applications. Originally developed at Google, Kubernetes has become the operating system for deploying cloud-native applications. It is also one of the first few projects to get graduated at CNCF.

Agile DevOps & Automation Using CI/CD

DevOps, the amalgamation of “development” and “operations describes the organisational structure, practices, and culture needed to enable rapid agile development and scalable, reliable operations. DevOps is about the culture, collaborative practices, and automation that aligns development and operations teams so they have a single mindset on improving customer experiences, responding faster to business needs, and ensuring that innovation is balanced with security and operational needs. Modern organisations believe in merging of development and operational people and responsibilities so that one DevOps team carries both responsibilities. In that way you just have one team who takes the responsibility of development, deployment and running the software in production.



Continuous Integration & Continuous Delivery

Continuous integration (CI) and continuous delivery (CD) is a set of operating principles that enable application development teams to deliver code changes more frequently and reliably. The technical goal of CI is to establish a consistent and automated way to build, package, and test applications. With consistency in the integration process in place, teams are more likely to commit code changes more frequently, which leads to better collaboration and software quality.

Continuous delivery picks up where continuous integration ends. CD automates the delivery of applications to selected infrastructure environments. It picks up the package built by CI, deploys into multiple environments like Dev, QA, Performance, Staging runs various tests like integration tests, performance tests etc and finally deploys into production. Continuous delivery normally has few manual steps in the pipeline whereas continuous deployment is a fully automated pipeline which automates the complete process from code checkin to production deployment.

Elastic — Dynamic scale-up/down

Cloud-native apps take advantage of the elasticity of the cloud by using increased resources during a use spike. If your cloud-based e-commerce app experiences a spike in use, you can have it set to use extra compute resources until the spike subsides and then turn off those resources. A cloud-native app can adjust to the increased resources and scale as needed.

Cloud Native Applications: Key Characteristics and Applications

By Simplilearn

Last updated on May 25, 2023

3106

Table of Contents

Increasing numbers of people, businesses, and organizations are turning to the cloud for their IT needs. Additionally, the demand for applications continues to grow as everyone's reliance on digital resources increases. Not only that, but consumers also want their existing applications to improve and give them a better user experience.

Now, imagine fusing these two cloud concepts: computing and applications, together. You get cloud native applications as a result, and this is a very productive partnership.

How much do you know about cloud native applications? If you're not familiar with the concept, then read on. This article explores the game-changing potential of these applications, what they are, their defining characteristics, their ups and downs, and how to start working with them.

Enhance your career prospects with expert-led University Courses at Simplilearn. Start learning today and transform your future.

What Are Cloud Native Applications?

When you hear the word "native," you will most likely think "associated with or indigenous to a specific area," as in "this plant is native to the Sahara," or "I am a native Bostonian." And yes, you'd be right — that's how people usually use the term.

However, in the context of cloud native applications, the word "native" describes less of where the application comes from and more about what components are used to build and deploy it. The "cloud native" approach of software application development takes advantage of the cloud computing delivery model. So, it is less of a "where" the application exists and more of "how" it was created.

Thus, a native cloud application is an application that's developed with cloud-based technologies and fully hosted and managed in the cloud. The application runs in the cloud from end to end; it has been written, tested, and deployed in the cloud, using cloud-based technologies and services.

Bear in mind, there is a profound difference between a cloud-based and a cloud-native application. A cloud-based application can be an older software program that's adapted to run in the cloud. Cloud native applications are designed, run, and maintained in a cloud environment enabling you to build and run your scalable app in a public, private, or hybrid cloud.

Defining Characteristics of a Cloud Native Application

There are four main characteristics of these applications:

- First, they emphasize high flexibility and agility, which means increased performance, better security, and an improved customer experience.
- You can apply changes, customize the application, and run new features, all at faster speeds.
- They don't rely on a single monolithic software codebase; rather, they are constructed in a modular way, taking advantage of cloud computing frameworks and multiple infrastructures.
- Finally, they consist of reusable, discrete components called microservices, designed to integrate into all types of cloud environments.

10 Key Attributes of Cloud Native Applications

The following are the ten main attributes common to all of these applications.

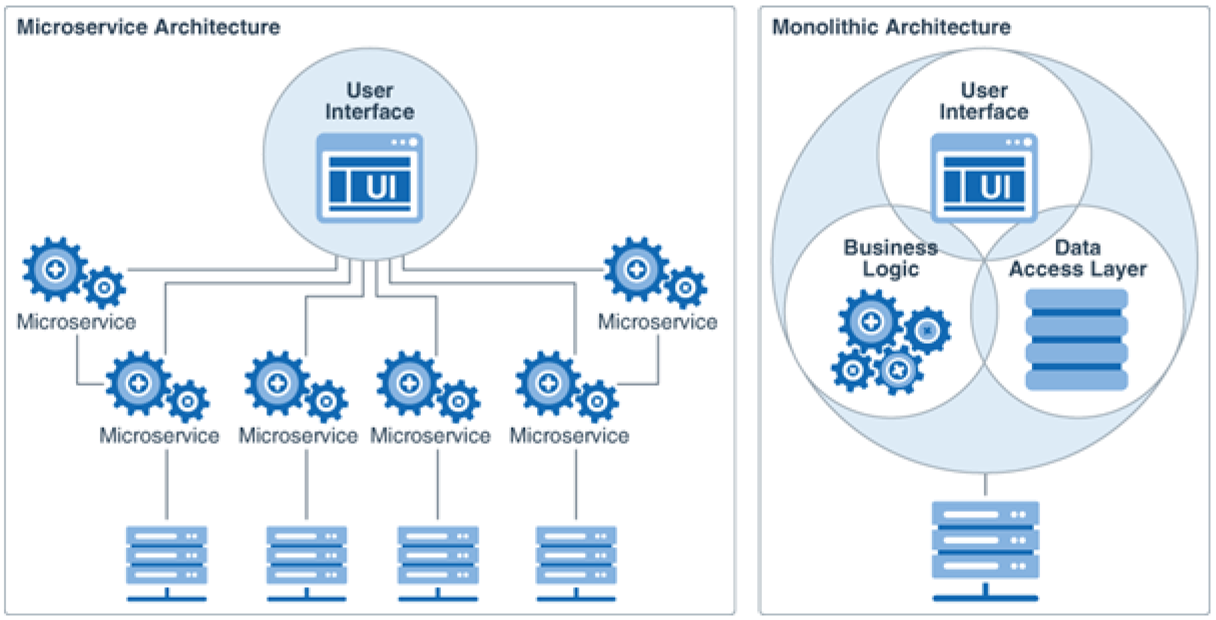
1. The applications are designed as loosely coupled microservices. Services belonging to the same application find each other via the application runtime but existing independently from the other services. When you correctly integrate the elastic infrastructure and application architectures, they are scaled out with efficiency and high performance.
2. These applications are deployed on a self-service, virtual, shared, and elastic infrastructure. As a result, the applications can align with the underlying infrastructure, adjusting themselves to the varying load, dynamically shrinking and growing as needed.
3. They are isolated from the server and operating system dependencies. In other words, they don't have an affinity for a particular machine or operating system. These applications function at a higher abstraction level. However, if a microservice needs specific capabilities like graphics processing units (GPUs) or solid-state drives (SSDs), a subset of machines can only provide.
4. Each application service goes through an independent life cycle managed through an Agile DevOps process. Developers can have several continuous integration/continuous delivery (CI/CD) pipelines working in tandem, deploying and managing cloud native applications.
5. They are packaged as lightweight containers, a collection of autonomous, independent services. Containers, unlike virtual machines, can quickly scale-in and scale-out. Scaling shifts to containers optimizes infrastructure utilization.
6. Each application service is developed using best-of-breed languages and frameworks that are best suited for the functionality. These applications are polyglot, using different languages, runtimes, and frameworks. For instance, you could build a real-time streaming service based on WebSockets, developed with Node.js, and choose Python to expose the application programming interface (API). This approach to developing microservices lets you choose the perfect language and framework for the specific task.
7. Centered around APIs for interaction and collaboration: Cloud-native services utilize lightweight APIs based on protocols like NATS, representational state

transfer (REST), or Google's own open-source remote procedure call (gRPC). NATS contains publish-subscribe features that facilitate asynchronous communication within the application. Developers and programmers use REST to expose APIs over hypertext transfer protocol (HTTP). Users rely on gRPC for performance, and typically used to let services communicate with each other.

8. The applications are architected with a clean distinction between stateless and stateful services since the two services exist independently. Persistent and durable services follow a different pattern that ensures higher availability and resiliency.
9. They can be automated to a greater extent, meshing well with the infrastructure as code concept. In fact, large and complex applications require a certain level of automation.
10. Finally, applications that are native to the cloud line up with governance models defined through a fixed set of allocation policies. They conform to policies like storage and central processing unit (CPU) quotas and network policies responsible for allocating resources to the services.

What Are Microservices?

Microservice architecture, also known as 'microservices,' is a development method that breaks down software into modules with specialized functions and detailed interfaces.



Difference Between Microservices and Monolithic Architectures

Source: microservices.io Opens a new window

Microservices have grown increasingly popular in the last few years as organizations adopt DevOps and continuous testing processes to become more agile. Leading online companies such as Amazon, eBay, Netflix, PayPal, Twitter, and Uber have dropped monolithic architectures and moved to microservices.

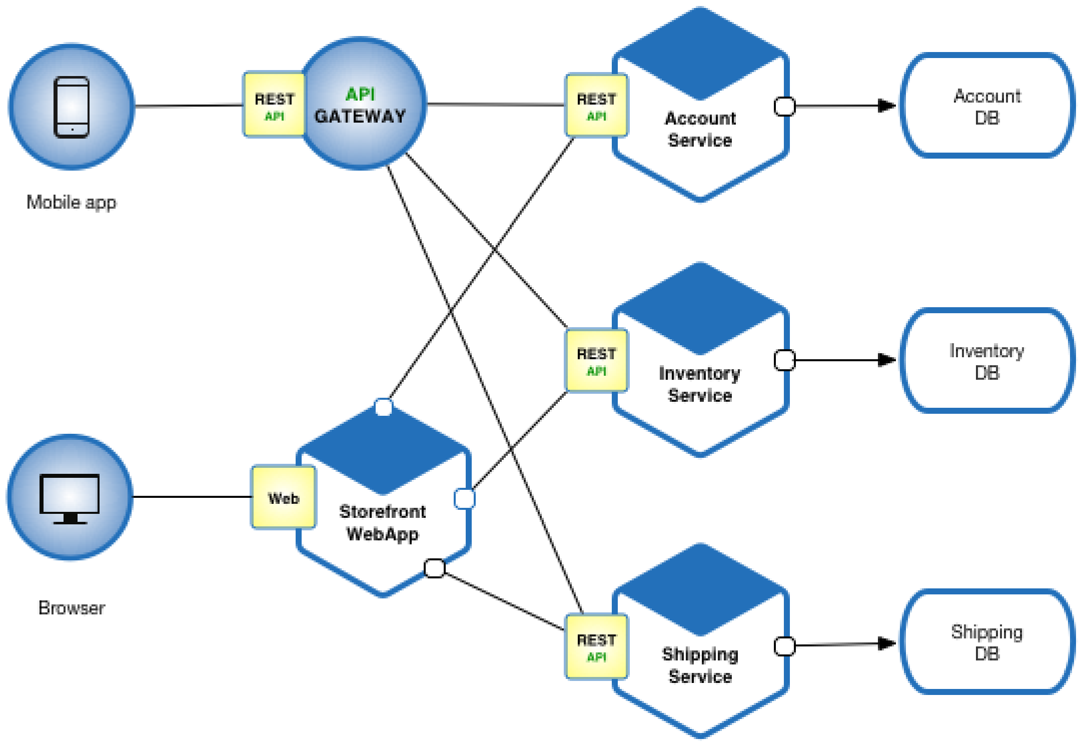
The monolithic architecture consists of applications built as large, autonomous units. Such applications cannot be changed easily because the entire system is heavily interconnected. Even a tiny modification to the code is likely to require creating and deploying a completely new version of the software. Monolithic applications are also difficult to scale, as scaling a specific function would require scaling the complete application.

Microservices address these problems of monolithic architecture by taking a modular approach to software development. In simple terms, microservices reimagine applications as a combination of multiple individual, interconnected services. Each service runs a specialized process and is deployed independently. If needed, the services can store and process data using different techniques and can be written in other programming languages.

See More: What Is DevOps? Definition, Goals, Methodology, and Best Practices

Understanding the Architecture of Microservices

In a monolithic application, all processes rely heavily on each other and operate as a single service. In such an architecture, an increase in the demand for the bandwidth of any one process would mean that the complete architecture needs to be scaled up.



The Architecture of an Ecommerce Application

Source: microservices.io Opens a new window

Since all the code in a monolithic application is deployed together on the same base, adding or enhancing features becomes a complicated process, especially as the code base expands in size and complexity. Additionally, monolithic applications might be susceptible to failure. This is because tightly coupled, essentially interdependent processes are easily affected if a single process goes down.

All this puts constraints on experimentation and can make it difficult for enterprises to stay fluid and responsive; this potentially puts them at a disadvantage in a highly dynamic, customer-centric market.

Microservices allow large applications to be split into smaller pieces that operate independently. Each ‘piece’ has its responsibilities and can carry them out regardless of what the other components are doing. A microservices-based application summons the collective services of these pieces to fulfill user requests.

The services in a microservices architecture ‘talk’ to each other using lightweight application programming interfaces (APIs) that connect to detailed interfaces. These services are created to perform specific business functions, such as monetary transactions, invoice creation, and data processing. Each service carries out a single operation. As they run independently, the services can be deployed, updated, and scaled according to the demand for their specific functions.

The implementation of a microservices architecture results in the creation of business systems that are flexible and scalable. However, shifting from monolith to microservices requires a dynamic renovation of IT infrastructure.

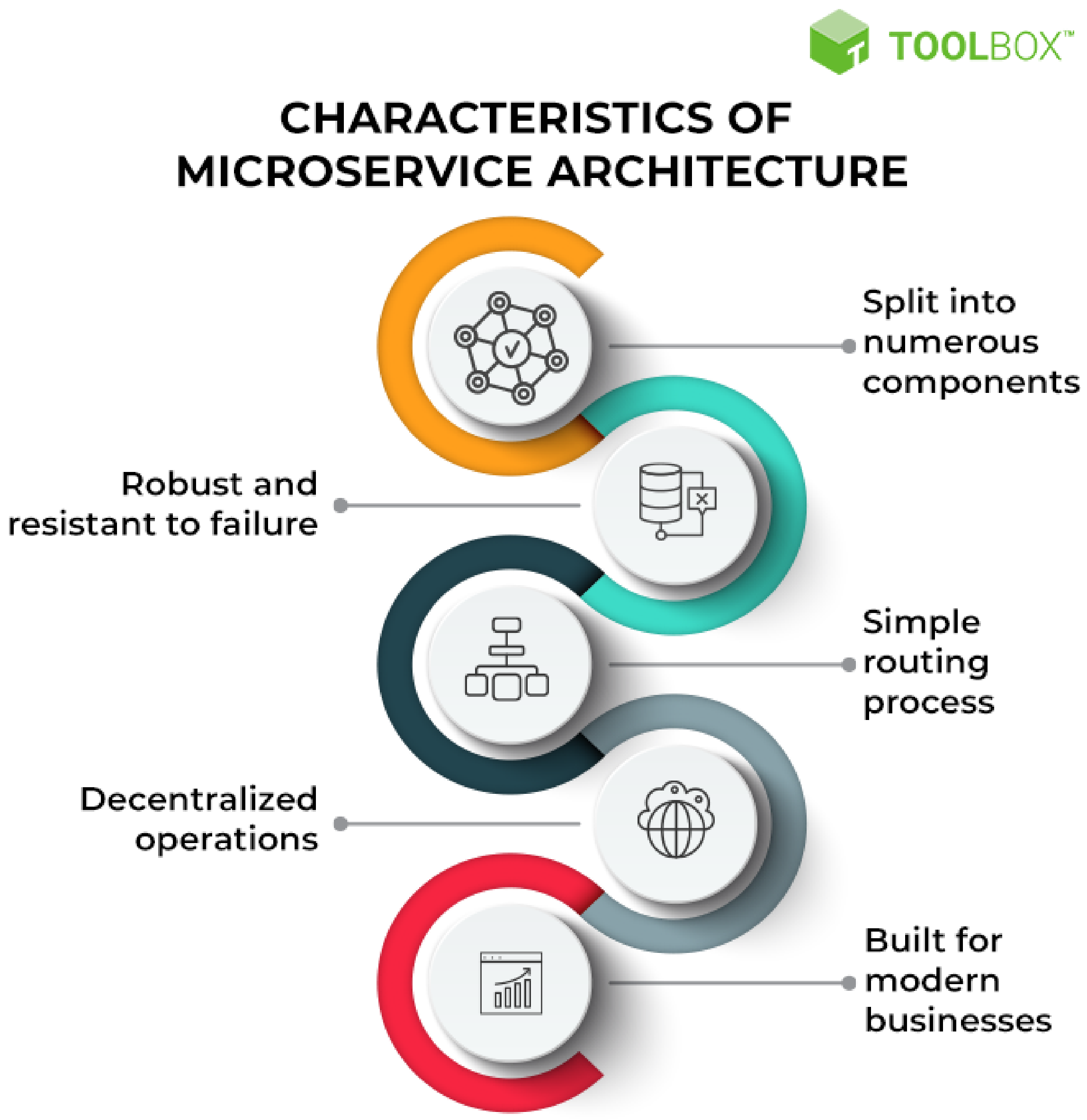
This isn’t necessarily bad, as microservices use many of the same solutions typically deployed in RESTful and web service environments. This means that they should be reasonably straightforward to work with for an adequately experienced IT team. For instance, API testing–a relatively common process–can be used to validate data flow throughout the microservices deployment.

Microservices architecture is ideal for modern digital businesses that cannot always account for all the different types of devices that will access their infrastructure. Numerous applications that started as a monolith were slowly revamped to use microservices as unforeseen requirements surfaced in the post-pandemic world. Revamping larger enterprise environments can be achieved using APIs to allow microservices to communicate with an older monolithic architecture.

Containers are an excellent example of microservices architecture as they allow businesses to focus on developing services without worrying about dependencies. Cloud-native applications are commonly built as microservices by leveraging containers.

Characteristics of a microservices architecture

Listed below are five primary characteristics of a microservices architecture.


TOOLBOX™
CHARACTERISTICS OF MICROSERVICE ARCHITECTURE
Split into numerous components
Robust and resistant to failure
Simple routing process
Decentralized operations
Built for modern businesses

Characteristics of a Microservices Architecture

1. Split into numerous components

Software built using a microservices architecture is, by definition, broken down into numerous component services. Each service can be created, deployed, and updated independently without compromising application integrity. The entire application can be scaled up by tweaking a few specific services instead of taking it down and redeploying it.

2. Robust and resistant to failure

It is not easy for an application built using a microservices architecture to fail. Of course, individual services can fail, undoubtedly affecting operations. After all, numerous diverse and unique services communicate with each other to carry out operations in a microservices environment, and failure is bound to occur at some point.

However, in a correctly configured microservices-based application, a function facing downtime should be able to reroute traffic away from itself while allowing its connected services to continue operating. It is also easy to reduce the risk of disruption by monitoring microservices and bringing them back up as soon as possible in case of failure.

3. Simple routing process

Microservices consist of intelligent components capable of processing data and applying logic. These components are connected by ‘dumb wires’ that transmit information from one element to another.

This simple routing process is the opposite of the architecture used by some other enterprise applications. For example, an enterprise service bus utilizes complex systems for message routing, choreography, and the application of business rules. Microservices, however, simply receive requests, process them, and produce an appropriate output to be transferred to the requesting component.

4. Decentralized operations

Microservices leverage numerous platforms and technologies. This makes traditional centralized governance methods inefficient for operating a microservices architecture.

Decentralized governance is better suited for microservices as developers worldwide create valuable tools to solve operational challenges. These tools can even be shared and used by other developers facing the same problems.

Similarly, a microservices architecture favors decentralized data management, as every microservice application manages its unique database. Conversely, monolithic systems typically operate using a centralized logical database for all applications.

5. Built for modern businesses

Microservices architecture is created to focus on fulfilling the requirements of modern, digital businesses. Traditional monolithic architectures have teams work on developing functions such as UI, technology layers, databases, and server-side logic. Microservices, on the other hand, rely on cross-functional teams. Each team takes responsibility for creating specific products based on individual services transmitting and receiving data through a message bus.

Monolith to Microservices:

Introduction to microservices

bookmark_border

Last reviewed 2021-06-24 UTC

This reference guide is the first in a four-part series about designing, building, and deploying microservices. This series describes the various elements of a microservices architecture. The series includes information about the benefits and drawbacks of the microservices architecture pattern, and how to apply it.

1. Introduction to microservices (this document)
2. Refactoring a monolith into microservices
3. Interservice communication in a microservices setup
4. Distributed tracing in a microservices application

This series is intended for application developers and architects who design and implement the migration to refactor a monolith application to a microservices application.

Monolithic applications

A monolithic application is a single-tiered software application in which different modules are combined into a single program. For example, if you're building an ecommerce application, the application is expected to have a modular architecture that is aligned with object-oriented programming (OOP) principles. The following diagram shows an example ecommerce application setup, in which the application consists of various modules. In a monolithic application, modules are defined using a combination of programming language constructs (such as Java packages) and build artifacts (such as Java JAR files).

Figure 1. Diagram of a monolithic ecommerce application with several modules using a combination of programming language constructs.

In figure 1, different modules in the ecommerce application correspond to business logic for payment, delivery, and order management. All of these modules are packaged and deployed as a single logical executable. The actual format depends on the application's language and framework. For example, many Java applications are packaged as JAR files and deployed on application servers such as Tomcat or Jetty. Similarly, a Rails or Node.js application is packaged as a directory hierarchy.

Monolith benefits

Monolithic architecture is a conventional solution for building applications. The following are some advantages of adopting a monolithic design for your application:

- You can implement end-to-end testing of a monolithic application by using tools like Selenium.
- To deploy a monolithic application, you can simply copy the packaged application to a server.

- All modules in a monolithic application share memory, space, and resources, so you can use a single solution to address cross-cutting concerns such as logging, caching, and security.
- The monolithic approach can provide performance advantages, because modules can call each other directly. By contrast, microservices typically require a network call to communicate with each other.

Monolith challenges

Complex monoliths often become progressively harder to build, debug, and reason about. At some point, the problems outweigh the benefits.

- Applications typically grow over time. It can become complicated to implement changes in a large and complex application that has tightly coupled modules. Because any code change affects the whole system, you have to thoroughly coordinate changes. Coordinating changes makes the overall development and testing process much longer compared to microservice applications.
- It can be complicated to achieve continuous integration and deployment (CI/CD) with a large monolith. This complexity is because you must redeploy the entire application in order to update any one part of it. Also, it's likely that you have to do extensive manual testing of the entire application to check for regressions.

- Monolithic applications can be difficult to scale when different modules have conflicting resource requirements. For example, one module might implement CPU-intensive image-processing logic. Another module might be an in-memory database. Because these modules are deployed together, you have to compromise on the choice of hardware.
- Because all modules run within the same process, a bug in any module, such as a memory leak, can potentially bring down the entire system.
- Monolithic applications add complexity when you want to adopt new frameworks and languages. For example, it is expensive (in both time and money) to rewrite an entire application to use a new framework, even if that framework is considerably better.

Microservices-based applications

A microservice typically implements a set of distinct features or functionality. Each microservice is a mini-application that has its own architecture and business logic. For example, some microservices expose an API that's consumed by other microservices or by the application's clients, such as third-party integrations with payment gateways and logistics.

Figure 1 showed a monolithic ecommerce application with several modules. The following diagram shows a possible decomposition of the ecommerce application into microservices:

Figure 2. Diagram of an ecommerce application with functional areas implemented by microservices.

In figure 2, a dedicated microserve implements each functional area of the ecommerce application. Each backend service might expose an API, and services consume APIs provided by other services. For example, to render web pages, the UI services invoke the checkout service and other services. Services might also use asynchronous, message-based communication. For more information about how services communicate with each other, see the third document in this series, [Interservice communication in a microservices setup](#).

The microservices architecture pattern significantly changes the relationship between the application and the database. Instead of sharing a single database with other services, we recommend that each service have its own database that best fits its requirements. When you have one database for each service, you ensure loose coupling between services because all requests for data go through the service API and not through the shared database directly. The following diagram shows a microservices architecture pattern in which each service has its own database:

Figure 3. Each service in a microservice architecture has its own database.

In figure 3, the order service in the ecommerce application functions well using a document-oriented database that has real-time search capabilities. The payment and delivery services rely on the strong atomicity, consistency, isolation, durability (ACID) guarantees of a relational database.

Microservices benefits

The microservices architecture pattern addresses the problem of complexity described in the preceding Monolith challenges section. A microservices architecture provides the following benefits:

- Although the total functionality is unchanged, you use microservices to separate the application into manageable chunks or services. Each service has a well-defined boundary in the form of an RPC or message-driven API. Therefore, individual services can be faster to develop, and easier to understand and maintain.
- Autonomous teams can independently develop individual services. You can organize microservices around business boundaries, not the technical capabilities of a product. You organize your teams for a single, independent responsibility for the entire lifecycle of their assigned piece of software from development to testing to deployment to maintenance and monitoring.
- Independent microservice development process also lets your developers write each microservice in a different programming language, creating a polyglot application. When you use the most effective language for each microservice, you can develop an application more quickly and optimize your application to reduce code complexity and to increase performance and functionality.
- When you decouple capabilities out of a monolith, you can have the independent teams release their microservice independently. Independent release cycles can help improve your teams' velocity and product time to market.
- Microservices architecture also lets you scale each service independently. You can deploy the number of instances of each service that satisfy its capacity and availability constraints. You can also use the hardware that best matches a

service's resource requirements. When you scale services independently, you help increase the availability and the reliability of the entire system.

The following are some specific instances in which it can be beneficial to migrate from a monolith to a microservice architecture:

- Implementing improvements in scalability, manageability, agility, or speed of delivery.
- Incrementally rewriting a large legacy application to a modern language and technology stack to meet new business demands.
- Extracting cross-cutting business applications or cross-cutting services so that you can reuse them across multiple channels. Examples of services you might want to reuse include payment services, login services, encryption services, flight search services, customer profile services, and notification services.
- Adopting a purpose-built language or framework for a specific functionality of an existing monolith.

Microservices challenges

Microservices have some challenges when compared to monoliths, including the following:

- A major challenge of microservices is the complexity that's caused because the application is a distributed system. Developers need to choose and implement an inter-services communication mechanism. The services must also handle partial failures and unavailability of upstream services.
- Another challenge with microservices is that you need to manage transactions across different microservices (also referred to as a *distributed transaction*). Business operations that update multiple business entities are fairly common, and they are usually applied in an atomic manner in which either all operations are applied or everything fails. When you wrap multiple operations in a single database transaction, you ensure atomicity.

  In a microservices-based application, business operations might be spread across different microservices, so you need to update multiple databases that different services own. If there is a failure, it's non-trivial to track the failure or success of calls to the different microservices and roll back state. The worst case scenario can result in inconsistent data between services when the rollback of state due to failures didn't happen correctly. For information about the various methodologies to set up distributed transactions between services, see the third document in this series, Interservice communication in a microservices setup.
- Comprehensive testing of microservices-based applications is more complex than testing a monolithic application. For example, to test the functionality of processing an order in a monolithic ecommerce service, you select items, add

them to a cart, and then check out. To test the same flow in a microservices-based architecture, multiple services - such as frontend, order, and payment - call each other to complete the test run.

- Deploying a microservices-based application is more complex than deploying a monolithic application. A microservice application typically consists of many services, each of which has multiple runtime instances. You also need to implement a service discovery mechanism that enables a service to discover the locations of any other services it needs to communicate with.
- A microservices architecture adds operations overhead because there are more services to monitor and alert on. Microservice architecture also has more points of failure due to the increased points of service-to-service communication. A monolithic application might be deployed to a small application server cluster. A microservices-based application might have tens of separate services to build, test, deploy and run, potentially in multiple languages and environments. All of these services need to be clustered for failover and resilience. Productionizing a microservices application requires high-quality monitoring and operations infrastructure.
- The division of services in a microservice architecture allows the application to perform more functions at the same time. However, because the modules run as isolated services, latency is introduced in the response time due to network calls between services.

- Not all applications are large enough to break down into microservices. Also, some applications require tight integration between components—for example, applications that must process rapid streams of real-time data. Any added layers of communication between services may slow real-time processing down. Thinking about the communication between services beforehand can provide helpful insights in clearly marking the service boundaries.

When deciding whether microservice architecture is best for your application, consider the following points:

- Microservice best practices require per-service databases. When you do data modeling for your application, notice whether per-service databases fit your application.
- When you implement a microservice architecture, you must instrument and monitor the environment so that you can identify bottlenecks, detect and prevent failures, and support diagnostics.
- In a microservice architecture, each service has separate access controls. To help ensure security, you need to secure access to each service both within the environment and from external applications that consume its APIs.
- Synchronous interservice communication typically reduces the availability of an application. For example, if the order service in an ecommerce application

synchronously invokes other services upstream, and if those services are unavailable, it can't create an order. Therefore, we recommend that you implement asynchronous, message-based communication.

When to migrate a monolithic application to microservices

If you're already successfully running a monolith, adopting microservices is a significant investment cost for your team. Different teams implement the principles of microservices in different ways. Each engineering team has unique outcomes for how small their microservices are, or how many microservices they need.

To determine if microservices are the best approach for your application, first identify the key business goals or pain points you want to address. There might be simpler ways to achieve your goals or address the issues that you identify. For example, if you want to scale your application up faster, you might find that autoscaling is a more efficient solution. If you're finding bugs in production, you can start by implementing unit tests and continuous integration (CI).

If you believe that a microservice approach is the best way to achieve your goals, start by extracting one service from the monolith and develop, test, and deploy it in production. For more information, see the next document in this series, Refactoring a monolith into microservices. After you have successfully extracted one service and have it running in production, start extraction of the next service and continue learning from each cycle.

The microservice architecture pattern decomposes a system into a set of independently deployable services. When you develop a monolithic application, you have to coordinate large teams, which can cause slow software development. When you implement a microservices architecture, you enable small, autonomous teams to work in parallel, which can accelerate your development.

In the next document in this series, Refactoring a monolith into microservices, you learn about various strategies for refactoring a monolithic application into microservices.

The process of transforming a monolithic application into microservices is a form of application modernization. To accomplish application modernization, we recommend that you don't refactor all of your code at the same time. Instead, we recommend that you incrementally refactor your monolithic application. When you incrementally refactor an application, you gradually build a new application that consists of microservices, and run the application along with your monolithic application. This

approach is also known as the Strangler Fig pattern. Over time, the amount of functionality that is implemented by the monolithic application shrinks until either it disappears entirely or it becomes another microservice.

To decouple capabilities from a monolith, you have to carefully extract the capability's data, logic, and user-facing components, and redirect them to the new service. It's important that you have a good understanding of the problem space before you move into the solution space.

When you understand the problem space, you understand the natural boundaries in the domain that provide the right level of isolation. We recommend that you create larger services instead of smaller services until you thoroughly understand the domain.

Defining service boundaries is an iterative process. Because this process is a non-trivial amount of work, you need to continuously evaluate the cost of decoupling against the benefits that you get. Following are factors to help you evaluate how you approach decoupling a monolith:

- Avoid refactoring everything all at once. To prioritize service decoupling, evaluate cost versus benefits.
- Services in a microservice architecture are organized around business concerns, and not technical concerns.

- When you incrementally migrate services, configure communication between services and monolith to go through well-defined API contracts.
- Microservices require much more automation: think in advance about continuous integration (CI), continuous deployment (CD), central logging, and monitoring.

The following sections discuss various strategies to decouple services and incrementally migrate your monolithic application.

Decouple by domain-driven design

Microservices should be designed around business capabilities, not horizontal layers such as data access or messaging. Microservices should also have loose coupling and high functional cohesion. Microservices are loosely coupled if you can change one service without requiring other services to be updated at the same time. A microservice is cohesive if it has a single, well-defined purpose, such as managing user accounts or processing payment.

Domain-driven design (DDD) requires a good understanding of the domain for which the application is written. The necessary domain knowledge to create the application resides within the people who understand it—the domain experts.

You can apply the DDD approach retroactively to an existing application as follows:

1. Identify a ubiquitous language—a common vocabulary that is shared between all stakeholders. As a developer, it's important to use terms in your code that a non-technical person can understand. What your code is trying to achieve should be a reflection of your company processes.
2. Identify the relevant modules in the monolithic application, and then apply the common vocabulary to those modules.
3. Define bounded contexts where you apply explicit boundaries to the identified modules with clearly defined responsibilities. The bounded contexts that you identify are candidates to be refactored into smaller microservices.

The following diagram shows how you can apply bounded contexts to an existing ecommerce application:

Figure 1. Application capabilities are separated into bounded contexts that migrate to services.

In figure 1, the ecommerce application's capabilities are separated into bounded contexts and migrated to services as follows:

- Order management and fulfillment capabilities are bound into the following categories:
    - The order management capability migrates to the order service.
    - The logistics delivery management capability migrates to the delivery service.
    - The inventory capability migrates to the inventory service.
- Accounting capabilities are bound into a single category:
    - The consumer, sellers, and third-party capabilities are bound together and migrate to the account service.

Prioritize services for migration

An ideal starting point to decouple services is to identify the loosely coupled modules in your monolithic application. You can choose a loosely coupled module as one of the first candidates to convert to a microservice. To complete a dependency analysis of each module, look at the following:

- The type of the dependency: dependencies from data or other modules.
- The scale of the dependency: how a change in the identified module might impact other modules.

Migrating a module with heavy data dependencies is usually a nontrivial task. If you migrate features first and migrate the related data later, you might be temporarily reading from and writing data to multiple databases. Therefore, you must account for data integrity and synchronization challenges.

We recommend that you extract modules that have different resource requirements compared to the rest of the monolith. For example, if a module has an in-memory database, you can convert it into a service, which can then be deployed on hosts with higher memory. When you turn modules with particular resource requirements into services, you can make your application much easier to scale.

From an operations standpoint, refactoring a module into its own service also means adjusting your existing team structures. The best path to clear accountability is to empower small teams that own an entire service.

Additional factors that can affect how you prioritize services for migration include business criticality, comprehensive test coverage, security posture of the application, and organizational buy-in. Based on your evaluations, you can rank services as described in the first document in this series, by the benefit you receive from refactoring.

Extract a service from a monolith

After you identify the ideal service candidate, you must identify a way for both microservice and monolithic modules to coexist. One way to manage this coexistence is to introduce an inter-process communication (IPC) adapter, which can help the modules work together. Over time, the microservice takes on the load and eliminates the monolithic component. This incremental process reduces the risk of moving from the monolithic application to the new microservice because you can detect bugs or performance issues in a gradual fashion.

The following diagram shows how to implement the IPC approach:

Figure 2. An IPC adapter coordinates communication between the monolithic application and a microservices module.

In figure 2, module Z is the service candidate that you want to extract from the monolithic application. Modules X and Y are dependent upon module Z. Microservice modules X and Y use an IPC adapter in the monolithic application to communicate with module Z through a REST API.

The next document in this series, Interservice communication in a microservices setup, describes the Strangler Fig pattern and how to deconstruct a service from the monolith.

Manage a monolithic database

Typically, monolithic applications have their own monolithic databases. One of the principles of a microservices architecture is to have one database for each microservice. Therefore, when you modernize your monolithic application into microservices, you must split the monolithic database based on the service boundaries that you identify.

To determine where to split a monolithic database, first analyze the database mappings. As part of the service extraction analysis , you gathered some insights on the microservices that you need to create. You can use the same approach to analyze database usage and to map tables or other database objects to the new microservices. Tools like SchemaCrawler, SchemaSpy, and ERBuilder can help you to perform such an analysis. Mapping tables and other objects helps you to understand the coupling between database objects that spans across your potential microservices boundaries.

However, splitting a monolithic database is complex because there might not be clear separation between database objects. You also need to consider other issues, such as data synchronization, transactional integrity, joins, and latency. The next section describes patterns that can help you respond to these issues when you split your monolithic database.

Reference tables

In monolithic applications, it's common for modules to access required data from a different module through an SQL join to the other module's table. The following diagram uses the previous ecommerce application example to show this SQL join access process:

Figure 3. A module joins data to a different module's table.

In figure 3, to get product information, an order module uses a `product_id` **foreign key to join an order to the products table.**

However, if you deconstruct modules as individual services, we recommend that you don't have the order service directly call the product service's database to run a join operation. The following sections describe options that you can consider to segregate the database objects.

Share data through an API

When you separate the core functionalities or modules into microservices, you typically use APIs to share and expose data. The referenced service exposes data as an API that the calling service needs, as shown in the following diagram:

Figure 4. A service uses an API call to get data from another service.

In figure 4, an order module uses an API call to get data from a product module. This implementation has obvious performance issues due to additional network and database calls. However, sharing data through an API works well when data size is limited. Also, if the called service is returning data that has a well-known rate of change, you can implement a local TTL cache on the caller to reduce network requests to the called service.

Replicate data

Another way to share data between two separate microservices is to replicate data in the dependent service database. The data replication is read-only and can be rebuilt any time. This pattern enables the service to be more cohesive. The following diagram shows how data replication works between two microservices:

Figure 5. Data from a service is replicated in a dependent service database.

In figure 5, the product service database is replicated to the order service database. This implementation lets the order service get product data without repeated calls to the product service.

To build data replication, you can use techniques like materialized views, change data capture (CDC), and event notifications. The replicated data is eventually consistent, but there can be lag in replicating data, so there is a risk of serving stale data.

Static data as configuration

Static data, such as country codes and supported currencies, are slow to change. You can inject such static data as a configuration in a microservice. Modern microservices and cloud frameworks provide features to manage such configuration data using configuration servers, key-value stores, and vaults. You can include these features declaratively.

Shared mutable data

Monolithic applications have a common pattern known as *shared mutable state*. In a shared mutable state configuration, multiple modules use a single table, as shown in the following diagram:

Figure 6. Multiple modules use a single table.

In figure 6, the order, payment, and shipping functionalities of the ecommerce application use the same `ShoppingStatus` **table to maintain the customer's order status throughout the shopping journey.**

To migrate a shared mutable state monolith, you can develop a separate ShoppingStatus microservice to manage the `ShoppingStatus` **database table. This microservice exposes APIs to manage a customer's shopping status, as shown in the following diagram:**

Figure 7. A microservice exposes APIs to multiple other services.

In figure 7, the payment, order, and shipping microservices use the ShoppingStatus microservice APIs. If the database table is closely related to one of the services, we recommend that you move the data to that service. You can then expose the data through an API for other services to consume. This implementation helps you ensure

that you don't have too many fine-grained services that call each other frequently. If you split services incorrectly, you need to revisit defining the service boundaries.

Distributed transactions

After you isolate the service from the monolith, a local transaction in the original monolithic system might get distributed between multiple services. A transaction that spans multiple services is considered a distributed transaction. In the monolithic application, the database system ensures that the transactions are atomic. To handle transactions between various services in a microservice-based system, you need to create a global transaction coordinator. The transaction coordinator handles rollback, compensating actions, and other transactions that are described in the next document in this series, Interservice communication in a microservices setup.

Data consistency

Distributed transactions introduce the challenge of maintaining data consistency across services. All updates must be done atomically. In a monolithic application, the properties of transactions guarantee that a query returns a consistent view of the database based on its isolation level.

In contrast, consider a multistep transaction in a microservices-based architecture. If any one service transaction fails, data must be reconciled by rolling back steps that have succeeded across the other services. Otherwise, the global view of the application's data is inconsistent between services.

It can be challenging to determine when a step that implements eventual consistency has failed. For example, a step might not fail immediately, but instead could block or time out. Therefore, you might need to implement some kind of time-out mechanism. If the duplicated data is stale when the called service accesses it, then caching or replicating data between services to reduce network latency can also result in inconsistent data.

The next document in the series, Interservice communication in a microservices setup, provides an example of a pattern to handle distributed transactions across microservices.

Design interservice communication

In a monolithic application, components (or application modules) invoke each other directly through function calls. In contrast, a microservices-based application consists of multiple services that interact with each other over the network.

When you design interservices communication, first think about how services are expected to interact with each other. Service interactions can be one of the following:

- One-to-one interaction: each client request is processed by exactly one service.

- **One-to-many interactions: each request is processed by multiple services.**

Also consider whether the interaction is synchronous or asynchronous:

- **Synchronous: the client expects a timely response from the service and it might block while it waits.**
- **Asynchronous: the client doesn't block while waiting for a response. The response, if any, isn't necessarily sent immediately.**

The following table shows combinations of interaction styles:

	One-to-one	One-to-many
Synchronous	Request and response: send a request to a service and wait for a response.	—
Asynchronous	Notification: send a request to a service, but no reply is expected or sent.	Publish and subscribe: the client publishes a notification message, and zero or more interested services consume the message.
	Request and asynchronous response: send a request to a service, which replies asynchronously. The client doesn't block.	Publish and asynchronous responses: the client publishes a request, and waits for responses from interested services.

Each service typically uses a combination of these interaction styles.

Implement interservices communication

To implement interservice communication, you can choose from different IPC technologies. For example, services can use synchronous request-response-based communication mechanisms such as HTTP-based REST, gRPC, or Thrift. Alternatively, services can use asynchronous, message-based communication mechanisms such as AMQP or STOMP. You can also choose from various different message formats. For example, services can use human-readable, text-based formats such as JSON or XML. Alternatively, services can use a binary format such as Avro or Protocol Buffers.

Configuring services to directly call other services leads to high coupling between services. Instead, we recommend using messaging or event-based communication:

- **Messaging: When you implement messaging, you remove the need for services to call each other directly. Instead, all services know of a message broker, and they push messages to that broker. The message broker saves these messages in a message queue. Other services can subscribe to the messages that they care about.**
- **Event-based communication: When you implement event-driven processing, communication between services takes place through events that individual services produce. Individual services write their events to a message broker.**

Services can listen to the events of interest. This pattern keeps services loosely coupled because the events don't include payloads.

In a microservices application, we recommend using asynchronous interservice communication instead of synchronous communication. Request-response is a well-understood architectural pattern, so designing a synchronous API might feel more natural than designing an asynchronous system. Asynchronous communication between services can be implemented using messaging or event-driven communication. Using asynchronous communication provides the following advantages:

- Loose coupling: An asynchronous model splits the request–response interaction into two separate messages, one for the request and another one for the response. The consumer of a service initiates the request message and waits for the response, and the service provider waits for request messages to which it replies with response messages. This setup means that the caller doesn't have to wait for the response message.
- Failure isolation: The sender can still continue to send messages even if the downstream consumer fails. The consumer picks up the backlog whenever it recovers. This ability is especially useful in a microservices architecture, because each service has its own lifecycle. Synchronous APIs, however, require the downstream service to be available or the operation fails.

- **Responsiveness: An upstream service can reply faster if it doesn't wait on downstream services. If there is a chain of service dependencies (service A calls B, which calls C, etc.), waiting on synchronous calls can add unacceptable amounts of latency.**
- **Flow control: A message queue acts as a buffer, so that receivers can process messages at their own rate.**

However, following are some challenges to using asynchronous messaging effectively:

- **Latency: If the message broker becomes a bottleneck, end-to-end latency might become high.**
- **Overhead in development and testing: Based on the choice of messaging or event infrastructure, there can be a possibility of having duplicate messages, which makes it difficult to make operations idempotent. It also can be hard to implement and test request-response semantics using asynchronous messaging. You need a way to correlate request and response messages.**
- **Throughput: Asynchronous message handling, either using a central queue or some other mechanism can become a bottleneck in the system. The backend systems, such as queues and downstream consumers, should scale to match the system's throughput requirements.**
- **Complicates error handling: In an asynchronous system, the caller doesn't know if a request was successful or failed, so error handling needs to be handled out**

of band. This type of system can make it difficult to implement logic like retries or exponential back-offs. Error handling is further complicated if there are multiple chained asynchronous calls that have to all succeed or fail.

What Are Different Types of Tests for Microservices?

Microservices can be fraught with many issues when not tested with the best tools. Discover how Parasoft's automated testing solutions can help you address microservices issues.

Thorough, accurate, and efficient methods of testing microservices are essential in today's Internet and mobile app-oriented world. In this piece, we look at:

- What do microservices do?
- How they do it.
- The different types of microservices software testing available.

Microservices Overview

When a person interacts with a website or uses an app, numerous functions operate "below the surface." When purchasing a product on Amazon, for example, you shop for the item, looking at price, size, color, and other options. Then make your selection and move to the checkout area.

From there, you choose delivery and payment options and finally complete the transaction. All the while, numerous microservices are operating. This includes your customer interaction but also the complex programming that is running unseen on the app or website, making the transaction seem seamless and easy.

Developers like to use microservices architecture because of its modular characteristics, which makes it easier than monolithic architecture to develop and test. The widespread use of function-as-a-service and cloud-native, serverless deployments like Microsoft Azure Cloud Functions and AWS Lambda have created the ideal environment for microservices to thrive in today's IT world. Microservices in these deployments allow for agility and faster delivery times.

In some ways, microservices, which incidentally have roots in monolithic architecture, came about in response to a need to address highly erratic user traffic. When developers partition application functions into separate services, the functions shrink or grow as needed to help ensure that enough processing power is available. This can get complicated when the application develops a constantly changing, dynamic network terrain of its own. As a result, rigorous functional testing is required.

Is Microservices Testing Complex?

Microservices testing can be complicated. With the proper testing tools, knowledge, and approach, it can be made less so. Let's look at some of the elements that can make microservices testing complex.

The Approach

In many respects, automatic testing of microservices is like the testing approach for applications that software developers have built on traditional architectures. Microservices employ familiar technologies like REST and message brokers for which software writers already use best practices and well-established testing tools during software development.

The special challenge that microservices present is the large number of services that comprise an application. Plus the fact that microservices are interdependent. An additional consideration is that these services must remain functioning even when other services they depend on are not available or do not respond properly.

Main Microservices Testing Types

Microservices software tests assure that the microservices do what they're supposed to do in an efficient and timely manner. Industry-wide, the three main types of software testing for microservices are:

- Functional testing for testing the business logic and behavior of the service. This is more complicated than testing in a traditional monolithic architecture because microservices do not have a UI for easy testing. The interface to be tested represents some type of remote client communicating over HTTP or some other protocol.
- Load testing for exposing areas of the application that are not designed properly and can cause crashes from high traffic volume. In microservices, each call to a service travels the network, which means other activity on the network can affect response times.
- Resiliency testing to find out how the software reacts to potential infrastructure failures. For instance, if a server running a specific service is not available, if it crashes, or if part of a network stops passing traffic. In these cases, the developer must test to determine if the microservices application can continue to run at endpoints and elsewhere.

Specific Types of Microservices Tests

When a developer needs to test the system, she or he can do so relatively easily because the microservices are separate, even though they work together. By contrast, when programmers build services on monoliths or monolithic architecture, the application code is inextricably linked, making testing difficult and slow.

To accomplish the basic tests mentioned above, developers employ the below.

Unit Testing

An often overlooked practice when testing microservices is unit testing. What are unit tests? These tests verify that the methods and classes developers write work as expected. While unit testing is a highly technical task for developers, a robust suite of unit tests provides a critical safety net for catching unintended consequences when developers change the code. And it pays dividends in alerting developers to exactly where in the code they've broken existing functionality.

This is a valuable practice for writing high-quality software. However, unit testing by itself isn't enough. As an analogy, just because all the parts of an engine are machined to perfect specification doesn't mean the engine will run and perform as expected.

Component Testing

This microservices testing does not concentrate on how the developer wrote the microservices code but instead focuses on running the microservice as a black box and testing the traffic moving over the interface. From the perspective of a single microservice, you're now testing the engine to ensure it's delivering on its requirements.

In most cases, you're testing a REST service. So you want automated tests that act as clients of the service, sending various positive and negative requests to the service and verifying the responses that the service returns.

One challenge is that it can be complex and difficult to test microservices in isolation because they often call many other microservices in order to reply to your test client's request. To test one microservice, you potentially need dozens more deployed and available for your microservice to talk to test it properly.

Service virtualization tools come to the rescue, allowing testers to simulate other microservices in order to test the microservice in isolation, thus simplifying the test environment and making test automation easier. This may involve stubs and test doubles. Think of service virtualization like hooking up an automobile engine in a lab to an artificial transmission so you can verify that it hooks up properly and delivers the expected power to the rest of the car without actually putting it in a car.

Integration Testing

When using service virtualization to simplify and stabilize testing the microservice as an individual component, you also want to test that the microservice works with the other REAL microservices involved. Developers often do this at a “QA” or “integration” stage, where many of the required systems in the overall ecosystem are deployed and integrated together. With this testing practice, you’re beginning to assemble the car to make sure every part fits and works together but you’re not testing it on the road yet.

End-To-End Testing

Also called system testing. At some point, a big web of microservices has entry points where the application’s end users interact. For example, a Netflix app on your Apple TV talks to microservices within Netflix’s data center. But they represent only a small portion of their core functionality, which are small, individual components responsible for specific things like a recommendations service, a video streaming service, an account details service, and so on. So this, too, is an opportunity for testing microservices.

Often it’s painfully slow and high maintenance to test these interactions automatically, whether web or mobile. For critical paths and user journeys, it’s a must, but being able to represent these complete or end-to-end transactions

from the end user perspective as a sequence of microservice calls has many benefits.

What is Devops?

DevOps (short for development and operations), like most new approaches, is only a buzzword for many people. Everyone talks about it, but not everyone knows what it is. In broad terms, DevOps is an approach based on lean and agile principles in which business owners and the development, operations, and quality assurance departments collaborate to deliver software in a continuous manner that enables the business to more quickly seize market opportunities and reduce the time to include customer feedback. Indeed, enterprise applications are so diverse and composed of multiple technologies, databases, end-user devices, and so on, that only a DevOps approach will be successful when dealing with these complexities. Opinions differ on how to use it, however. Some people say that DevOps is for practitioners only; others say that it revolves around the cloud. IBM takes a broad and holistic view and sees DevOps as a business-driven software delivery approach — an approach that takes a new or enhanced business capability from an idea all the way to production, providing business value to customers in an efficient manner and capturing feedback as customers engage with the capability. To do this, you need participation from stakeholders beyond just the development and operations teams. A true DevOps approach includes lines of business, practitioners, executives, partners, suppliers, and so on.

1. Introduction:

Implementing change in the business as usual is challenging and requires investment. The adoption of new technologies, methodologies, or approaches, such as DevOps, must be driven by a clear business need. This chapter provides a foundational understanding of the business need for DevOps by addressing the challenges in software development and delivery.

2. Understanding the Business Need for DevOps:

Organizations aim to solve business problems by creating innovative applications or services. However, a significant number of software projects fail, leading to missed business opportunities. The shift from traditional systems of record to modern systems of engagement necessitates a DevOps approach, particularly in delivering applications with a focus on user experience, speed, and agility.

3. Recognizing the Business Value of DevOps:

DevOps, applying agile and lean principles, is positioned as an essential business process rather than just an IT capability. The document discusses the return on investment in terms of enhanced customer experience, increased innovation capacity, and faster time to value.

- ***Enhanced Customer Experience:*** **DevOps facilitates the delivery of a differentiated and engaging customer experience, fostering loyalty and increasing market share. The ability to obtain and respond to customer feedback in systems of engagement is crucial.**
- ***Increased Capacity to Innovate:*** **Modern organizations leverage lean thinking to increase their innovation capacity. DevOps supports practices like A-B testing, allowing organizations to efficiently test and roll out improvements based on user feedback.**
- ***Faster Time to Value:*** **DevOps, as a business capability, enables organizations to deliver value rapidly and efficiently. Speeding up the software delivery process through a DevOps approach contributes to achieving organizational goals.**

4. Seeing How DevOps Works:

The document explores the key principles of DevOps: developing and testing against production-like systems, deploying with repeatable processes, monitoring and validating operational quality, and amplifying feedback loops. These principles are fundamental to adopting a holistic DevOps approach across organizations of all sizes.

- ***Develop and Test Against Production-like Systems:*** **This principle advocates for early exposure of applications to production-like systems, allowing for comprehensive testing and validation of delivery processes.**
- ***Deploy with Repeatable, Reliable Processes:*** **Automation is emphasized to create iterative, frequent, repeatable, and reliable deployment processes, reducing the risk of deployment failures.**
- ***Monitor and Validate Operational Quality:*** **DevOps encourages early and frequent monitoring of applications' functional and non-functional characteristics, providing early warnings about potential operational and quality issues.**
- ***Amplify Feedback Loops:*** **The principle calls for creating communication channels that enable rapid feedback and learning from every action, allowing stakeholders at various levels to make informed adjustments.**