The Role of Microsoft .Net Core in Modernizing Legacy Applications

Arun K Gangula

arunkgangula@gmail.com

Abstract
The role of Microsoft .Net Core in modernizing legacy applications developed in conventional frameworks. This paper highlights the limitations and advantages of migrating applications to .Net core, which can improve performance, scalability, and cross-platform support. This paper delves into the challenges posed by conventional frameworks while emphasizing how migration to .Net Core can transform applications to deliver higher efficiency, adaptability, and longevity in today's fast-paced software environment.

Keywords: .Net Core, Azure DevOps, Model-View Controllers (MVC), Web Services, Windows Communication Foundation (WCF), Application Programming Interface (API)

1. INTRODUCTION

In the modern business world, companies are increasingly planning to upgrade their legacy applications to increase speed and gain a competitive edge. The most effective way to achieve this is by upgrading legacy Microsoft .Net framework applications to Microsoft .NET Core and its corresponding tools. Microsoft.NET Core is an open-source platform with a cross-platform solution that helps developers build high-performance apps that are compatible with the current infrastructure and are up to date with business trends [1].

2. LIMITATIONS OF .NET FRAMEWORK OVER .NET CORE:

Conventional frameworks lack Microsoft .NET Core advantages, such as enhanced performance, security, deployment, and cross-platform capabilities. Upgrading from legacy systems to new architecture enables organizations to improve their performance, adopt new technologies, automate business processes, and better address the needs of their customers and partners. When implemented well in a company's technology stack, these solutions give the organization confidence that they are on the right track, significantly when modernizing legacy applications using Microsoft .NET Core [2].

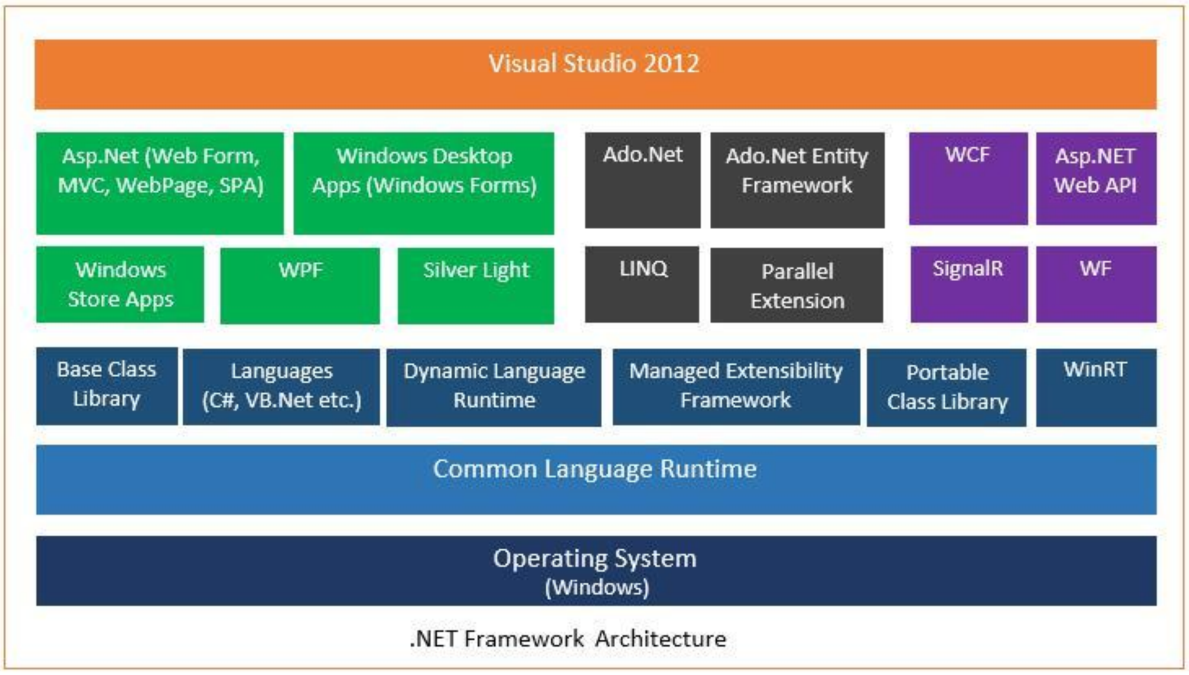

Fig. 1 .NET Framework Architecture [3]

2.1 Platform Dependency:

Many enterprises are now adopting cloud-first strategies where applications must be deployable on multiple operating systems. Traditional .Net framework legacy applications are platform-dependent; ASP.NET Web Forms, WCF Web Services, Windows Services, and VB6 are designed to work only on a Windows environment, limiting their cross-platform usability, which is crucial in our connected world.

2.2. Architecture:

.NET framework applications are monolithic, meaning all the functions are packed into one application and not separated. This architecture leads to poor scalability and agility because deploying or updating certain parts of the application individually is impossible. Due to their limitations, conventional frameworks cannot meet organizations' current and future business requirements.

2.3. Performance and User Interface:

Most legacy applications were Web Forms, Win-forms, etc., which lack responsiveness, a rich User Interface, and don’t support the latest browser technologies. .NET Core introduces Blazor, which leverages web Assembly and offers near-native performance for building modern web applications with rich user interfaces. Microsoft .Net Core is a modular and lightweight version of the .NET development platform that provides a runtime for building apps that can be run on Windows, Linux, and macOS.

3. Advantages of Migrating to .Net Core:

3.1. Architecture:

Microsoft .NET Core allows developers to decouple the UI from the logic and to develop microservices that can be deployed and managed independently. This modular design has several advantages that enhance scalability, agility, and reliability to meet different business needs. The main advantages of converting legacy applications to .NET Core include improved efficiency, security, and the system's architecture, which gives the user a secure feeling of technology.

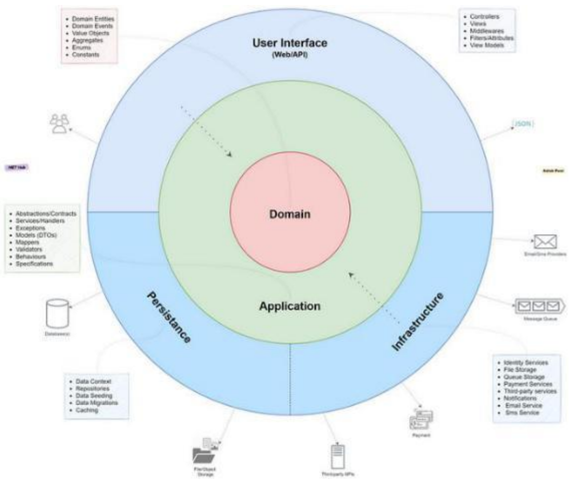

Fig. 2 Clean Architecture (Decoupled) [4]

Microservice architecture is also known as Microservices, a loosely coupled services architecture. Each service in this architecture focuses on a specific business function and communicates with other services through API's. At the same time, each application begins as a monolithic architecture and, over time, tends to become interconnected microservices.

Each microservice can be independently developed, tested, and deployed based on the specific requirement. Microservices support using diverse programming languages and frameworks for each service, allowing teams to choose the best technology stack.

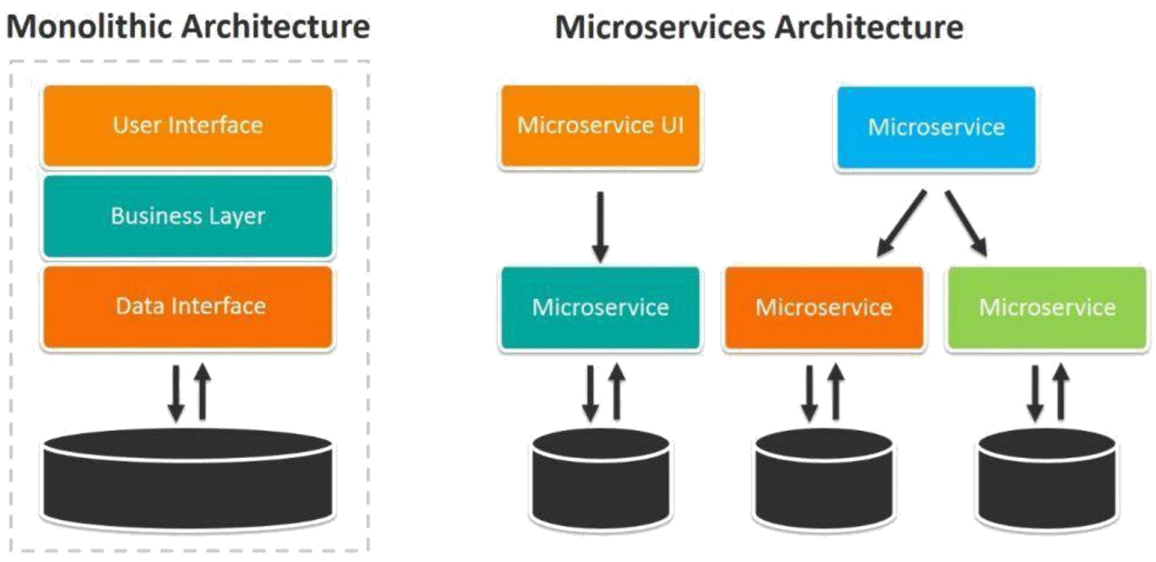

Fig. 3 Monolithic Architecture vs Microservice Architecture [5]

Microsoft .Net Core is excellent for cloud-Native development, which will optimize cost savings and increase scalability and availability.

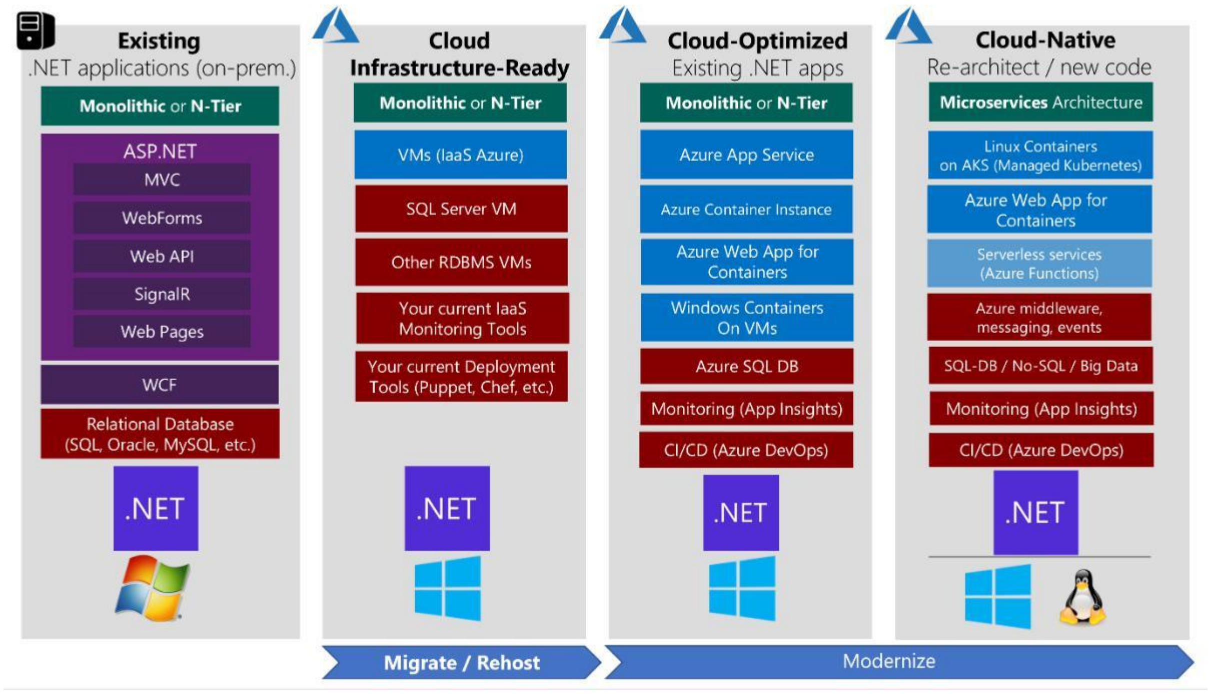
Existing
.NET applications (on-prem.)
Monolithic or N-Tier
ASP.NET
MVC
WebForms
Web API
SignalR
Web Pages
WCF
Relational Database (SQL, Oracle, MySQL, etc.)
.NET
Cloud Infrastructure-Ready
Monolithic or N-Tier
VMs (IaaS Azure)
SQL Server VM
Other RDBMS VMs
Your current IaaS Monitoring Tools
Your current Deployment Tools (Puppet, Chef, etc.)
.NET
Cloud-Optimized
Existing .NET apps
Monolithic or N-Tier
Azure App Service
Azure Container Instance
Azure Web App for Containers
Windows Containers On VMs
Azure SQL DB
Monitoring (App Insights)
CI/CD (Azure DevOps)
.NET
Cloud-Native
Re-architect / new code
Microservices Architecture
Linux Containers on AKS (Managed Kubernetes)
Azure Web App for Containers
Serverless services (Azure Functions)
Azure middleware, messaging, events
SQL-DB / No-SQL / Big Data
Monitoring (App Insights)
CI/CD (Azure DevOps)
.NET
Migrate / Rehost
Modernize

Fig. 4 Cloud Native Application Architecture [6]

3.2. Runtime Compilers & Memory Management:

The .NET Core uses Common Language Runtime (CLR) and the Virtual Machine (VM) to manage applications, improving performance by optimizing resource utilization and execution time. It also has a modular structure that enables developers to use reusable libraries and packages to reduce the time and money spent on the development process and ensure product quality. In addition, the modular approach enhances security; developers can selectively include only the necessary APIs and libraries to reduce the attack surface. Also, Core has a scalable application architecture suitable for high-traffic websites and manages peak loads without over-provisioning. When migrating legacy applications to.NET Core, ASP.NET Core should be used as the web server.

The new Kestrel web server will support asynchronous programming models. Combining Kestrel and ASP.Net Core will speed up application performance. Automatic memory management (Garbage collection) in .Net Core is not usually edited; unlike the traditional framework, the developer is responsible for allocating memory by creating instances [7] and eliminating programming errors and memory leaks.

3.3. Portability:

ASP.NET Core is a new web development tool that provides a single development experience for building web apps, APIs, and microservices. Optimized for API-driven development, supporting Open API (Swagger) for seamless API documentation and integration. Its modular approach and minimal system library dependencies make it easier to port existing apps to.NET Core with less code change. For example, applications developed with ASP.NET Web Forms can be ported to ASP.NET Core MVC or Blazor, better suited for developing web apps. Also, Core has built-in support for popular databases such as MS SQL Server, MySQL, and MongoDB, which are helpful in the movement of data and for growth during the transition.

3.4. Container Support:

The lightweight characteristic of containers shares the host machine operating system rather than having a separate operating system for each container, allowing applications to run the same on any infrastructure [8]. Unlike .NET Framework, .NET Core is optimized for containerized deployments using Docker and Kubernetes, making it a preferred choice for cloud-native applications.

3.5. Tools and resources:

Another great advantage of the transition to .NET Core is the availability of many tools and resources from Microsoft and other developers. .NET Core comes with development tools like Visual Studio Code, a powerful cross-platform IDE. It is also integrated with popular version control systems like Azure DevOps, GitHub, and GitLab for collaboration and code sharing. Microsoft Developer Network (MSDN) and GitHub [9] sample projects are valuable resources for developers.

4. CONCLUSION

By embracing .Net Core, legacy applications are modernized and aligned with modern software development trends such as Continuous Integration and Continuous Deployments, cloud computing, and microservices. Organizations can enhance application architecture, Cross-platform capabilities, lightweight architecture, and performance optimizations, making it an essential upgrade for enterprises seeking scalability and agility. It offers a foundation for building a robust, secure, and better-aligned tech stack. In a rapidly changing world, .Net Core is a stable yet evolving platform that enables organizations to grow and develop their strategies in the digital environment.

REFERENCES

[1] Philip Japike, Kevin Grossnicklaus and Ben Dewey, *Building Web Applications with .NET Core 2.1 and JavaScript*, 2nd ed., 2021

[2] de la Torre Sr, C., Hunter, S., Yuknewicz, P., Guthrie Sr, L., Zorrilla, U., & Valero, J. (2019). Modernize existing .NET applications with Azure Cloud and Windows Containers (2.2.).

[3] AAPNA Infotech site [Online] Available: https://www.aapnainfotech.com/microsoft-net-framework-4-5-architecture/- Microsoft .Net Framework Architecture

[4] (September 2021) The Medium site [Online] Available: https://medium.com/dotnet-hub/clean-architecture-with-dotnet-and-dotnet-core-aspnetcore-overview-introduction-getting-started-ec922e53bb97- Clean Architecture with .Net and .Net Core – Overview

[5] Jonathan Johnson, Laura Shiff March 2021. The BMC site [Online] is available: https://www.bmc.com/blogs/microservices-architecture/.

[6] de la Torre Sr, C., Hunter, S., Yuknewicz, P., Guthrie Sr, L., Zorrilla, U., & Valero, J. (2019). Modernize existing .NET applications with Azure Cloud and Windows Containers (2.2.).

[7] December 2018 The Dotnetcurry site [Online] Available: https://www.dotnetcurry.com/csharp/1471/garbage-collection-csharp-dotnet-core Garbage Collection in C#.

[8] (April 2021) The RedHat site [Online] Available: https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization -container-orchestration- What is containerization?

[9] The GitHub Site [Online] Available: https://github.com/- Software build and ship collaborative platform