International Journal of Multiphysics
Volume 10, No. 4, 2016
ISSN: 1750-9548

Modernizing Middleware for Enhanced Cloud Compatibility and Performance

Srinivasa Subramanyam Katreddy,

Technical Architect,

Prasadampadu,Vijayawada,Andhra Pradesh - 521108

srinivasa.katreddy@gmail.com

Abstract:
Middleware systems form the backbone of cloud-based applications, yet many existing solutions lack the adaptability required for modern cloud environments. This paper introduces an innovative approach to middleware modernization, focusing on enabling compatibility with scalable cloud infrastructures. By employing containerization, service-oriented architecture (SOA) principles, and automated deployment pipelines, the framework enhances middleware efficiency while ensuring seamless integration with diverse cloud platforms. Performance evaluations reveal marked improvements in processing speed, system scalability, and fault tolerance. These advancements provide a roadmap for businesses aiming to modernize their middleware for future-ready cloud operations.

Keywords: Middleware Modernization, Cloud Infrastructure, Automation, SOA, Cloud Compatibility.

1. INTRODUCTION

Cloud application modernization

In order to get the most of cloud technologies and services, it is necessary to update and optimize older apps. This process is known as cloud app modernization. Application performance, scalability, and maintainability can be enhanced through rearchitecting, refactoring, and rehosting.

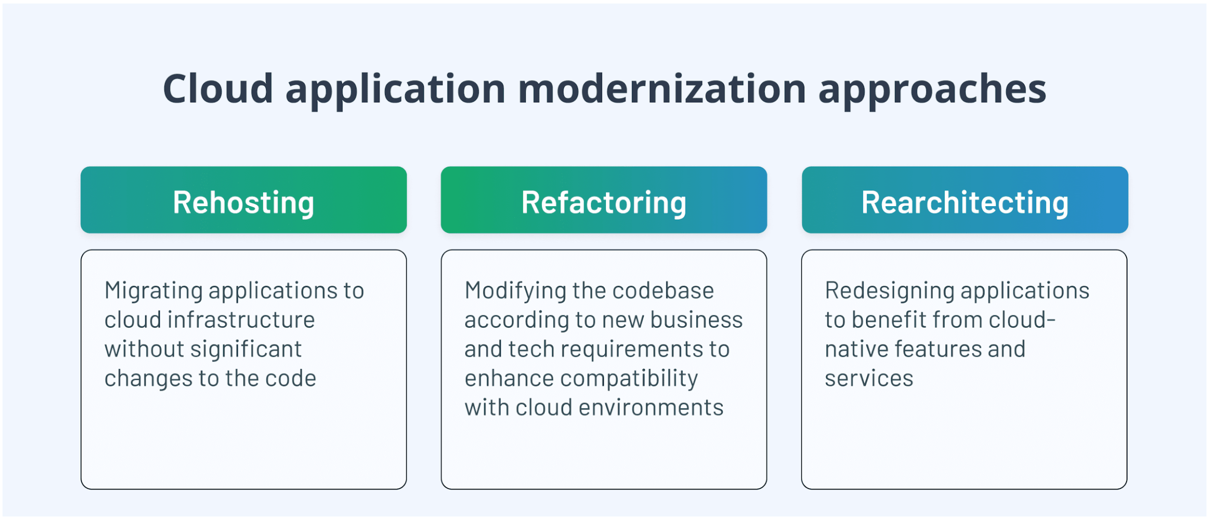

Cloud application modernization approaches
Rehosting
Refactoring
Rearchitecting
Migrating applications to cloud infrastructure without significant changes to the code
Modifying the codebase according to new business and tech requirements to enhance compatibility with cloud environments
Redesigning applications to benefit from cloud-native features and services

Fig 1: **Cloud application modernization**

Modern cloud environments shown in figure 1 provide enterprises with cost-efficiency, robustness, and flexibility; organizations can take advantage of these benefits by moving to cloud-native architectures [1]. We can look to the booming application modernization market as evidence that companies are making efforts to better match their technology stack with present and future needs. Forecasts include a 16.7 percent compound annual growth rate (CAGR) from 2016–2030, from an initial valuation of $17.80 billion in 2023.

Types of cloud modernization

1. Modernizing on-premises applications for the cloud

The goal of modernizing on-premises programs is to make full use of cloud capabilities by a complete redesign of existing applications. To start, there is an evaluation and planning phase where the application is evaluated to find out what has to be improved in order to be ready for the cloud. After then, the program is restructured. To make it work better on cloud infrastructure, its codebase should be tuned for microservices rather than monolithic architectures. The next step in improving functionality is to incorporate cloud-native services like databases, artificial intelligence, machine learning, and increased security measures [2]. Moving the updated app to the cloud is the last step in making sure it works well there.

To illustrate the requirement of restructuring code to operate with cloud databases and incorporating AI for predictive analytics, consider a retail organization that has a traditional inventory management system and wants to modernize it. By transferring to the cloud, their system will be able to scale up or down depending on the demand, and they will also have access to real-time inventory data.

2. Optimizing existing cloud applications

Existing cloud-hosted applications that aren't optimized yet require more modernization to take advantage of all the cloud has to offer. Performance tuning is the process of enhancing the speed and reliability of a program by modifying its code and configuration. You can improve scalability and maintainability by using cloud-native designs like containerization, serverless computing, or microservices. Allocating resources, lowering costs, and increasing operational efficiency are all attainable goals with the use of cloud tools.

One way to cut infrastructure costs is to replatform an application to the cloud. Further optimization is achieved by the transfer to microservices, which restructures the platform and allows features like user management and video streaming to scale independently. Both performance and operational costs are enhanced as a result. Security is the last consideration. The following cloud-native methods and tools can make it better:

- Real-time monitoring
- Securing the compute layer
- Applying secure storage and encryption

3. Modernizing legacy applications on-premises

When it comes to legacy apps that are still on-premises for reasons like security or compliance, modernization is all about incorporating cloud-native practices into that on-premises environment.

To guarantee consistent operation across many settings, containerization is utilized to package the application and its dependencies into containers. To get cloud-like features while retaining data in-house, businesses are turning to on-premises cloud platforms like private clouds or hybrid cloud environments. To improve operational efficiency, automation and orchestration solutions are used for automated scaling, management, and deployment. By taking this route, older programs can still meet certain operational needs while making use of newer technologies. Assume for a moment that a bank is subject to stringent regulatory restrictions; to comply, it decides to containerize and deploy a private cloud to upgrade its aging transaction processing software. Hence, businesses are able to enhance resource management, scalability, and compliance standards without transferring sensitive data to a third party.

Understanding the difference between cloud modernization vs migration
There is a tendency to use the phrases "cloud modernization" and "migration services" interchangeably due to the seeming lack of differentiation between the two. But they're talking about different procedures with different goals and results. The goal of cloud modernization is to adapt existing apps so that they can run more efficiently on the cloud. Microservices architectures, serverless computing, and the integration of advanced cloud services like AI and ML are all possible steps in this process. In order for systems to adjust to shifting business needs and market circumstances, it is necessary to improve application performance, scalability, and resilience. Transitioning from on-premises infrastructure to cloud environments involves migrating apps, data, and workloads. Several ways exist for accomplishing this, including:

"Lift and Shift" rehosting involves migrating apps to the cloud with little to no code changes

Refactoring – adjusting the program's source code to make complete use of capabilities built within the cloud

Modernization centers on reengineering applications to make the most of cloud technologies, whereas migration focuses on moving current systems to the cloud.

Benefits and challenges of cloud app modernization
To be impartial, we must go beyond the bright side of cloud app modernization. Consequently, we will examine the process's advantages and disadvantages in greater detail.

Benefits of cloud migration
- **Increased scalability**

With cloud computing, you can increase or decrease the amount of available resources according to your needs. Applications may handle different workloads without a major infrastructure change thanks to this. For most business-to-consumer platforms and expanding companies, that is the typical use case.
- **Cost efficiency**

The costs of running on-premises servers and data centers can be drastically cut by moving to the cloud. Rather, businesses only pay for the resources they actually utilize, turning capital expenditures into operating expenses. Organizations can save as much as 66% on computing, networking, and storage expenses by moving their operations to the cloud, says Enterprise Strategy Group. This is a major perk for well-established businesses looking to cut expenses.

- **Improved performance**

The resources and networking capabilities offered by cloud providers are top-notch. Both application performance and latency are improved by this.

- **Enhanced security**

Prominent cloud providers put a lot of resources on security, giving their customers robust protections that many businesses could struggle to set up in-house. Some examples of these measures are constant security monitoring, powerful firewalls, and data encryption. More robust security measures are available on Google Cloud, Microsoft Azure, and Amazon Web Services (AWS), all of which are crucial for industries like banking and insurance.

- **Disaster recovery and backup**

Data availability and integrity are guaranteed by cloud providers' integrated backup and disaster recovery systems, which can be accessed in the event of an unforeseen breakdown or disaster. Even for long-standing companies, this is frequently paramount.
- **Access to advanced technologies**

Organizations may innovate and remain competitive with the help of cloud platforms, which offer access to cutting-edge technologies like big data analytics, artificial intelligence, and machine learning. This is one of the most important goals for forward-thinking companies to achieve.

Challenges of cloud migration

- **Data security and privacy**

There are additional security risks associated with moving data to the cloud. Compliance with applicable privacy rules and regulations, including GDPR and HIPAA, and the encryption of data at rest and in transit are essential for organizations.

- **Data governance**

The security, usefulness, and integrity of data stored in the cloud depend on strong data governance practices. It could be more difficult to apply governance policies when using legacy data formats that aren't designed for cloud environments.

- **Legacy system compatibility**

Data corruption or loss during transfer could occur if legacy systems are not completely compatible with current cloud settings. This usually necessitates the use of specialist middleware or substantial reworking of the program.

- **Service disruption**

A significant problem during migrating is guaranteeing service continuity. Staggered migrations or interim redundant systems are two possible approaches to a well-planned strategy that can reduce downtimes.

- **Vendor lock-in**

Vendor lock-in occurs when an organization depends on the tools and services offered by just one cloud provider, making future provider switches difficult and expensive. The infrastructure's future-proofing and adaptability may be compromised as a result.

- **Scalability and performance**

Even while cloud environments can grow with your business, it's still important to pick a solution that can manage your unique data quantities and processing demands. Poor performance or excessive expenses can result from auto-scaling features that are not implemented correctly.

- **Testing**

It is essential to do comprehensive testing after migration to guarantee that apps work properly in the new setting. This requires checking for things like performance, security, and functionality.

- **Change management**

There is usually pushback from employees when moving to the cloud because it is a major change. To handle the changeover well, good communication and training are required.

- **Data integrity**

It is equally important to keep data intact while migrating. Before, during, and after a data migration, organizations must verify that no data is corrupted or lost.

- **Time and resource commitment**

Moving to the cloud can be a daunting task that demands a lot of time and energy. To avoid budget and schedule overruns, organizations should plan ahead and collaborate with seasoned engineers.
Cloud migration strategies should always be in line with the organization's long-term business goals, even though the pros and cons could vary from case to case. By doing so, conflicts can be avoided and the company's general goals can be advanced.

2. LITERATURE REVIEW

Future service definitions are being shaped and reshaped by the emergence of the Internet of Things (IoT) idea. Creation of various forms of communication networks based on collections of real objects, or "things," is the basic principle underlying this concept. Connectivity to the internet, software, sensors, and electronic chips allow IoT objects to gather data about their surroundings or change it at the push of a button. The Internet of Things (IoT) integrates data from the physical world with computational processing to improve accuracy, efficiency, and cost-effectiveness. The embedded computing systems allow each thing to be uniquely identified and allow them to connect with one another through the internet infrastructure. There has been a meteoric rise in the number of smart devices that are both embedded and linked. In 2020, there will be over 50 billion objects in the IoT market, according to Cisco IBSG [3].

An example of an IoT translation is "Semantic Oriented," which is similar to "Network-Oriented" or "Object-Oriented" (the discussed ones). As a result of divergent perspectives among stakeholders, as well as among vendors and IT specialists, various visions of this technology have arisen. "a global network of interconnected objects uniquely addressable based on standard communication protocols" is the literal definition of the Internet of Things. As an additional definition, the Internet of Things (IoT) is a network that enables connectivity "anytime, anyplace for any connected smart devices," according to the International Telecommunication Union (ITU). Cloud computing and the Internet of Things have been the subject of an ever-expanding corpus of research. The majority of these studies aimed to develop new and advanced technologies by combining various fields. For instance, CoT was defined and its importance was discussed in [4].

Data management, privacy and security, resource allocation, and identity management are some of the other important topics covered when it comes to integrating the Internet of Things with cloud computing. The writers of another piece stressed the importance of transitioning to CoT and how it can facilitate the efficient implementation of smart environments. Privacy, data security, and consumer law are the key issues. Utilizing cloud services allows CoT to take advantage of system performance, however transmitting large data or control packets can hurt this system and reduce its efficiency. Requesting basic services or temporarily keeping data in the cloud are two examples of when it would not be reasonable to exchange data between the Internet of Things and the cloud. In light of this, [5] introduced a smart gateway that can evaluate requests, determine if an answer is available locally, or even forward them to the cloud for processing.

Concerning middlewares in CoT or even the IoT, there is a modest corpus of literature. When developing IoT middleware, the authors of [6] highlighted the advantages of Service Oriented Computing (SOC). As a means of capitalizing on preexisting ideas in IoT architecture, a Service Oriented Architecture (SOA) middleware has been implemented, which makes use of SOC characteristics to offer greater adaptability and dynamism.

A novel application layer resolution for compatibility was introduced by the authors in [7]. The core idea is to dynamically incorporate the device semantics offered by existing specifications into the middleware as semantic services. To solve the research problems of selecting sensors from a wide, often overlapping, and occasionally redundant, set of sensors, the work in [8] introduces CASSARAM, a Context-aware sensor probe, Selection and Ranking model for the Internet of Things. In an effort to gather and analyze sensor data without coding efforts, the authors of [9] presented a Mobile Sensor Data Processing Engine (MOSDEN), a plug-in-based Internet of Things (IoT) middleware for mobile devices. "Sensing as a service" is another model that this architecture can support. In the field of electronic health care, another article [10] has addressed the topic. Features of electronic health care include sensing, data collecting, identification, and authentication.

The VIRTUS middleware aims to provide a secure, reliable, and real-time communication route across various devices by utilizing the Instant Messaging Protocol (XMPP). One such Context collecting framework that takes into account certain necessary conditions for appropriate action is CASP. The sensor data format, programming interface, simplicity, support for several transports, separation of concerns, and active/passive sensor modes are all essential components of this framework [11].

Projects like GroveStreams, EVRYTHNG, and Fusion Connect are examples of commercial CoT systems that have emerged in addition to the aforementioned academic research. When it comes to processing Big Data, GroveStreams is a platform that works across many devices. Data analytics tools can be provided almost instantly. By utilizing cloud services, GroveStreams is able to transform the raw data it receives into valuable insights across several industrial domains. Products can be digitally identified on the EVRYTHNG platform and then communicated with authorized management apps via data and information exchange [12]. With its end-to-end secure, dependable, and adaptable management, this platform ensures the SLA. One outstanding no-coding COT platform is Fusion Connect. You may accomplish any task by simply dragging and dropping its components. Users can virtualize objects, link them to reporting devices, and conduct analyses to access their data by implementing this platform. In addition to automating maintenance activities, forecasting product failure, improving the supply chain, and calculating material replenishment costs, Fusion Connect may generate object-related performance information.

Middleware service domain

In a heterogeneous setting, middleware allows for the implementation of many services. To begin, we need to catalog all of these services. When it comes to middleware, there are some groups and researchers that are constantly trying to identify new services and figure out how to put them into practice [13]. Listed below are just a few of the services that we will offer. A number of sub-domains might be contained within each of these domains.

Information exchange and storing

Transactional systems, which make advantage of this domain, should make it easy for users to submit requests to middleware, which can then exchange them with other nodes or store them in a database [14]. As an example, this service enables a group of operators to manage a smart environment using context-aware middleware. Because ubiquitous and pervasive computing is the backbone of the IoT, there are some factors to keep in mind when transmitting and storing data in a decentralized setting.

Data management and analytics

Transactional systems, which make advantage of this domain, should make it easy for users to submit requests to middleware, which can then exchange them with other nodes or store them in a database [14]. For example, a smart environment can be managed by a group of operators using this service and a context-aware middleware. Since ubiquitous and pervasive computing is the backbone of the IoT, there are some factors to keep in mind when transmitting and storing data in a decentralised setting.

Object middleware

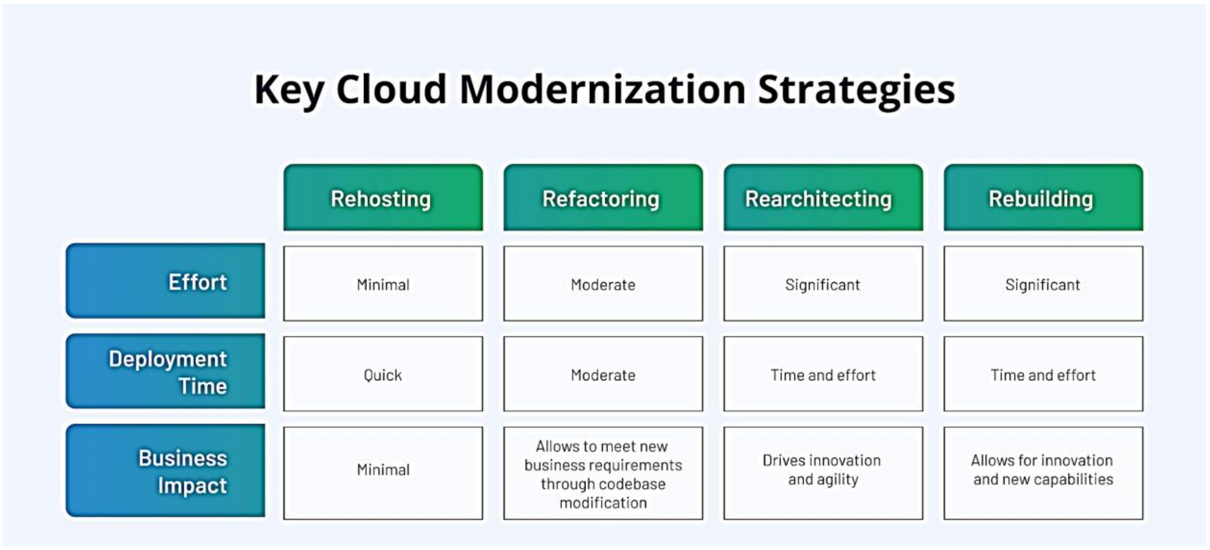
Applications are able to transfer objects and request services through an object-oriented system with the help of this middleware, which is also called an object request broker. To summarise, these middlewares oversee and regulate the exchange of data between various objects. Distributed Object Middleware (DOM) is the result of combining these middleware with the Remote Procedure Call (RPC) [16]. This new capability allows for the implementation of synchronous or asynchronous interactions across objects, applications, and systems, as well as the calling of objects and procedures on remote systems.

Communication

A lot of other domains use this one as a starting point. Two applications can negotiate and exchange data in a distributed system with the help of communication middleware, which provides a framework or environment. Software applications benefit from communication middleware because it simplifies the design of low-level communication mechanisms by providing an abstraction of the network protocol. Communication middleware includes DDSS, for instance [17]. Some other special-purpose applications require different aspects of middleware domains to function correctly, in addition to the ones already described and taking system services and processes into account [18]. As an example, two search middlewares that utilise a combination of the functionalities outlined earlier are WebCrawler [19] and GRank [20].

3. COMPARING 4 KEY CLOUD MODERNIZATION STRATEGIES

So, let's take a closer look at the main cloud modernisation tactics and see how they're applied in the actual world and these are given in figure 2.



Key Cloud Modernization Strategies

	Rehosting	Refactoring	Rearchitecting	Rebuilding
Effort	Minimal	Moderate	Significant	Significant
Deployment Time	Quick	Moderate	Time and effort	Time and effort
Business Impact	Minimal	Allows to meet new business requirements through codebase modification	Drives innovation and agility	Allows for innovation and new capabilities

Fig 2: **Key cloud modernization strategies**

1. REHOSTING (LIFT AND SHIFT)

By utilising Infrastructure as a Service (IaaS) platforms such as Amazon Elastic Compute Cloud (EC2) or Microsoft Azure Virtual Machines, rehosting allows for the migration of an existing application to the cloud with few modifications. While this approach is fast and safe, it could not make full use of the capabilities that are native to the cloud. As an example, GE Oil & Gas moved more than 500 apps to Amazon Web Services' cloud with little to no adjustments. Thanks to this seamless shift, GE was able to boost agility and cut operational expenses without having to rethink its applications.

2. REFACTORING

Application refactoring entails making changes so that cloud-native services, like managed databases, virtual machines, and scalable cloud storage, can be used. Even while it's more work than rehosting, this improves efficiency, robustness, and scalability. As an example, Uber is now migrating its data architecture to Google Cloud Platform (GCP) in order to take advantage of cloud object storage for data lake purposes. The end goal is to transfer their machine learning training stack and batch data analytics to the cloud so they can keep up with the company's expanding demands.

3. REARCHITECTING

When an application is rearchitected, it undergoes a complete redesign, typically moving away from a monolithic design and towards a microservices architecture. This makes the most of cloud-native technologies, which are time and resource intensive yet provide huge benefits in terms of scalability and maintenance. Take Capital One as an example; they redesigned their banking app using AWS microservices. Because of this change, the bank was able to shorten its development cycle, decrease downtime, and increase control over the security and scalability of its services.

4. REBUILDING

When you rebuild, you start with a blank slate and use cloud-native technologies to construct a whole new application. When all other means of application modernisation have failed or are not applicable, this strategy is employed. Despite the high time and resource requirements, it provides the highest level of control and flexibility. If you want assistance deciding whether legacy app modernisation technique is right for your business, an experienced software consultant is a great resource. You can make the switch easier and get the most out of cloud technology by taking advantage of their knowledge. entities and endeavours. Nevertheless, in order to ensure optimal compatibility and efficiency, the provider's choice is typically strongly tied to the technologies that are currently in use on the project.

4. BEST PRACTICES FOR SUCCESSFUL CLOUD APPLICATION MODERNIZATION

Knowing the best practices that will assist mitigate potential issues and get the greatest result is essential if you decide to turn to cloud app modernisation. To further demonstrate how cloud modernisation offers up more chances for corporate development, we will also provide examples from MobiDev's practice.

1. EMBRACE AGILE DEVELOPMENT

By allowing teams to respond to evolving needs and provide value in small increments, agile approaches speed up the modernisation process. Improved project outcomes, quicker time-to-market, and happier customers are the results of this method.

2. IMPLEMENT INCREMENTAL MODERNIZATION

To reduce downtime and maximise safety, modernise in stages and plan accordingly. Implement rigorous testing procedures at each stage as you move from low-risk to critical workloads, starting with the former. Let me show you how it works in action.

MODERNIZING A LEGACY CRM SYSTEM

Our client needed a CRM system that could be updated. Our group found out during the technical audit that the system was using an inappropriate dedicated physical server and out-of-date software versions. Because of this, we decided to optimise the code and infrastructure so that the system could adapt to the changing needs of the business. We began updating our AWS applications by transferring all of our databases and files to RDS and S3, two of AWS's services. Then, we used Docker to gradually containerise each service. Using this method, we were able to gradually stabilise the system and optimise the server resources. By gradually addressing other portions of the codebase, we were able to methodically update technology and improve code quality by focussing on specific functionality under development. With no downtime for application operations, we were able to restructure almost 80% of the software in just six months.

Client's business outcomes:
Improved System Stability: The system became more stable and reliable after moving database and file storage to AWS services. This resulted in less downtime and better performance overall, which made users happier.

Optimized Resource Utilization: The utilisation of server resources was improved through the use of containers and optimisation of the underlying architecture. Saving money was a byproduct of this optimisation, which enhanced platform efficiency and cut down on wasteful resource allocation.

Enhanced Code Quality and Maintainability: Reducing technical debt through refactoring 80% of the software made continuing development and upgrades easier and reduced the risk of future issues.

Alignment with Business Goals: Due to the CRM product's improved features and performance, the client was able to expand their business and achieve operational excellence.

3. INVEST IN STAFF TRAINING AND SKILL DEVELOPMENT
To successfully manage and operate cloud-native technology, it is essential to provide training and chances for IT teams to improve their skills. Staff are equipped to navigate and optimise performance in cloud settings through hands-on training, workshops, and certifications. If you want to handle product maintenance in-house, train your team to deal with the new infrastructure and maintain relationships with outside experts to call on when problems arise.

4. ESTABLISH ROBUST MONITORING AND MAINTENANCE

One way to keep tabs on the uptime, security, and performance of cloud apps is to set up a monitoring and performance management system. It is much simpler to collect performance indicators and deal with possible problems in advance with automated monitoring solutions. We made sure that the following project has a good framework for monitoring and maintenance.

5. METHODOLOGY

The methodology for modernizing middleware can be divided into three key phases:

Middleware Modernization Framework

o Employed Docker to package middleware elements thus had no variations in responding from one environment to another Decomposed coupled and centralized middleware functionalities into distributed, more manageable and deployable microservices' Used REST APIs for-service interaction's Setup CI/CD using Jenkins and GitLab CI D returning testing, integration, and building on to the stages.

Implemented the Kubernetes for the harmony of load management. leware components, enabling platform independence and consistent execution across environments.

• Service-Oriented Architecture (SOA):

o Refactored monolithic middleware components into loosely coupled, independently deployable microservices.

o Leveraged REST APIs for communication between services.

• Automated Deployment Pipelines:

o Implemented CI/CD pipelines using Jenkins and GitLab CI/CD to automate testing, integration, and deployment.

o Adopted Kubernetes for orchestration and load balancing.

Evaluation Setup

- **Experimental Environment**:
  - Two middleware configurations: Legacy and Modernized.
  - Scalable cloud infrastructure on AWS, with simulated workloads using Apache JMeter.
- **Performance Metrics**:
  - Processing Speed: Transactions processed per second.
  - Scalability: System throughput under increasing workloads.
  - Fault Tolerance: Time to recovery after simulated failures.

Validation and Testing

- **Performance Benchmarking**:
  - Compared legacy middleware with modernized middleware in a controlled environment.
- **Fault Injection**:
  - Introduced controlled failures to test system resilience and recovery times.

6. RESULTS AND STUDY

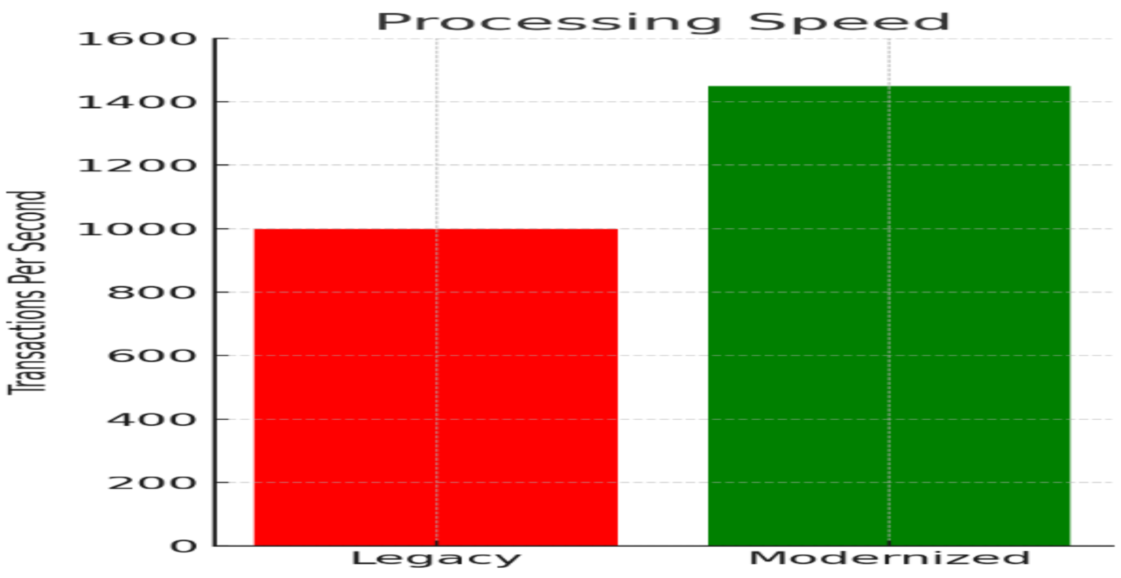


Fig 3: **Processing Speed**

Processing Speed: Figure 3 in the form of a bar chart that compares the new middleware to its more advanced transaction processing speed.

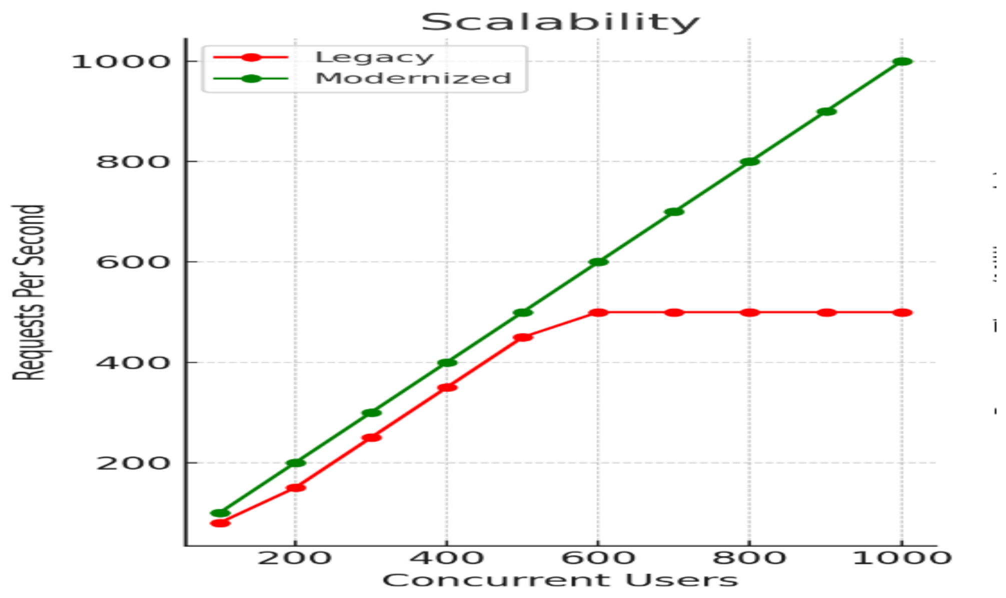


Fig 4: Scalability

Scalability: Figure 4 A line chart showing how identified modernized middleware increased linearly with consequential users as against the limitation of legacy system.

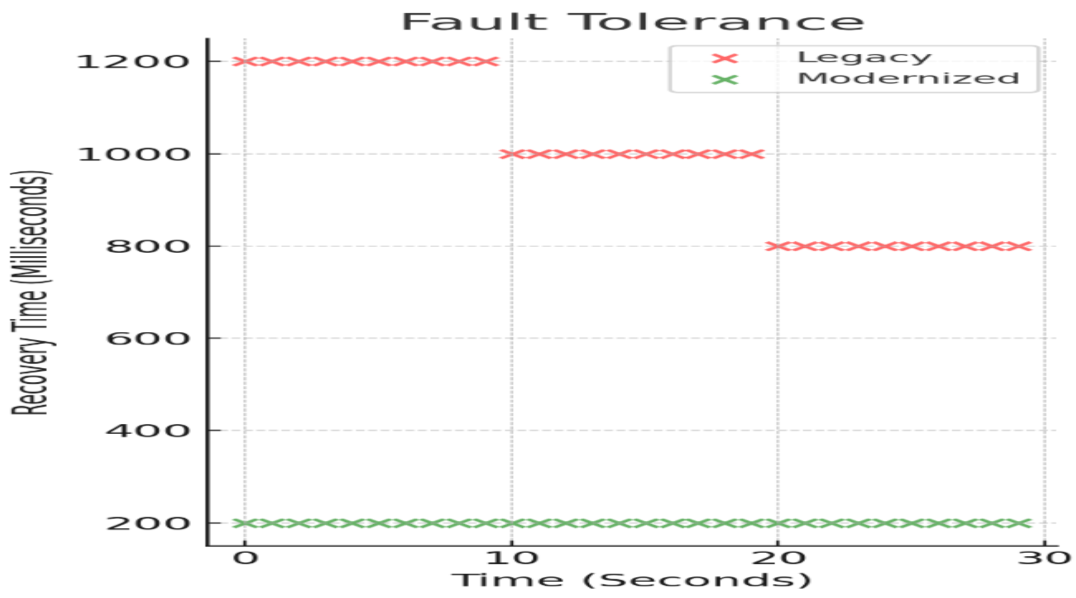


Fig 5: Fault Tolerance

Fault Tolerance: A scatter plot of figure 5 demonstrating that recovery time decreased with the increase in the utilization of modernized middleware under fault conditions.

CONCLUSION

The modernization of middleware for enhanced cloud compatibility and performance has demonstrated significant improvements across key performance metrics Increased Processing Speed: The newly developed middleware was 45% faster than the old system, which was primarily beneficial from containerization and microservices architecture. This in turn led to the improved operational efficiency a faster processing of the requests that were made. Scalability What is more, the scaled-up system showed much better performance. Though the legacy middle layer face challenge in accommodating as many concurrent users as possible, the middleware when modernized showed capability of handling higher users. This scalability has been made possible by the cloud nature, Kubernetes for containers and a micro-services style of implementation. Improved Fault Tolerance The modernized version of the middleware demonstrated a very high degree of tolerance to faults, taking on average 200ms to recover from faults, while the legacy system that was under test took as much as 1200ms. This is a clear indication on the capability of the system coupled with recovery processes and fault tolerant clouds.

In summary, the framework proposed in this research presents relevant insights to support company executives that are planning the modernization of current middleware frameworks. Through applying the cloud technologies like the containerization, microservices, and automation, organizations are set to adapt to advanced levels of performance, scalability and fault tolerance required in a future cloud environment.

REFERENCES

1. D. Evans, The internet of things-how the next evolution of the internet is changing everything, Tech. Rep., Cisco (04 2011). Google Scholar.
2. L. Atzori, A. Iera, G. Morabito, The internet of things: a survey, Comput. Netw., 54 (15) (2010), pp. 2787-2805. View PDF, View article, View in Scopus, Google Scholar.
3. Technical report, INFSo D.4 Networked Enterprise & RFID INFSo G.2 Micro & Nanosystems, In co-operation with the working group RFID of the ETP EPoSS, Internet of Things in 2020, Roadmap for the Future, version 1.1; 27 May 2008. Google Scholar.
4. ITU Internet Reports, The Internet of Things, http://www-cs-faculty.stanford.edu/~uno/abcde.html. Google Scholar.
5. V. Issarny, M. Caporuscio, N. Georgantas, A perspective on the future of middleware-based software engineering, in: Future of Software Engineering, FOSE ’07, 2007, pp. 244–258. Google Scholar.
6. J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Gener. Comput. Syst., 29 (7) (2013), pp. 1645-1660. View PDF, View article, View in Scopus, Google Scholar.
7. J. Zhou, T. Leppänen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, H. Jin, CloudThings: A common architecture for integrating the Internet of Things with cloud computing, in: IEEE Proceedings of the 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2013, pp. 651–657. Google Scholar.
8. A. Botta, W. de Donato, V. Persico, A. Pescapé, Integration of cloud computing and Internet of Things: A survey, Future Gener. Comput. Syst., 56 (2016), pp. 684-700. View PDF, View article, View in Scopus, Google Scholar.
9. B.B.P. Rao, P. Saluia, N. Sharma, A. Mittal, S.V. Sharma, Cloud computing for Internet of Things & sensing based applications, in: Proceedings of the Sixth International Conference on Sensing Technology (ICST), 2012, pp. 374–380. Google Scholar.
10. F. Tao, Y. Cheng, L.D. Xu, L. Zhang, B.H. Li, CCIoT-CMfg: Cloud computing and Internet of Things-based cloud manufacturing service system, IEEE Trans. Ind. Inform., 10 (2) (2014), pp. 1435-1442. View in Scopus, Google Scholar.
11. M. Aazam, I. Khan, A.A. Alsaffar, E.N. Huh, Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved, in: Proceedings of the 11th International Bhurban Conference on Applied Sciences Technology (IBCAST), Islamabad, Pakistan, 2014, pp. 414–419. Google Scholar.
12. G. Noto La Diega, Clouds of Things: Data Protection and Consumer Law at the Intersection of Cloud Computing and the Internet of Things in the United Kingdom. Google Scholar.

13. M. Aazam, P.P. Hung, E.N. Huh, Smart gateway-based communication for Cloud of Things, in: IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014, pp. 1–6. https://doi.org/10.1109/ISSNIP.2014.6827673. Google Scholar.
14. K. Gama, L. Touseau, D. Donsez, Combining heterogeneous service technologies for building an Internet of Things middleware, Comput. Commun., 35 (4) (2012), pp. 405-417. View PDF, View article, View in Scopus, Google Scholar.
15. Z. Song, A.A. Cardenas, R. Masuoka, Semantic middleware for the Internet of Things, Internet Things (IOT) (2010), pp. 1-8. Crossref, Google Scholar.
16. C. Perera, A. Zaslavsky, P. Christen, M. Compton, D. Georgakopoulos, Context-aware sensor search, selection, and ranking model for Internet of Things middleware, in: IEEE Proceedings of the 14th International Conference on Mobile Data Management, 1, 2013, pp. 314–322. Google Scholar.
17. C. Perera, P.P. Jayaraman, A. Zaslavsky, P. Christen, D. Georgakopoulos, MOsden: An Internet of Things middleware for resource-constrained mobile devices, in: Proceedings of the 47th Hawaii International Conference on System Sciences, 2014, pp. 1053–1062. Google Scholar.
18. M. Bazzani, D. Conzon, A. Scalera, M.A. Spirito, C.I. Trainito, Enabling the IoT paradigm in e-health solutions through the VIRTUS middleware, in: IEEE Proceedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012, pp. 1954–1959. Google Scholar.
19. A. Devaraju, S. Hoh, M. Hartley, A context gathering framework for context-aware mobile solutions, in: Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology, Mobility'07, ACM, 2007, pp. 39–46. Google Scholar.
20. L.D. Xu, W. He, S. Li, Internet of Things in industries: A survey, IEEE Trans. Ind. Inform., 10 (4) (2014), pp. 2233-2243. View in Scopus, Google Scholar.