Monolithic applications are more of a single complete package, having all the related needed components and services encapsulated in one package.

Monolithic System Limitations

As the application grows, so does the associated code base, which can overload your development environment each time it loads the application, reducing developer productivity.

Because the application has been packaged in one EAR/WAR, changing the technology stack of the application becomes a difficult task. With this kind of architecture, refactoring the code base becomes difficult because it’s hard to predict how it will impact application functionality.

If any single application function or component fails, then the entire application goes down. Imagine a web application with separate functions including payment, login, and history. If a particular function starts consuming more processing power, the entire application’s performance will be compromised.

Scaling monolithic applications such as the one described in the example can only be accomplished by deploying the same EAR/WAR packages in additional servers, known as horizontal scaling. Each copy of the application in additIonal servers will utilize the same amount of underlying resources, which is inefficient in its design.

Monolithic architecture impacts both the development and application deployment stage. As applications increase in size, it’s even more important that developers be able to break their applications down into smaller components. Because everything in the monolithic approach is tied together, developers cannot work independently to develop or deploy their own modules and must remain totally dependent on others, increasing overall development time.

Microservices

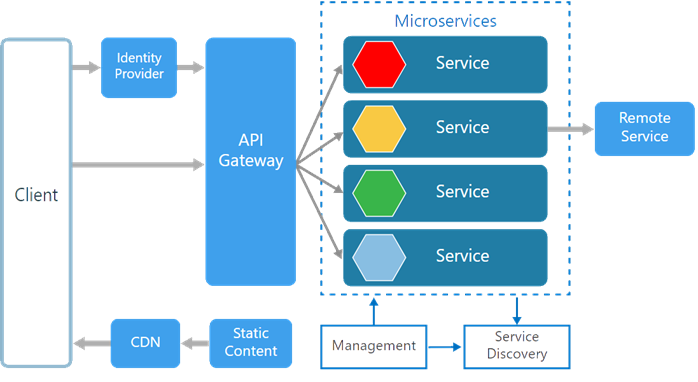
A microservice is an approach to create small services, each running in their own space that can communicate via messaging. These are independent services directly calling their own database.

Microservices are a way of breaking large software projects into loosely coupled modules, which communicate with each other through simple Application Programming Interfaces (APIs).

Microservices are deployed independently with their own database per service

In monolithic architecture, the database remains the same for all the functionalities even if an approach of service-oriented architecture is followed, whereas in microservices each service will have their own database.

## Microservices Architecture



There are various components in a microservices architecture apart from the microservices themselves.

**Management :** The Management take care of placement of services on nodes, checking for failures, rebalancing services across nodes in case of any failures.

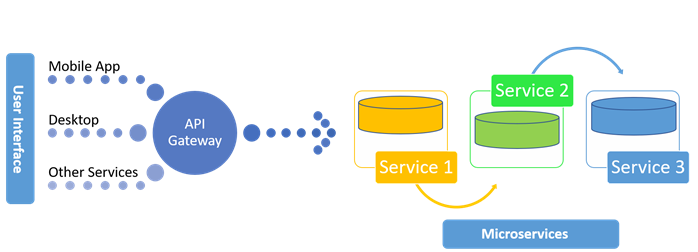
**Service Discovery :** Maintains a list of services and the nodes where each service are located, and also enables the service to look up to find the endpoint for a particular service.

**API Gateway :** The entry point for clients where all the calls from the client will be taken, analyze it and forward to appropriate services. In case some calls needed from multiple services API Gateway will aggregate and will return the aggregated result.

**CDN**. A content delivery network to serve static resources. For example, pages and web content in a distributed network.

**Static Content** The static resources like pages and web content.

Microservices are deployed independently with their own database per service so the underlying microservices look as shown in the following picture:



WHY SHOULD WE USE MICROSERVICES INSTEAD OF A MONOLITHIC APPROACH?

Microservices is an approach to develop small services that each run in its own process. We should develop microservices instead of one service (a monolithic approach) for a multitude of benefits, including:

* Microservices are smaller in size
* Microservices are easier to develop, deploy, and debug, because a fix only needs to be deployed onto the microservice with the bug, instead of across the board
* Microservices can be scaled quickly and can be reused among different projects
* Microservices work well with containers like Docker
* Microservices are independent of each other, meaning that if one of the microservices goes down, there is little risk of the full application shutting down

## The Advantages of Microservices

* Services can be written in different programming language and can be accessed by using any framework.
* Independently develop, deploy, redeploy, version and scale component services in seconds without compromising the integrity of an application
* Better fault isolation keeps other services to work even though on got failed.
* Zero downtime upgrades.
* Services can be of from different servers or even different datacenters.
* Interaction with other services in a well-defined protocol
* Monitor, capture, and report health diagnostics
* Reliable and self-healing
* Supports continuous integration and delivery
* Easy to transfer knowledge to the new team member
* Easy to integrate with third parties
* Software built as microservices can be broken down into multiple component services, so that each of these services can be deployed and then redeployed independently without compromising the integrity of an application. That means that microservice architecture gives developers the freedom to independently develop and deploy services.
* Better fault isolation; if one microservice fails, the others will continue to work.
* Code for different services can be written in different languages.
* Easy integration and automatic deployment; using open-source [continuous integration](https://apiumhub.com/tech-blog-barcelona/benefits-of-continuous-integration/) tools such as [Jenkins](https://apiumhub.com/tech-blog-barcelona/best-jenkins-plugins/), etc.
* The microservice architecture enables continuous delivery.
* Easy to understand since they represent a small piece of functionality, and easy to modify for developers, thus they can help a new team member become productive quickly.
* The code is organized around business capabilities.
* Scalability and reusability, as well as efficiency. Easy to scale and integrate with third-party services.
* Components can be spread across multiple servers or even multiple data centers.
* Work very well with containers, such as Docker.
* Complement cloud activities.
* Microservices simplify security monitoring because the various parts of an app are isolated. A security problem could happen in one section without affecting other areas of the project.
* Increase the autonomy of individual development teams within an organization, as ideas can be implemented and deployed without having to coordinate with a wider IT delivery function.

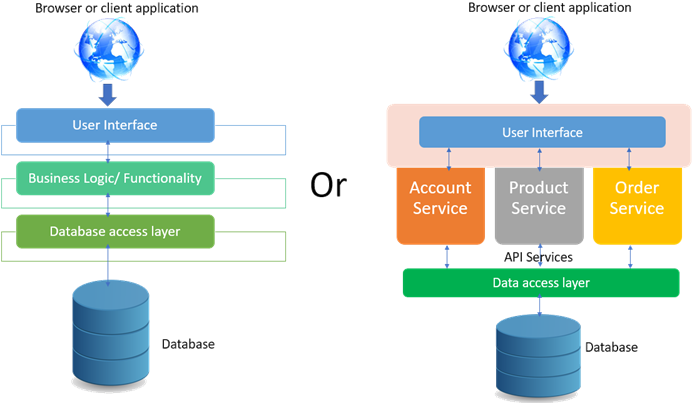
## The Disadvantages of Microservices

* The additional complexity for implementation of an inter-process communication mechanism between services.
* Writing automated tests involving multiple services is challenging and It can be difficult to create consistent testing environments.
* Requires high level of automation to manage multiple instances of different types of services in production.
* Everyone has to manage eventual consistency as maintaining string consistency becomes extremely difficult.
* Managing multiple databases and their transactions are difficult.
* Inter-process calls are slow.
* Debugging will become difficult.
* Complexity in DevOps.
* Production monitoring cost is higher.
* Formal documentation overhead.
* Lack of governance.

## Monolithic vs Microservices Architecture

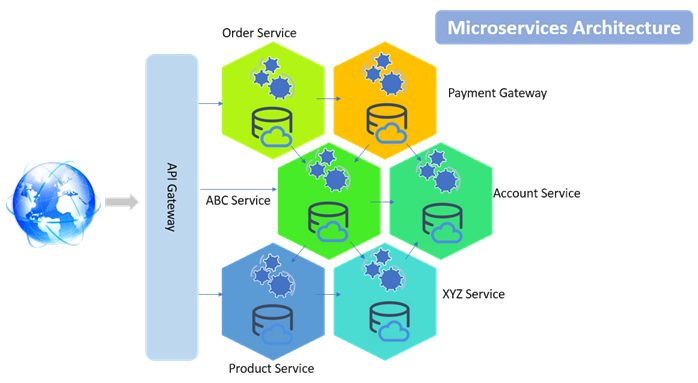
Monolithic applications are more of a single complete package, having all the related needed components and services encapsulated in one package.

The following is the diagrammatic representation of monolithic architecture whether packaged completely or service based.



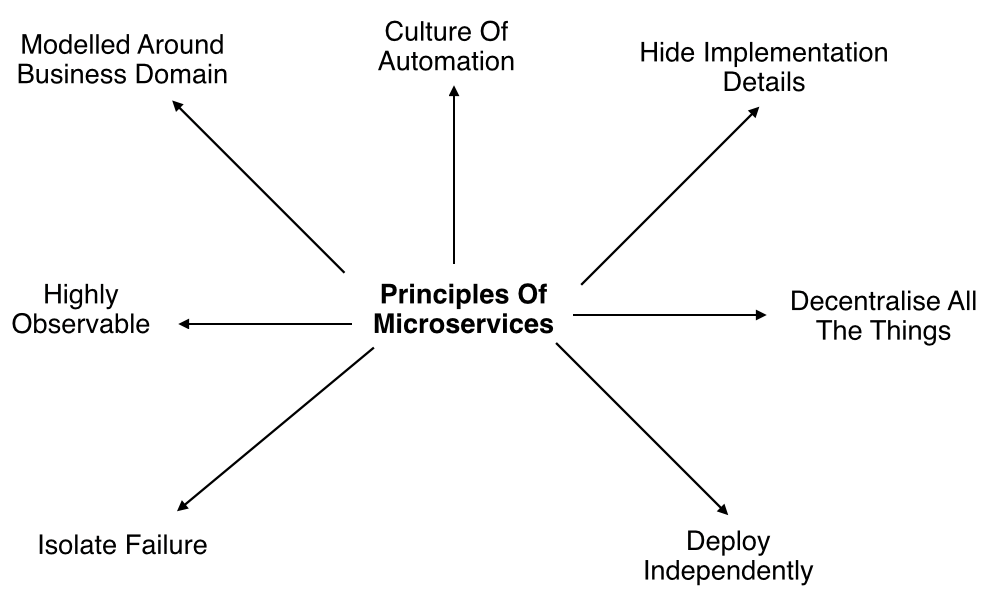
A microservice is an approach to create small services, each running in their own space that can communicate via messaging. These are independent services directly calling their own database.

The following is the diagrammatic representation of microservices architecture.



In monolithic architecture, the database remains the same for all the functionalities even if an approach of service-oriented architecture is followed, whereas in microservices each service will have their own database.

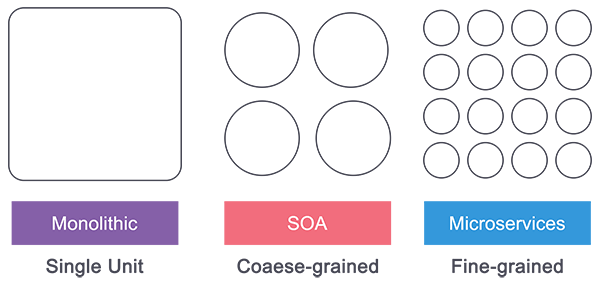
## Microservices Principles



* **Modelled around business domain :** Microservices architecture lets us separate system capability into different domains. Each domain will focus on one thing and its associated logic and can easily migrate independently to the next version, and also scale independently according to requirement.
* **Culture of Automation :** As we are building, testing, deploying and monitoring each service separately and there is an increase in the number of deployment units compared to monolithic architecture we should follow the culture of automation by designing it for continues integration and continuous delivery.
* **Hide implementation details :** Microservices should be architected in such a way that it won’t expose the internal details; neither technical implementation nor the business rules that drive it. This will reduce the coupling and helps to do changes and improvement without affecting the overall architecture.
* **Decentralization :** In traditional monolithic implementations, the software is designed to use a single database with different tables where microservices is designed in such a way to manage its own database.
* **Deploy Independently :** To enjoy the complete benefits of the architecture, microservices should be independently deployable. If you are failing to do so, check for any coupling in the application and solve it.
* **Failure Isolation :** The impact of a failure is less in microservice architecture compares to monolithic type as it will only affect that particular services and its associated while other services can keep running. The associated services should handle such scenarios when dependent is unresponsive or slow.
* **Highly Observable :** The services should collect as much information to analyze what is happening within each of them like log events and stats.

## Monolithic vs. SOA vs. Microservices

**Monolithic** architectures are the simplest form of architecture as it is having only one application layer that bundles together all the software components, and is hosted and delivered together. This type has been widely used by many small and mid-sized companies. The main challenge in this system is during scaling up as we need to duplicate the whole system including all the features of other machines which increases the cost. Also, the failure of one feature will affect the whole system making it unreliable.



**Service Oriented Architecture (SOA)** follows a coarse-grained structure where the features of an application are broken down into smaller components as services comprised of some tasks. This type of architecture allowed us to horizontally scale each service, and also more flexibility and performance at the cost of increasing the complexity of the architecture compared to the monolithic. Each service can be written in different languages and the communication between them can be done with the help of a middleware

**Microservices** has technically evolved out of SOA where those features are further broken down into tasks level services making it fine-grained architecture. While Service Oriented Architecture followed a centrally governed architecture where each component is controlled by a central middleware, in microservices it’s a decentralized governing system where components talk directly to each other and can be written in different programming languages and communicate without the help of any broker and are done with the help of REST API.