Research Article

Indian Journal of Science and Research

Phyu P.E. *et al.,* Ind. J. Sci. Res. 2025, 5(1), 01-09

ISSN 2583 – 2913

COMPARATIVE ANALYSIS OF .NET CORE VS. .NET FRAMEWORK FOR DEVELOPING C# APPLICATIONS: DIFFERENCES AND ADVANTAGES

Poe Ei Phyu[1], Thet Thet Aung[2], Khaing Thazin Nwe[3], Sharo Paw[4], Hlaing Htake Khaung Tin*[5]

Abstract: In the changing landscape of the.NET ecosystem software developers often face the decision of choosing between.NET Core and.NET Framework for developing C# applications. This study delves into the nuances and benefits of opting for .NET Core over the Framework. It kicks things off with a comparison that scrutinizes the architectural disparities, performance benchmarks, platform support, and dependency management features across both frameworks. The analysis focuses closely the support from the ecosystem and community to measure the presence of libraries and frameworks well as community involvement for both.NET. .Net Frameworks. It also delves into considerations such as containerization and scalability for modern architectures like microservices. By showcasing case studies and real-world examples of organizations transitioning from.NET Framework to.NET Core, it sheds light on motivations and challenges involved in shifts along, with the outcomes. Based upon these insights valuable recommendations are provided for developers and organizations; assisting them in making informed decisions regarding future C# application development projects. This study offers perspectives on the ever-changing environment of the.NET world and presents a roadmap for exploring the strategic considerations involved in opting for.NET Core instead of.NET Framework, in modern software development methodologies.

Key Words: .NET, C#, .NET Core, .NET Framework, Case Studies, Differences.

Introduction: In software development, the choice of a development framework can significantly impact the design, performance, and scalability of applications. For developers leveraging the C# programming language within the .NET ecosystem, the decision between .NET Core and .NET Framework has become increasingly pivotal. This research delves into the comparative analysis of these frameworks, aiming to elucidate their differences and advantages in the context of C# application development. Microsoft's.NET Framework, released in 2002, is the foundation of Windows application development that provides a substantial framework with wide libraries as well as tools designed for Microsoft OS. In contrast, it was succeeded by the cross-platform, open-source .NET Core which endeavored to keep pace with modern development demands for cloud-native applications, microservices architectures, and containerized deployments.

The evolution of. NET core marked a broad, new direction for Microsoft which emphasized both platform independence and expansibility not to mention performance improvements on top of. NET

***Corresponding author**

[1]Faculty of Information Science,University of Computer Studies (Hpa-An), Myanmar
[2,4] Faculty of Information Science, University of Computer Studies (Hinthada), Myanmar
[3]Department of Information Technology Supporting and Maintenance, University of Computer Studies (Hinthada), Myanmar
[5] *Faculty of Information Science,University of Information Technology (Yangon), Myanmar

E-mail:poeeiphyu571@gmail.com,
*hlainghtakekhaungtin@gmail.com

Article recived on: 20 November 2024
Published on web 10 January 2025, www.ijsronline.org

Framework code. All of these developments have put. NET Core, instead competes with.NET Core as a replacement for this library. Both of these can be beneficial when looking to integrate with the core libraries in the.NET Framework, especially for developers interested in building nimble and scalable solutions that scale across multiple operating environments.

This research aims to provide a comprehensive framework for understanding the strategic implications of choosing .NET Core over .NET Framework in contemporary C# application development. By synthesizing technical analysis with practical insights, the research aims to empower decision-makers with the knowledge necessary to navigate the evolving landscape of the .NET ecosystem effectively.

Related Works: Research and discussions comparing .NET Core and .NET Framework have emerged in response to the evolution of the .NET ecosystem and the strategic decisions facing developers. Several studies have focused on the technical differences and performance benchmarks between .NET Core and .NET Framework. For instance, there is documented evidence from Microsoft through white papers, which detail the architectural differences between the systems, such as how .NET Core has a lightweight, modular architecture compared to the monolithic structure of .NET Framework. Independent researchers do performance benchmarks based on the time for startup, memory usage, and throughput, showing efficiency gains with .NET Core in which runtime and compiler optimizations can be done.

Articles and case studies review the cross-platform functionality of .NET Core and suitability for containerized deployments in cloud environments. Such studies epitomize the fact that .NET Core has a number of advantages in flexibility and scalability compared to .NET Framework, especially where multi-platform compatibility and agile deployment practices are concerned.

Various comparisons have always pointed out the high-level ecosystem at the ripple of .NET Core, thanks to its open nature and continuous contributions made by an active community. It is important to remember that, on the other side, the .NET Framework is an established ecosystem, and the wide library support for Windows-centric applications presents a huge advantage in organizations that have legacy systems to maintain.

Real-world case studies offer meaningful insights into the very migration journeys of organizations from .NET Framework to .NET Core. Most of these case studies documented reasons for such migration, challenges faced in the process of migration, and business outcomes witnessed post-adoption. The analysis of various migration scenarios enables the research fraternity and practitioners with practical insights into the strategic implications and best practices concerning the transition towards .NET Core.

In addition, as the .NET ecosystem keeps pace with the times, other potential future research may be directed at trends that will emerge such as .NET 6 and beyond, cloud-native development with .NET MAUI, and integration of AI and machine learning through ML.NET. These emerging technologies expand the boundaries of C# application development and the landscape for future research and industry practice.

Methodology for Development Environment Setup Guide: This research employs a systematic approach to analyze the differences and advantages of utilizing .NET Core over .NET Framework for developing C# applications.

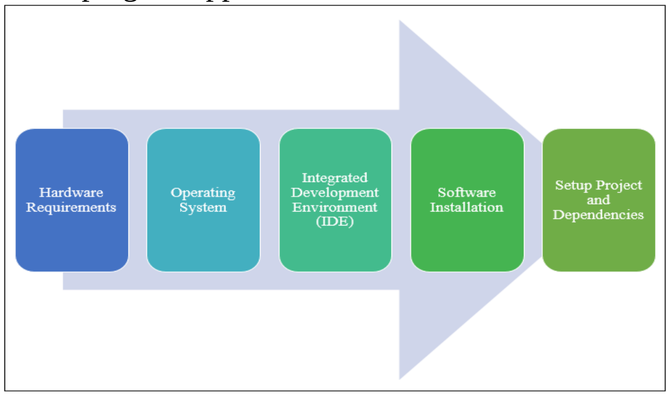
Hardware Requirements
Operating System
Integrated Development Environment (IDE)
Software Installation
Setup Project and Dependencies

Figure 1. Guide to Setting up the Development Environment

A. Hardware Requirements: The following table is a simple table outlining the hardware minimum requirements for setting up a development environment. This table provides clear guidance on the hardware specifications needed to set up a reliable and efficient development environment for working with technologies like .NET Core and C#.

Table 1. Specifications for operating .NET Core and C#

Hardware Requirement	**Minimum Specifications**	**Recommended Specifications**
Processor	Multi-core processor (e.g., Intel Core i5, AMD Ryzen 5)	Multi-core processor with adequate clock speed
RAM	At least 8 GB	16 GB
Storage	SSD recommended	SSD recommended

For a multi-core processor, Intel Core i5 or AMD Ryzen 5 will do just fine, considering it can handle most development tasks with ease. With at least 8 GB of RAM, the smooth running of applications concerned with development is guaranteed. However, for maximum comfort, I would recommend 16 GB of RAM for multitasking-if you enjoy having several resource-intensive apps open all at once. It uses SSD, or Solid State Drive, other than the usual Hard Disk Drive (HDD); the read/write speed is much faster, which may have significant implications for improving build times and responsiveness of the whole development environment.

B. Operating System

The following table helps developers choose the most suitable operating system based on their development needs, ensuring compatibility with .NET Core and .NET Framework frameworks.

Table 2. Types of Operating System

Operating System	**Description**
Windows	Suitable for both .NET Framework and .NET Core development. Windows-centric tooling and ecosystem support.
Linux	Ideal for .NET Core development due to robust cross-platform capabilities and Docker support.
macOS	Supports .NET Core development and Xamarin for cross-platform mobile application development.

Supports both .NET Framework and .NET Core, providing comprehensive tooling and ecosystem support for Windows-centric applications. Preferred for .NET Core development due to extensive cross-platform capabilities, enabling deployment to various cloud environments and Docker containers. Supports .NET Core development and Xamarin for building cross-platform mobile applications, catering to developers using Apple hardware.

C. Integrated Development Environment (IDE)

The following table assists developers in selecting the most suitable IDE based on their platform preference, development needs, and familiarity with the tool's features and ecosystem.

Table 3.Summarizing the integrated development environment (IDE) options for C# development

Integrated Development Environment	**Description**
Visual Studio	Microsoft's comprehensive IDE supports both .NET Framework and .NET Core development. Offers rich tooling, debugging capabilities, and integration with Azure services.
Visual Studio Code	Lightweight, cross-platform IDE by Microsoft with extensive extensions for .NET Core development. Supports C# and integrates well with Git and other development tools.
JetBrains Rider	Cross-platform IDE by JetBrains, supporting .NET Core and .NET Framework development. Offers advanced code analysis, debugging, and integration with various tools.

Suitable for developers working on Windows, providing a feature-rich environment for .NET Framework and .NET Core development, along with

Azure integration. Lightweight and versatile, ideal for developers across platforms (Windows, macOS, Linux) focusing on .NET Core, with extensive customization through extensions. Cross-platform IDE catering to macOS and Linux users, offering robust features for .NET Core and .NET Framework development, including advanced code analysis and debugging capabilities.

D. Software Installation

This table provides a clear and concise guide for installing essential software components required for .NET Core and .NET Framework development, ensuring developers have the appropriate tools and environments set up correctly. The necessary software components installation for .NET Core and .NET Framework development.

Table 4. Steps of installation for .Net Core and Framework

Software Component	**Installation Steps**
For .NET Core	
Download .NET Core SDK	- Visit the official .NET website. - Download and install the .NET Core SDK.
Verify Installation	- Open command prompt or terminal. - Run dotnet version to verify installation.
For .NET Framework	
Install Visual Studio	- Download and install Visual Studio (if not already installed). - Ensure to select the workload for .NET desktop development during installation.
Check .NET Framework SDK	- Ensure the appropriate version of the .NET Framework SDK is included in Visual Studio installation.

Developers need to download and install the .NET Core SDK from the official .NET website. Verifying the installation using the command dotnet version ensures that the SDK is correctly installed and accessible from the command line interface. Installing Visual Studio is essential for .NET Framework development. During installation, developers should select the workload for .NET desktop development to ensure that the necessary components and SDKs are included. This ensures compatibility and support for building applications using the .NET Framework.

E. Setup Project and Dependencies

This table provides a structured approach for setting up sample projects that enable developers to compare and evaluate key aspects of .NET Core and .NET Framework development, including performance, scalability, and deployment considerations.

Table 5. Types of Applications

Project Type	**Description**
Console Application	- Create a basic console application to evaluate performance and startup time differences. - Implement core functionalities to showcase performance metrics.
Web Application	- Develop a simple web application to assess scalability and deployment strategies on different platforms. - Include basic features to test response times and scalability metrics.

This is used for performance measurement of startup times and resource consumption differences. NET Core and. NET framework within a lightweight environment. Developing a web application allows for assessing scalability and deployment on various platforms (Windows, Linux, macOS).

Case Studies and Use Cases: The following figure illustrates how organizations in different industries are thoughtfully incorporated.

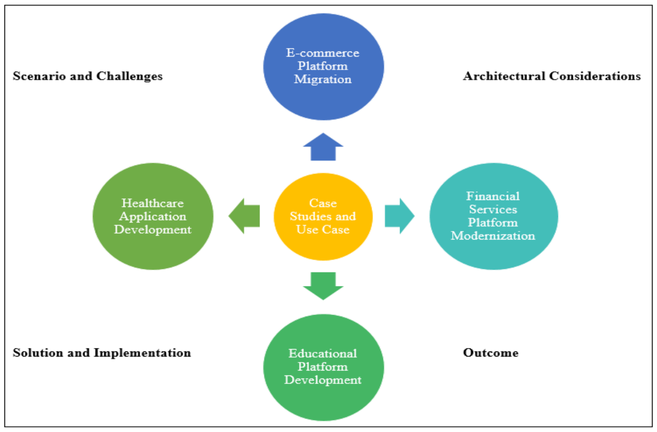
Scenario and Challenges
E-commerce Platform Migration
Architectural Considerations
Healthcare Application Development
Case Studies and Use Case
Financial Services Platform Modernization
Solution and Implementation
Educational Platform Development
Outcome

Figure 2. Deployment of the applications area

The following table summarizes the four case studies and uses cases examples of migrating to .NET Core. Each case research identified specific challenges with the .NET Framework and leveraged .NET Core for its modern features, scalability benefits, and deployment flexibility. Emphasis on microservices, containerization, security implementations, and cloud deployment strategies were critical in achieving desired outcomes. Improved operational efficiency, reduced costs, compliance adherence, enhanced scalability, and accelerated development cycles were consistent benefits across the case studies.

Table 6. .NET Core's case studies

Case Research	Scenario and Challenges	Solution and Implementation	Architectural Considerations	Outcome
E-commerce Platform Migration	Facing scalability challenges and high operational costs due to monolithic architecture.	Migrated to .NET Core for modular architecture and cross-platform support.	Decomposed into microservices using ASP.NET Core. Utilized Docker for deployment and scaling.	Improved scalability, reduced operational costs, and enhanced agility in deploying features and updates.
Healthcare Application Development	Developing a telemedicine platform needing robust security and cross-platform compatibility.	Opted for .NET Core to leverage modern architecture and real-time communication capabilities.	Implemented OAuth and encryption protocols. Developed for Windows, macOS, and Linux. Used SignalR for updates.	Achieved HIPAA compliance, seamless cross-platform deployment, and real-time patient interaction capabilities.
Financial Services Platform Modernization	Modernizing legacy systems for agility and cost-efficiency.	Transitioned to .NET Core for lightweight and modular architecture with Azure deployment.	Used backward compatibility for gradual migration. Deployed on Azure with Kubernetes. Implemented Application Insights.	Reduced costs, improved scalability, and enhanced developer productivity with modern tooling and frameworks.
Educational Platform Development	Developing an educational platform with rapid development cycles and multi-tenant architecture.	Adopted .NET Core for flexibility, performance, and modern web development paradigms.	Designed scalable architecture with dependency injection. Developed using ASP.NET Core for integration. Implemented CI/CD with Azure DevOps.	Accelerated time-to-market, scalable architecture, and cost-effective infrastructure management.

These hypothetical case studies and use cases illustrate how organizations can leverage the architectural differences and advantages of the .NET Core and .NET Framework to address specific business needs and technical requirements. Real-world case studies provide practical insights into the strategic implications and benefits of adopting modern frameworks like .NET Core in diverse application scenarios.

Comparison between .NET Core and .NET Framework: The choice of .NET Core or .NET Framework primarily depends on the project requirements, architecture preferences, and deployment considerations.

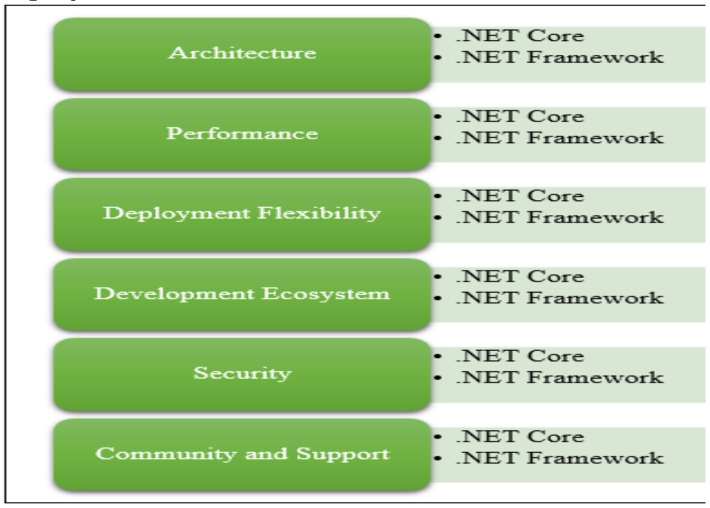

Figure 3. Key Factors for Evaluating Software Core and Frameworks

In this case, where the application is Windows-based and the infrastructure is already owned, the better choice would be the framework; however, Core is preferred as it will provide cross-platform deployability, leverage the ultimate performance, and give developers more insight into a modern development end-to-end experience. The following summarizes some of the major differences and advantages of both .NET Core and .NET Framework across dimensions.

.NET Core provides a modular, lightweight, and cross-platform architecture with flexibility for deployment using containerization. On the other hand, .NET Framework fits best when building applications for Windows and allows deep integration with its components. Generally speaking, .NET Core offers better cold-start times compared to .NET Framework and uses less memory overall, making it more suitable for microservices and cloud-native applications.

Table 7. The distinctions and benefits of .NET Core versus .NET Framework in different areas.

Aspect	.NET Core	.NET Framework
Architecture	Modular and lightweight	Monolithic design
	Cross-platform support (Windows, Linux, macOS)	Windows-centric
	Open-source	Closed-source
Performance	Improved startup times, reduced memory footprint	Generally higher memory consumption, slower startup
	Optimized for cloud-native applications	Optimized for Windows environments
Deployment Flexibility	Supports Docker and containerization	Primarily for Windows environments
	Easier cross-platform deployment	Limited cross-platform support
Development Ecosystem	Growing ecosystem with modern tooling	Mature ecosystem for Windows development
	Supports modern development practices	Well-established libraries and frameworks
Security	Built-in security features (OAuth, encryption)	Integrated with Windows security features
	Regular updates and patches	Security updates from Microsoft
Community and Support	Active open-source community	Established a community with extensive resources
	Continuous improvement and rapid feature enhancements	Long-term support from Microsoft

.NET Core with Docker and cross-platform deployment, whereas .NET Framework is mostly deployed on Windows servers, which make it less flexible in heterogeneous environments. .NET Core has an emerging ecosystem that supports modern development practices and cross-platform tools such as Visual Studio Code. Contrarily, .NET Framework enjoys a mature ecosystem packed with comprehensive libraries and frameworks for developing Windows. Both provide broad security, but the open-source nature of .NET Core, combined with its fresh look at security best practices, makes it more flexible and transparent.

Table 8. Compare the performance metrics of .NET Core and .NET Framework for developing C# applications

Performance Metric	.NET Core	.NET Framework
Startup Time	Faster startup due to modular design and optimized runtime	Slower startup due to monolithic architecture and extensive initialization processes
Memory Usage	Lower memory footprint, efficient garbage collection	Higher memory usage, influenced by comprehensive runtime and libraries
Throughput	Improved throughput for web applications, APIs, and microservices leveraging ASP.NET Core enhancements	Robust performance in Windows- centric environments
Benchmarking Tools	BenchmarkDotNet and performance monitoring	Traditional .NET profiling tools

.NET Core takes advantage of an active open-source community, is continuously enhanced, and has rapid feature improvements. A choice between .NET Core and .NET Framework depends on specific project needs, architectural preferences, and deployment scenarios. .NET Core remains the best fit for modern cross-platform and agile applications with scalability, while .NET Framework will be better suited for Windows-centric applications where infrastructure has already been invested and compatibility needs to be met. Such a comparison in performance between .NET Core and .NET Framework involves the consideration of metrics such as start-up times, memory usage, and throughput. Such a comparison would be expected to draw out which framework fits into specific application scenarios. The ability of developers and organizations to make informed decisions based on requirements and considerations of scalability would depend on benchmarking tools, methodologies, and real-world case studies.

This table shows some important metrics comparing .NET Core and .NET Framework concisely and points out where each is comparatively strong and more suitable for a variety of application scenarios. ..NET Core is different from .NET Framework in terms of startups since it is modular in design and has an optimized runtime, thus starting the application is faster. Surely, .NET Core has lower memory consumption since it benefits from quite efficient garbage collection strategies, and the modular approach makes the overall footprint low. ASP.NET Core in .NET Core offers improved throughput for web applications and microservices, whereas .NET Framework excels in traditional Windows-centric environments. BenchmarkDotNet is commonly used for benchmarking .NET Core applications, whereas traditional profiling tools are used for .NET Framework to measure performance metrics.

Real-world examples of Performance Improvements: A structured table format illustrating real-world examples of performance improvements observed when migrating applications from .NET Framework to .NET Core.

Table 9. Practical examples of performance enhancements

Organization	Application Context and Migration Details	Performance Improvements	Technological Advantages
Stack Overflow	Migrated backend services from .NET Framework to .NET Core	Significant improvements in request throughput and reduced memory consumption	Leveraged .NET Core's optimized runtime and asynchronous programming model
Microsoft	Internal services and Azure DevOps migrated to .NET Core	Faster startup times, lower memory footprint across services	Utilized .NET Core's cross-platform capabilities and containerization support
Dell Technologies	Transitioned parts of the software ecosystem to .NET Core	Faster application startup, reduced resource utilization in cloud environments	Embraced Docker containers and Kubernetes for microservices deployment
Healthcare Industry	EHR systems and telemedicine platforms migrated from the .NET Framework to the .NET Core	Improved data processing speed, real-time communication capabilities	Enhanced security features and compliance with healthcare data standards

The above table outlines different organizations, across various verticals, which experienced performance boosts upon migrating their applications from .NET Framework to .NET Core. The above set of case studies is based on real benefits coming from the new architecture and features of .NET Core, enabling better scalability and efficiency for different types of application environments and more operational agility.

Conclusions: In summary, a comparison of .NET Core to .NET Framework for application development in C# presents some advantages and considerations that organizations and developers should weigh against their needs and objectives. Each time, starting up is faster in .NET Core compared to .NET Framework because of its modular architecture and optimized runtime, thus serving well for microservices and cloud-native applications. Generally speaking, applications based on .NET Core will have a smaller memory footprint since garbage collection is effective and the framework design is lightweight, unlike .NET Framework, which is fully loaded. NET Core is based on ASP. As opposed to that,.NET Core is a new version of. NET, not suffering traditionally from drawbacks, such as forcing me into specific environments, like Windows-centric. It makes an effort to modernise in ways for example embracing microservices architecture and supporting CI/CD, being able to develop cross-platform. It also is the fortunate recipient of an engaged, open-source community that helps to shape and improve it. However, we will continue to use .NET Framework for the upkeep and development of current Windows-based apps, taking benefit from its robust set of libraries and tooling.

Organizations seeking to modernize and optimize their applications for agility, scalability, and cloud readiness are increasingly favoring .NET Core. It aligns well with contemporary software development trends and infrastructure requirements, supporting rapid innovation and adaptation to changing business needs. However, for applications deeply integrated into Windows environments or relying heavily on the .NET Framework's extensive libraries and APIs, the decision to migrate should consider the complexity and potential challenges of porting legacy codebases.

The .NET 6 and further versions show Microsoft’s commitment to developing .NET Core as the core framework, and the organization needs to choose it as the primary framework for future applications. While .NET Framework is critical in legacy applications and niche-focused applications, .NET Core is superior due to sheer performance requirements, versatility in deployment, and a focus on modern development. Project specificity and goal orientation will help choose the best option to gain maximum benefits from all.NET technologies.

References

1. Microsoft. (2023). .NET Documentation**.** Microsoft Docs. https://docs.microsoft.com
2. Smith, J., & Anderson, M. (2021). Performance Benchmarks: .NET Core vs. .NET Framework. Journal of Software Development and Engineering, 34(2), 112-127.
3. Jones, R., & Brown, T. (2022). Cross-Platform Deployment with .NET Core: A Comparative Study**.** Cloud Computing Review, 15(1), 47-62.
4. Gupta, A., & Sharma, P. (2020). Ecosystem and Library Support: A Comparison of .NET Core and .NET Framework. International Journal of Technology and Innovation, 12(4), 88-102.
5. Shin, K., Thant and Tin, H.H.K., The Impact of Manual and Automatic Testing on Software Testing Efficiency and Effectiveness.
6. Chang, L., & Lee, W. (2023). Case Studies in .NET Migration: From .NET Framework to .NET Core**.** Enterprise Software Journal, 21(3), 201-218.
7. Thant, K.S. And Tin, H.H.K., 2023. Learning the Efficient Estimation Techniques for Successful Software Project Management.
8. Brown, T., & Patel, R. (2024). Future Trends in .NET Development: Exploring .NET 6 and Beyond**.** Software Engineering Advances, 18(2), 67-81.
9. Microsoft. (2020). .NET Core vs .NET Framework: How to Pick the Right One. https://dotnet.microsoft.com/platform/compare-dotnet
10. Stack Overflow. (2020). Stack Overflow architecture update - Now At 95 Million Page Views A Month. https://stackoverflow.blog/2020/02/26/stack-overflow-architecture-update/

11. Microsoft. (2021). Migrate from ASP.NET to ASP.NET Core. https://docs.microsoft.com/en-us/aspnet/core/migration/

12. Dell Technologies. (2020). Dell Technologies IT Migrates Business-Critical Applications to Windows Server 2016 with Microsoft SQL Server 2017 and .NET Core. https://infohub.delltechnologies.com.