

Research Assignment on:

MONOLISTIC
and
MICROSERVICE ARCHITECTURE

by
JUSTICE ISREAL AGBONMA
August 31st, 2022.

Submitted to:
Mr. Micheal Adebayo .
StringCode Ltd.

Monolithic Architecture

What is monolithic architecture?

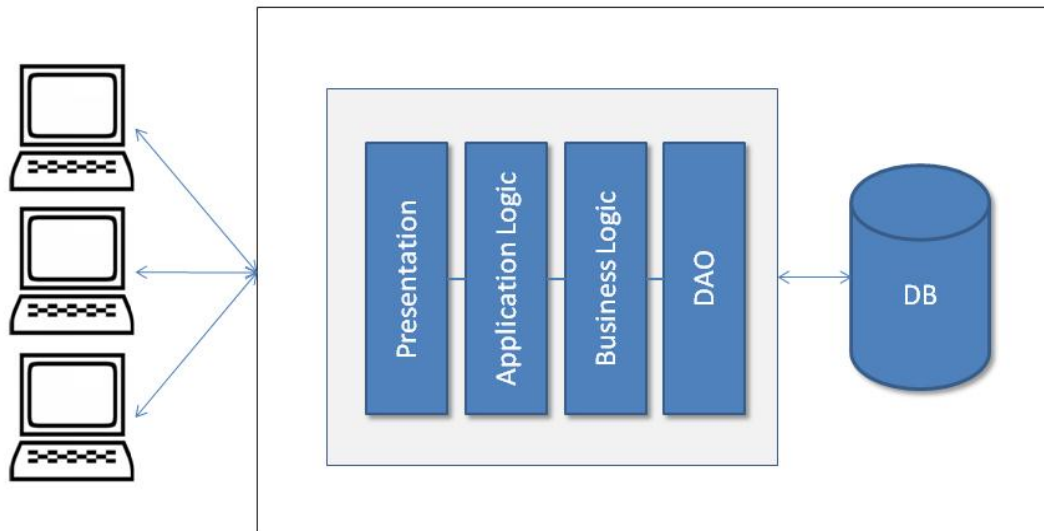
A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means "composed all in one piece." According to the Cambridge dictionary, the adjective monolithic also means both "too large" and "unable to be changed."

Monolithic software is designed to be self-contained, wherein the program's components or functions are tightly coupled rather than loosely coupled, like in modular software programs. In a monolithic architecture, each component and its associated components must all be present for code to be executed or compiled and for the software to run.

Monolithic applications are single-tiered, which means multiple components are combined into one large application. Consequently, they tend to have large codebases, which can be cumbersome to manage over time.

Furthermore, if one program component must be updated, other elements may also require rewriting, and the whole application has to be recompiled and tested. The process can be time-consuming and may limit the agility and speed of software development teams. Despite these issues, the approach is still in use because it does offer some advantages. Also, many early applications were developed as monolithic software, so the approach cannot be completely disregarded when those applications are still in use and require updates.

Monolithic architecture in software often requires a whole application to be recompiled even if only one part is changed.



eg.

To understand monolithic architecture, let's take an example of a banking application. The banking application website first authorizes customers, logs them in to their account and enables them to make online money transfers to other accounts. There are several components involved in this entire process, including the customer-facing user interface, plus services for user authentication, statement downloads, money transfers, etc.

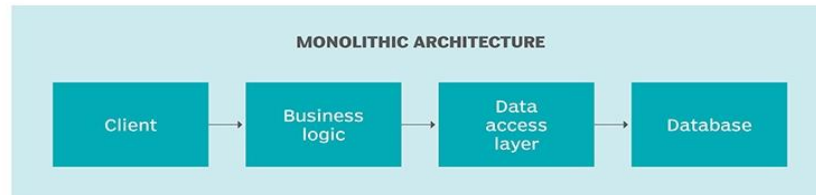
If the application uses a monolithic architecture, it is built and deployed as a single application, regardless of how a customer uses it. Thus, whether users access the application from their desktop or from a mobile device, the application remains tightly coupled, and all the various components and modules are directly connected to each other. It may also use a relational database management system as a single data source. Finally, if changes are needed for any one component, code changes are required for all other affected components as well.

Components of monolithic applications

Monolithic applications typically consist of multiple components that are interconnected to form one large application. These components may include these features:

- **Authorization.** To authorize a user and allow them to use the application.
- **Presentation.** To handle Hypertext Transfer Protocol requests and respond with Hypertext Markup Language, Extensible Markup Language or JavaScript Object Notation.
- **Business logic.** The underlying business logic that drives the application's functionality and features.
- **Database layer.** Includes the data access objects that access the application's database.
- **Application integration.** Controls and manages the application's integration with other services or data sources.

Some applications may also include a notification module to control and send automated email communications to users.



Benefits of monolithic architecture

There are benefits to monolithic architectures, which is why many applications are still created using this development paradigm. For one, monolithic programs may have better throughput than modular applications. They may also be easier to test and debug because, with fewer elements, there are fewer testing variables and scenarios that come into play.

At the beginning of the software development lifecycle, it is usually easier to go with the monolithic architecture since development can be simpler during the early stages. A single codebase also simplifies logging, configuration management, application performance monitoring and other development concerns.

Deployment can also be easier by copying the packaged application to a server. Finally, multiple copies of the application can be placed behind a load balancer to scale it horizontally.

That said, the monolithic approach is usually better for simple, lightweight applications. For more complex applications with frequent expected code changes or evolving scalability requirements, this approach is not suitable.

The single codebase of monolithic software architecture tasks such as configuration management and other development concerns.

Drawbacks of monolithic architecture

Generally, monolithic architectures suffer from drawbacks that can delay application development and deployment. These drawbacks become especially significant when the product's complexity increases or when the development team grows in size.

The codebase of monolithic applications can be difficult to understand because they may be extensive, which can make it difficult for new developers to modify the code to meet changing business or technical requirements. As requirements evolve or become more complex, it becomes difficult to correctly implement changes without hampering the quality of the code and affecting the overall operation of the application.

Following each update to a monolithic application, developers must compile the entire codebase and redeploy the full application rather than just the part that was updated. This makes continuous or regular deployments difficult, which then affects the application's and team's agility.

The application's size can also increase startup time and add to delays. In some cases, different parts of the application may have conflicting resource requirements. This makes it harder to find the resources required to scale the application.

In addition to limited scalability, reliability is another concern with monolithic software. A bug in any one component can potentially bring down the entire application. Considering the banking application example, suppose there's a memory leak in the user authorization module. This bug can bring the entire application down and make it unavailable to all users.

Finally, by virtue of their size and complexity, monolithic applications are not particularly adaptable to new technologies. A new development framework or language can affect the application as a whole, so adopting it can be both time-consuming and costly. Small organizations or companies on tight budgets may not have the funds or staff available to update the application, so they may end up maintaining the status quo, potentially leaving them unable to take advantage of a new language or framework.

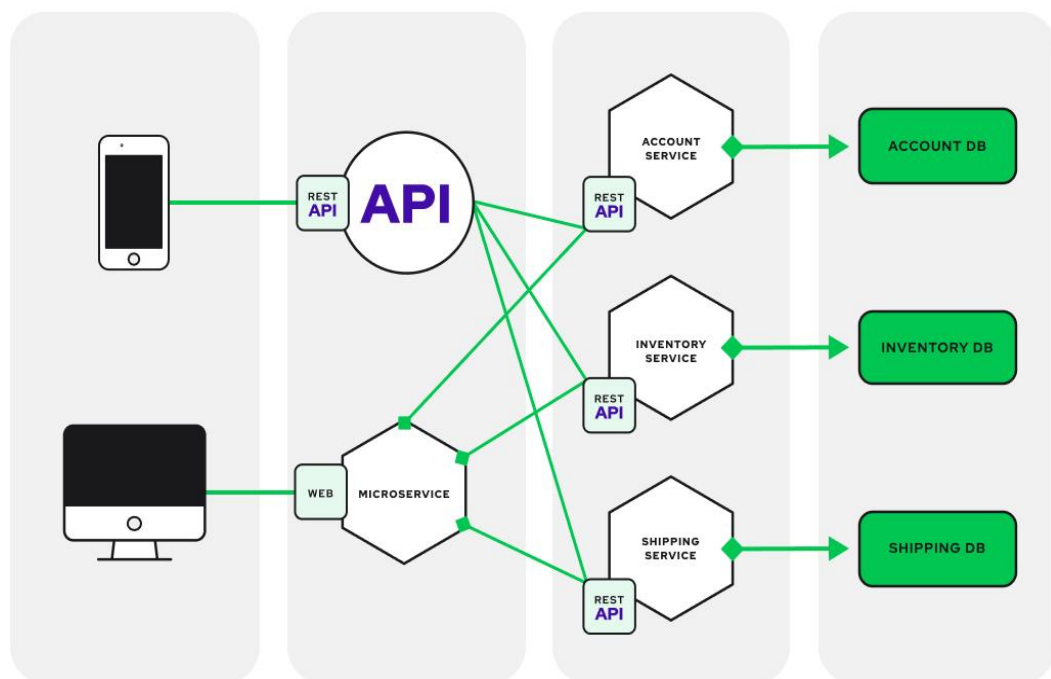
Today, many organizations are moving away from monolith architectures and adopting a microservices architecture (MSA) because it offers multiple advantages.

Microservices Architecture

What is microservices architecture?

Microservices architecture (often shortened to microservices) refers to an architectural style for developing applications. Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility. To serve a single user request, a microservices-based application can call on many internal microservices to compose its response.

Containers are a well-suited microservices architecture example, since they let you focus on developing the services without worrying about the dependencies. Modern cloud-native applications are usually built as microservices using containers.



A microservices architecture is a type of application architecture where the application is developed as a collection of services. It provides the framework to develop, deploy, and maintain microservices architecture diagrams and services independently. Within a microservices architecture, each microservice is a single service built to accommodate an application feature and handle

discrete tasks. Each microservice communicates with other services through simple interfaces to solve business problems.

Microservices architecture, often called microservices, is an architectural approach or style of application development. It involves dividing large applications into smaller, functional units capable of functioning and communicating independently.

Microservices architecture was developed to overcome the challenges and difficulties posed by monolithic architectural approaches to application development.

Monolithic architecture is like a large container holding all software components of an application: user interface, business layer, and data interface. This had several limitations such as inflexibility, lack of reliability, difficulty scaling, slow development, and so on. It was to bypass these issues that microservices architecture was created.

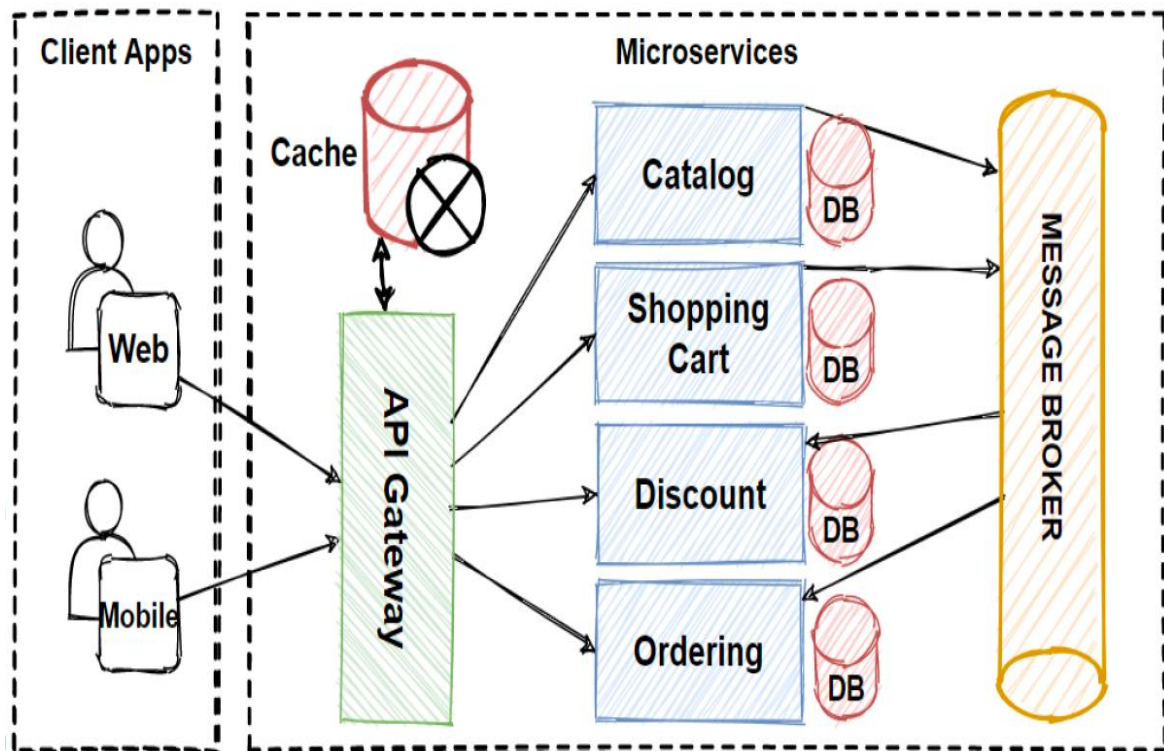
Applications running on microservices architecture are deconstructed and separated into smaller units (microservices) that the client can access using an API gateway. These microservices are each independently deployable but can communicate with one another when necessary.

Breaking down an application into microservices allows for faster development, easier bug detection and resolution, smoother maintenance, flexibility, and higher availability and scalability.

How does Microservices Architecture work?

Microservices architecture focuses on classifying the otherwise large, bulky applications. Each microservice is designed to address an application's particular aspect and function, such as logging, data search, and more. Multiple such microservices come together to form one efficient application.

This intuitive, functional division of an application offers several benefits. The client can use the user interface to generate requests. At the same time, one or more microservices are commissioned through the API gateway to perform the requested task. As a result, even larger complex problems that require a combination of microservices can be solved relatively easily.



Key Benefits of Microservices Architecture

Using microservices architecture in application development can be highly beneficial. Below are some of its key benefits.

1. Requires less development effort to scale up

Due to the independent nature of microservices, smaller development teams can parallelly work on different components to update existing functionalities. This makes it significantly easier to

not only identify hot services and scale independently from the rest of the application but also improve the application as a whole.

2. Can be deployed independently

Each microservice constituting an application needs to be a full stack. This enables microservices to be deployed independently at any point. Since the microservices are granular in nature, it's easy for development teams to work on one microservice to fix errors and then redeploy it without redeploying the entire application.

3. Microservices offer improved fault isolation

In monolithic applications, the failure of even a small component of the overall application can make it inaccessible as a whole. In some cases, determining the error could also be tedious. With microservices, isolating the problem-causing component is easy since the entire application is divided into standalone, fully functional software units. If errors occur, other non-related units will still continue to function.

4. No dependence on one Tech Stack

A technology stack refers to a set of programming languages, databases, front-end and back-end tools, frameworks, and other such components used by the developers to build an application. With microservices, developers have the freedom to pick a technology stack that is best suited for one particular microservice and its functions. They have absolute control instead of having to opt for one standardized tech stack that encompasses all of an application's functions.

What is microservices architecture used for?

Put simply, the microservices architecture makes app development quicker and more efficient. Agile deployment capabilities combined

with flexible application of different technologies drastically reduce the duration of the development cycle. The following are some of the most vital applications of microservices architecture.

Typically, microservices are used to speed up application development. Microservices architectures built using Java are common, especially Spring Boot ones. It's also common to compare microservices versus service-oriented architecture. Both have the same objective, which is to break up monolithic applications into smaller components, but they have different approaches. Here are some microservices architecture examples:

Website migration

A complex website that's hosted on a monolithic platform can be migrated to a cloud-based and container-based microservices platform. Website migration involves a substantial change and redevelopment of a website's major areas, such as its domain, structure, user interface, and so on. Using microservices will help you avoid business-damaging downtime and ensure your migration plans execute smoothly without any hassles.

Media content

Using microservices architecture, images and video assets can be stored in a scalable object storage system and served directly to web or mobile. Companies such as Netflix and Amazon Prime Video handle billions of API requests daily. Services such as OTT platforms offering their users massive media content will benefit from deploying a microservices architecture. Microservices will ensure that the plethora of requests for different subdomains worldwide is processed without delays or errors.

Transactions and invoices

Payment processing and ordering can be separated as independent units of services so payments continue to be accepted if invoicing is not working. Microservices are perfect for

applications handling high payments and transactions volumes and generating invoices for the same. The failure of an application to process payments can cause huge losses for companies. With the help of microservices, the transaction functionality can be made more robust without changing the rest of the application.

Data processing

A microservices platform can extend cloud support for existing modular data processing services.

Microservices speed up data processing tasks since applications running on microservice architecture can handle more simultaneous requests. Larger amounts of information can be processed in less time, allowing for faster and more efficient application performance.

Microservices tools

Building a microservices architecture requires a mix of tools and processes to perform the core building tasks and support the overall framework. Some of these tools are listed below.

1. Operating system

Arguably the most basic tool required to build an application is an operating system (OS). One such operating system that allows great flexibility in development and use is Linux. It offers a largely self-contained environment for the execution of program codes and a series of options for large and small applications in terms of security, storage, and networking.

2. Programming languages


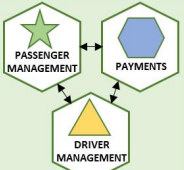
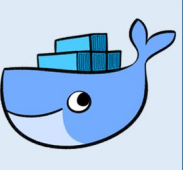



One of the benefits of using a microservices architecture is that you can use a variety of programming languages across applications for different services. Different programming languages have different utilities deployed based on the nature of the microservice.

3. API management and testing tools

When building an application using a microservices architecture, the various services need to communicate. This is accomplished using application programming interfaces (APIs). For APIs to work optimally and desirably, they need to be constantly monitored, managed and tested, and API management and testing tools are essential for this.

4. Messaging tools

Messaging tools enable microservices to communicate both internally and externally. Rabbit MQ and Apache Kafka are examples of messaging tools deployed as part of a microservice system.

| Monolith | Microservices | Docker | Kubernetes | Openshift | Istio |
|--------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
|  |  |  |  |  |  |
| Benefits Simple to Develop. Simple to Deploy. Simple Testing. | Benefits Better Scaling. New technologies. Frequent Releases. | Benefits Faster Deployment. Faster start time. Lightweight Image. | Benefits Storage options. Auto-scaling. Self-healing. | Benefits OOTB Webconsole. OOTB Build Images. Built in Jenkins. Build Images and store in Registry. | Benefits Tracing. Circuit Breaker. Canary Release. Dark Launches. Telemetry. |
| Challenges Hard to scale. Too Large/Complex. Redeploy the entire application on each update. | Challenges Slower Deployment. Service Discovery. Complex Configs. Load Balancing. Central Logging. | Challenges No Storage Option. No Autoscaling. No health-checks. Networking is hard. | Challenges Hard to install. No Building Image. No built in Jenkins. Separate install for Dashboard. | Challenges Requires additional API Management. Limited installation options. | Challenges Added Complexity. Adds Overhead. Required Expertise. |

5. Toolkits

Toolkits in a microservices architecture are sets of tools used to build and develop applications. There are different toolkits available to the developers, and these kits fulfill different purposes. Fabric8 and Seneca are some examples of microservices toolkits.

6. Architectural frameworks

Microservices architectural frameworks offer convenient solutions for application development and usually contain a library of code and tools to help configure and deploy an application.

7. Orchestration tools

A container is a set of executables, codes, libraries, and files necessary to run a microservice. Container orchestration tools provide a framework to manage and optimize containers within microservices architecture systems.

8. Monitoring tools

Once a microservices application is up and running, you need to constantly monitor it to ensure that everything is working smoothly and as intended. Monitoring tools help developers stay on top of how the application works and avoid potential bugs or glitches.

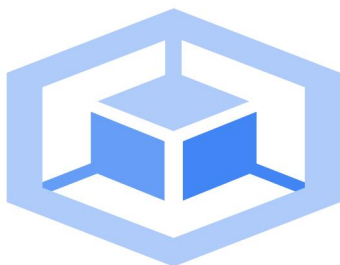
9. Serverless tools

Serverless tools further add flexibility and mobility to the various microservices within an application by eliminating server dependency. This helps in the easier rationalization and division of application tasks.

Related products and services

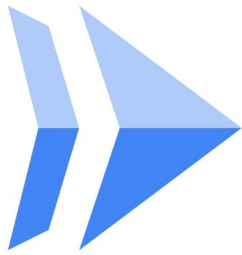
When you use Google Cloud, you can easily deploy microservices using either the managed container service, Google Kubernetes Engine, or the fully managed serverless offering, Cloud Run. Depending on the use case, Cloud SQL and other Google Cloud products and services can be readily integrated to support microservices architectures.

eg.



Google Kubernetes Engine

Secured and managed Kubernetes service with four-way auto scaling and multi-cluster support.



Cloud Run

Fully managed compute platform for deploying and scaling containerized applications quickly and securely.



Cloud SQL

Fully managed relational database service for MySQL, PostgreSQL, and SQL Server.



Anthos

Modernize existing applications and build cloud-native apps anywhere to promote agility and cost savings.

Cloud-native application development

Build, run, and operate cloud-native apps with Google Cloud. Embrace modern approaches like serverless, microservices, and containers. Quickly code, build, deploy, and manage without compromising security or quality.

Unlocking legacy applications using APIs

Extend the life of legacy applications, build modern services, and quickly deliver new experiences with Google's API management platform as an abstraction layer on top of existing services.

Challenges associated with microservices architecture

The microservices architecture comes with its fair share of challenges, from deployment to operation and maintenance. Some of these challenges are discussed below.

1. Inter-service communication

Although microservices can exist and function independently, they often need to interact and communicate with other microservices to fulfill certain demands or tasks. This necessitates maintaining a fully functional API serving as a communication channel between multiple services constituting the application.

2. Distributed logging

When different and independently operating microservices are deployed as part of an application, each of these services has a distinct logging mechanism. This results in large volumes of distributed log data that are unstructured and difficult to organize and maintain.

3. Transaction spanning

Distributed transactions refer to transactions that require the deployment and proper functioning of a series of microservices to run. This means that a transaction spans multiple microservices and databases and a small failure in just one will result in a transaction failure.

4. Cyclic dependencies between services

A **cyclic dependency** in a microservices architecture refers to the code dependency of two or more application services or modules. Cyclic dependencies can make it difficult to scale the application or independently deploy and manage microservices. They're also infamous for making code more complex to maintain. If they persist for long, decoupling becomes close to impossible.

Examples of companies that adopted microservices architecture

With recent advances in cloud technology, many big brands have now advocated moving from a monolithic to a microservices architecture for better functionality. Let's look at two such companies that have vastly improved their business by leveraging microservices.

1. Amazon

If you look at Amazon's retail website in 2001, it essentially worked as a single, monolithic application. The

lack of application flexibility made developers struggle while untangling dependencies when upgrading or scaling the services. As a result, Amazon struggled to meet the needs of its rapidly growing customer base.

To solve this issue, the Amazon development team split the large, monolithic application into smaller, independent services managed and handled by separate developer teams. Amazon's efforts eventually led to the creation of a highly decoupled, service-oriented architecture that we now refer to as microservices architecture. These changes made Amazon overcome all of its scalability and service outage problems and hit a market cap of \$1.7 million. The image below is a visual representation of the microservices architecture deployed by Amazon in 2008.

2. Netflix

Netflix started its movie streaming business in 2007. Within just a year, it started facing severe scalability issues and service outages. In 2008, Netflix failed to ship DVDs to customers for three consecutive days. This is when they decided to switch to a distributed cloud system with Amazon Web Services (AWS) as the cloud provider.

Later in 2009, Netflix began moving its application architecture from monolithic to microservices, and this process was finally completed in 2012.

This move to a microservices architecture enabled Netflix to overcome its scalability challenges and offer its services to millions around the world. In 2015, Netflix's API gateway successfully processed 2 billion API requests daily – thanks to a group of over 500 cloud-hosted microservices. The cost of streaming was also reduced,

which allowed Netflix to make significant financial gains. Netflix's application consisted of over 700 microservices by 2017, and the streaming service has grown exponentially over the past decade.

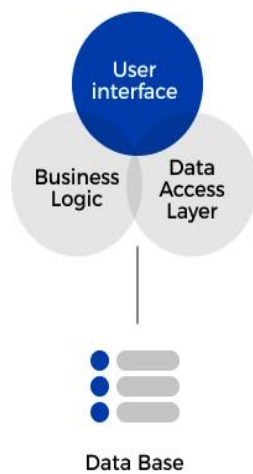
Switch from monolithic to microservices architecture

Microservices architecture is the most accepted and reliable approach to developing cloud applications today. It's highly backed due to the various advantages it brings to tech development teams in the form of shorter development times, improved flexibility, and greater reliability.

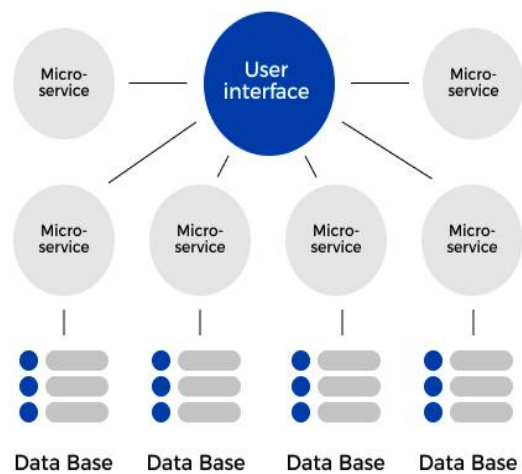
It is also unnecessary for a business with existing applications to build the microservices architecture from scratch. Several options are now available that allow converting web applications from monolithic to microservices.

If you are looking for ways to scale your organization's tech capabilities and overcome outages, poor performance, and geographic challenges, check out our solutions.

MONOLITHIC ARCHITECTURE



MICROSERVICE ARCHITECTURE



Advantages of microservices architecture

MSA supports modular applications where any single module in a system, such as a microservice, can be changed independently without affecting the other parts of the program and without creating unanticipated changes within other elements.

Modular programs are also more adaptable to iterative development processes and Agile practices compared to monolithic programs. They are also more scalable and can be tested individually due to loose coupling between the various components. Modules also communicate with each other, have their own databases and increase application startup speed.