

# Microservices

For the Enterprise Agility and Scalability



# Table of contents

<b>Introduction</b>	<b>3</b>
<b>Traditional Monolithic Architecture Patterns Vs Modern Microservice</b>	<b>4</b>
<b>Advantages of a Microservices Architecture</b>	<b>6</b>
<b>Patterns on Microservice</b>	<b>8</b>
<b>Technology and Tooling</b>	<b>10</b>
<b>Raising On Agility</b>	<b>14</b>
<b>Conclusion</b>	<b>14</b>



## Introduction

Microservices is an architectural type that structures an application as a cluster of small autonomous services to develop modern applications.

It allows teams to focus on narrower domains and smaller units, aiding enterprise agility and scalability to sustain the surging market demands. Microservices have made the mark in the software architecture market, providing advantages such as improved fault tolerance that keeps larger applications from getting affected by failures of smaller modules.



# Traditional Monolithic Architecture Patterns vs Modern Microservice

Monolithic architecture pattern has traditionally been followed in most organizations. In a monolith approach, the complete application is developed as a single large module which should be deployed every time, even on minor changes. In contrast, in the case of microservices architecture, we need to deploy only the module which is affected by the changes.

The biggest pain point of the monolithic application is on every change cycle, the complete application should be verified, whereas, in case of the microservice we need to verify only the affected module.

In monolithic, the complete application should be developed using one specific

technology, but microservice is technology agnostic. We can develop every individual module in the application with different technologies.

Microservice allows to scale every individual module to the level that is needed for the business to perform but in case of monolithic, it is possible to only scale the whole application.

Microservice architecture is an ingenious way to create software quickly while keeping risks to a minimum. However, while beginning with the transition, the future-focused businesses must identify and acknowledge their application requirements to ensure agility, scalability, efficiency, and success.

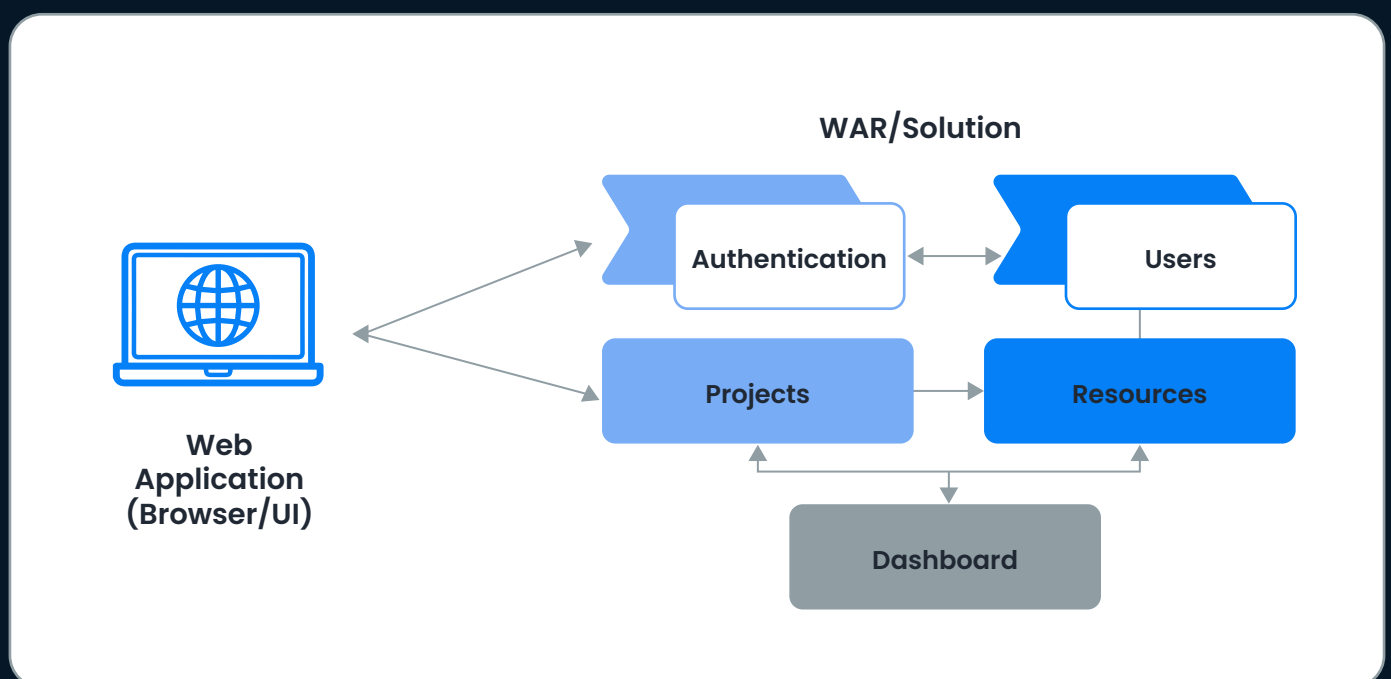


Diagram depicts a simple application that follows the monolithic approach

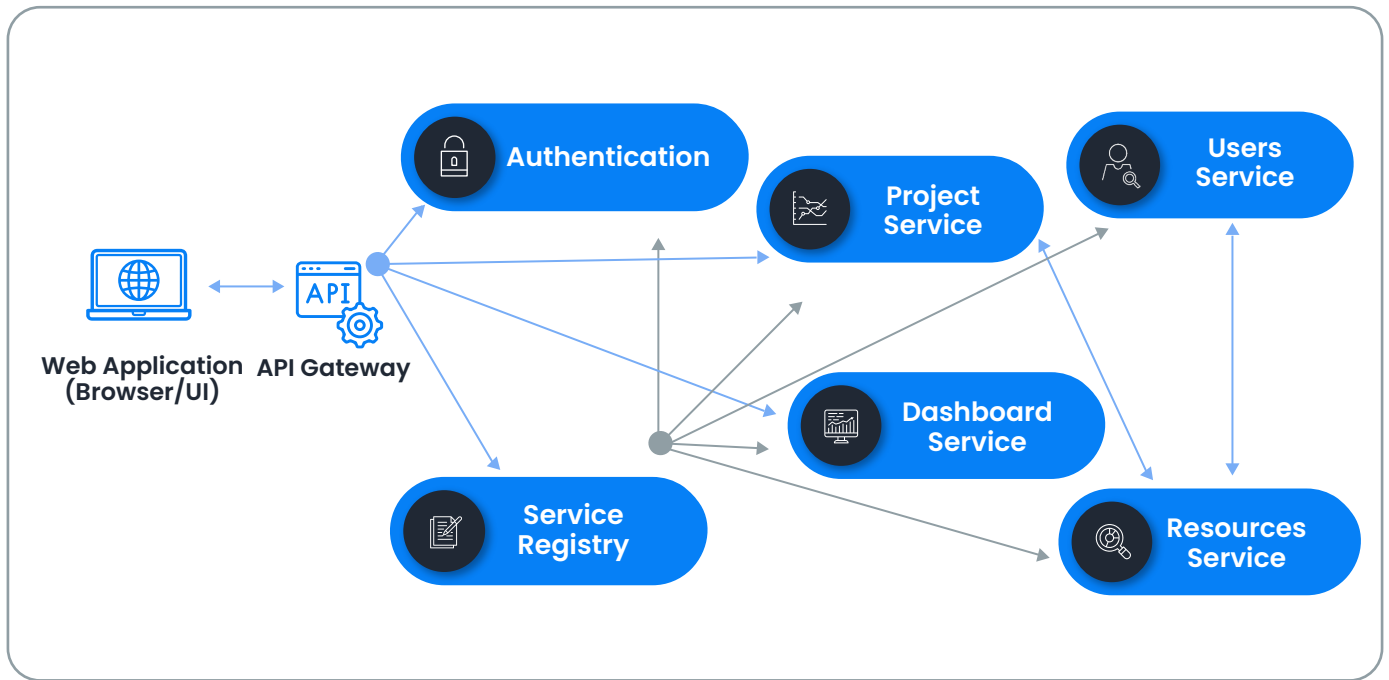


Diagram depicts the same application in microservice approach

“With larger, monolithic systems, there are fewer opportunities for people to step up and own something. With microservices, on the other hand, we have multiple autonomous codebases that will have their own independent lifecycles. Helping people step up by having them take ownership of individual services before accepting more responsibility can be a great way to help them achieve their own career goals, and at the same time lightens the load on whoever is in charge!”

- Sam Newman, Building Microservices

# Advantages of a Microservices Architecture

Microservices architecture brings a lot of advantages to modern applications. Let us discuss some of those briefly below



## Technology Agnostics

Microservice architecture pattern provides freedom to develop each module in the application with different technology which suits that module.



## Agility

Microservice pattern allows the team to focus only on the module they are responsible for and deliver much faster.



## Easy Deployment

Microservice allows deploying only the changes that reduce the deployment effort of deploying the whole application every time.



## Cloud Native

Microservices are more cloud native. It is easy to setup in any public or private cloud as it is mostly packed as containers and can be deployed easily.



## Scalability

Microservice architecture allows to scale any individual module to the level of performance that is needed by the business. For example, in an air ticket booking application, the flight search module is heavily used by the customers. Microservice architecture allows to develop the flight search module as a separate module and scale it to the level of performance needed to satisfy the business need.



## Better Fault Isolation

In case of any failure in any module, only the module specific functionality will not be available to the users, but users should continue to use the application with other functionalities which was not affected. For example, in the ecommerce application where the product review service is down, the users should be able to continue using the application's other functionalities like searching for the product, getting the product list, purchasing the product, etc., & users could not be able to see only the product review due to which there won't be a huge impact on the business in case of failure in any modules.



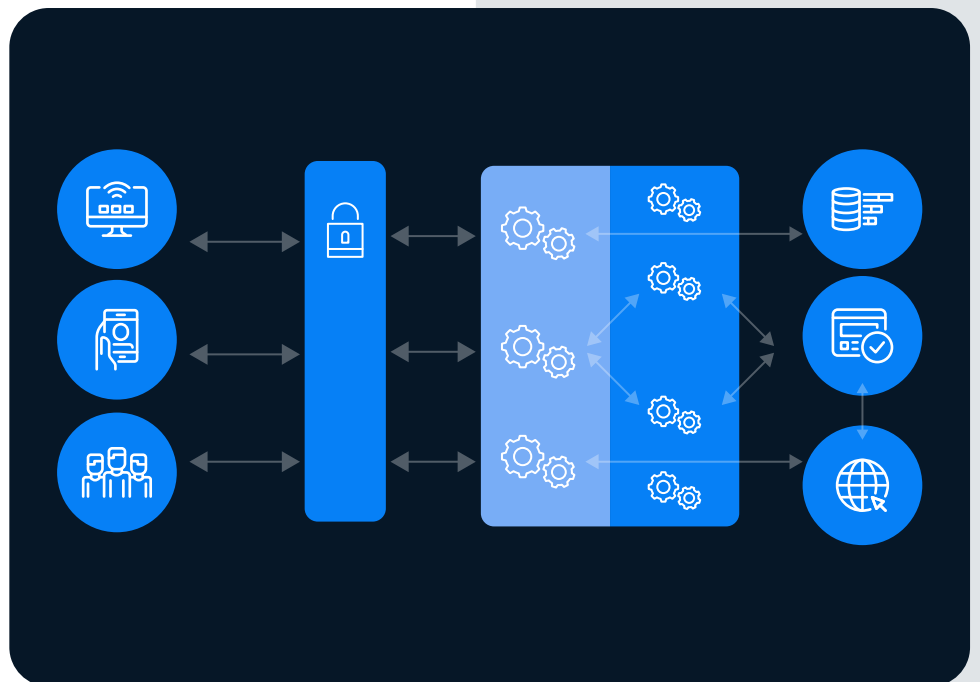
The global microservices architecture market size was valued at \$2,073 million in 2018, and is projected to reach \$8,073 million by 2026, registering a CAGR of 18.6% from 2019 to 2026.

**Global Microservices Architecture Market**

Opportunities and Forecast,

**Global Microservices Architecture Market is Expected to reach**

Growing at a **CAGR of 18.6%** (2019-2026)



# Patterns on Microservice

Microservice architecture provides various patterns for developing high performing and scalable enterprise applications.



## Decomposition Patterns

Decomposition patterns provide various approaches on how the application should be broken into multiple microservices based on the business requirement such as decompose by business capability, decompose by subdomain, self-contained service, & service per team.



## Refactoring Monolithic to Microservice

There are patterns that allow the existing monolithic application to refactor to multiple microservices. Some of the approaches for refactoring are strangler application, anti-corruption layer, etc.



## Integration Patterns

Integration patterns suggest how the microservice communication should happen inside the application and from the presentation layer and help us protect the microservices from any kind of vulnerabilities. In ACL digital, we analyze the customer requirement and based on that, we apply the right integration pattern for the smooth performance of the application. The various integration patterns are API gateway pattern, aggregator pattern, client-side UI composition pattern, etc.



## Database Patterns

As the microservices are more independent, it is very crucial to decide how the database should be organized for microservices. The database patterns suggest various ways in which we can solve the database requirement for the microservices. The various database patterns are database per service, shared database per service, command query responsibility segregation (CQRS), and finally, the saga pattern for maintaining the atomicity of the transaction.





## Observability Patterns

One of the primary disadvantages of microservices is that they are distributed and autonomous, so it is very difficult to address the common functionality. Observability patterns suggest the various approaches like centralized log aggregation, centralized metrics aggregation, distributed tracing, health check, etc. to solve this issue.



## Concern Patterns

When it comes to Microservices, there are various cross-cutting concerns like application configuration, service identity, service reliability, etc. All these concerns can be addressed respectively by using external configuration, service discovery, circuit breaker pattern, etc.





## Technology and Tooling

Various tools and technologies are available to implement and deploy an application using microservice architecture. We suggest you some of the best tools & technologies based on our experience in building cutting-edge enterprise applications catering to diverse customer requirements. You will be reading about some of them here.

### Container Orchestration

In case of microservices, container orchestration is the key requirement of teams aiming for rich performance and scalability. We suggest the below tools for container orchestration

#### Kubernetes

Kubernetes (aka K8s) is the open-source container orchestration platform used for automated software deployment, scaling, & management. It was originally developed by Google, but Cloud Native Computing Foundation maintains the project now.

#### Docker Swarm

Docker swarm is another open-source container orchestration platform used extensively in orchestrating the docker containers due to its easy-to-use, high security and scalability features.

#### CI/CD

Setting up the CI/CD pipeline for the microservice based applications allows each team to build and deploy services independently with no disruption or dependencies.



## Jenkins

Jenkins is the leading open-source automation server that helps automate various SDLC processes, such as building the source code to artifacts, testing, and deployment. It supports multiple build automation tools such as Apache Ant, Apache Maven, MS Build, etc. Jenkins also supports various source control tools such as Git, CVS, Subversion, etc., allowing to run the shell scripts and Windows batch commands.

## GitLab

It's the DevOps platform developed by GitLab Inc. GitLab allows to develop, secure, and operate software in a single application. It was originally developed in Ruby programming language & some modules are rewritten in the Go programming language.

## Service Discovery

In microservice architecture, the whole application is decomposed into multiple microservices so it is imperative that the communication between the microservices is seamless. Service registry and discovery are essential to discover every microservice for smooth communication. There are various tools available, but we recommend using Eureka!



## Circuit Breaker

As the application is broken into multiple smaller microservices, most of the requests need communication to multiple microservices to process the response, but when one of the services is down, it may result in the application getting stalled on request. To address this issue, various circuit breaker tools are available that help stop the application from getting stalled by canceling the request for service that is not available and responding to the caller with the available information. The circuit breakers that we recommend are Netflix Hystrix, and Spring Cloud circuit breaker.

## API Gateway

As the application is broken into multiple smaller microservices, it is very challenging to call multiple services to provide the response, the API Gateway provides a single-entry point for all clients. The API gateway also helps in providing API analytics, API monetization, API documentation, API limiting, Developer portal, API proxy, API aggregation, etc. There are various frameworks and platforms available for API gateway and we recommend using Kong API Gateway, Tyk API Gateway, and Ocelot.

## External Configuration

A service normally calls other services, and the service in turn calls databases. The endpoint URL or configuration properties might differ for each environment like Dev, QA, UAT, and Production. A shift in any of those properties might demand a re-build and re-deploy of the service, so it is necessary to avoid code modification for any configuration changes.

The ideal approach is to externalize all the configurations, including endpoint URLs and credentials, so the application should load them either at startup or on trigger.

Spring Cloud config server allows to externalize the properties to GitHub and load them as the environment properties. These can be accessed by the application either at startup or can be refreshed without a server restart by just triggers.

Alternatively, Azure App Configuration acts as centralized storage of configuration values and provides many additional features to make configuration management easier. Azure Key Vault is the service from Azure allows storing microservices configuration centrally. While Azure Key Vault may seem like a replacement for Azure App Configuration, it's an addition to it.

## Distributed Logging

In case of a microservice application, it consists of multiple instances of a service running on multiple machines, so the requests will be distributed across multiple service instances. Every individual service instance generates a log file in a standardized format. So how we trace the failure through logs when it is distributed across multiple services for any request is the key challenge.

To address this issue, we first need a centralized logging service that aggregates logs from each service instance so that the users can search and analyze the logs and also configure alerts triggered when there are certain errors spotted in the logs.

## ELK Stack

The ELK stands for three open-source projects—Elasticsearch, Logstash, & Kibana, which can be used to setup the centralized logging & error tracing.

Elasticsearch is a full-text search and analytics engine. Logstash is a log aggregator that collects the logs from multiple processes and stores the log into a centralized log store which is, Elasticsearch. And finally, Kibana provides a user interface, allowing users with the dashboard to visualize, search, and analyze the log data through various charts and graphs.

These are independent tools integrated by Elastic and it provides an end-to-end log analysis solution.



## Raising On Agility

Adopting scalable, agile, friendly, cloud native, containerized Microservices can help enterprise transform their legacy monolithic to shifts in software development, tools, techniques, and frameworks. By having smaller independent modules, any changes in your application can be rapidly developed and deployed with minimal effort.

Using an agile process like Scrum helps the time bound development deliver in a short span of time and this is the perfect fit for the microservice based application development.

## Conclusion

Satisfying the needs of any competitive enterprise is the never-ending process of adding new features while also maintaining existing functionality. Primary challenge for enterprises using Monolithic applications is that the cost of maintaining Monolithic has increased to the point that it no longer suffice skyrocketing enterprise requirements. It is very difficult to modernize the monolith application.

Moving forward, enterprises need an architecture that allows them to build and sustain the demanding competitiveness and efficiency that they seek for the minimum of the next 10 years.




Microservices is the hot iron that addresses the problems we are facing with other architecture. In ACL, we have built many successful projects and platform using the Microservices architectural style, which allows our customers to stay ahead and thrive in the dynamic business landscape. Moving forward, this will be the default style for building enterprise applications.

## References

- <https://sematext.com/guides/elk-stack/>
- <https://www.alliedmarketresearch.com/microservices-architecture-market#:~:text=The%20global%20microservices%20architecture%20market,18.6%25%20from%202019%20to%202026.>

ACL Digital is a design-led Digital Experience, Product Innovation, Engineering and Enterprise IT offerings leader. From strategy, to design, implementation and management we help accelerate innovation and transform businesses. ACL Digital is a part of ALTEN group, a leader in technology consulting and engineering services.

[business@acldigital.com](mailto:business@acldigital.com) | [www.acldigital.com](http://www.acldigital.com)

USA | UK | France | India   

Proprietary content. No content of this document can be reproduced without the prior written agreement of ACL Digital. All other company and product names may be trademarks of the respective companies with which they are associated.

