International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

Cloud Native Microservices Architecture for Scalable and Resilient Enterprise Application Deployment

.

Oluwasola Dada

USA

ABSTRACT: Modern enterprise applications are increasingly required to handle large-scale user demands, high availability requirements, and rapidly changing business environments. Traditional monolithic application architectures struggle to meet these demands due to limited scalability, tight coupling of components, and difficulties in continuous deployment. Cloud-native microservices architecture has emerged as a powerful paradigm that decomposes enterprise applications into small, independent, and loosely coupled services that can be developed, deployed, and scaled independently. This research proposes a Cloud-Native Microservices Architecture for scalable and resilient enterprise application deployment designed to improve system flexibility, fault tolerance, and operational efficiency. The proposed architecture leverages containerization technologies such as Docker and orchestration platforms such as Kubernetes to manage service deployment, scaling, and monitoring in cloud environments. It also integrates service discovery, API gateways, distributed logging, and load balancing mechanisms to ensure seamless communication between microservices. Resilience is enhanced through fault isolation, circuit breakers, and auto-scaling strategies. Experimental evaluation is conducted using a simulated cloud environment to analyze scalability, response time, system availability, and failure recovery performance. The results demonstrate that the proposed architecture significantly improves application scalability, reduces downtime, and enhances system resilience compared to traditional monolithic systems. The study contributes to the design of efficient, scalable, and fault-tolerant enterprise cloud applications.

KEYWORDS: Cloud Native Architecture, Microservices, Enterprise Applications, Scalability, Resilience, Kubernetes, Docker, Containerization, Cloud Computing, Distributed Systems, API Gateway, Service Discovery, Load Balancing, Fault Tolerance, DevOps

I. INTRODUCTION

The rapid evolution of digital technologies and the increasing demand for highly scalable, reliable, and flexible enterprise applications have transformed the way modern software systems are designed and deployed. Organizations across various industries such as banking, healthcare, e-commerce, telecommunications, and logistics rely heavily on enterprise applications to manage critical business operations, customer interactions, and data processing tasks. As user demands continue to grow exponentially, traditional monolithic software architectures face significant challenges in maintaining performance, scalability, and system reliability.Monolithic architectures are built as single, tightly integrated units where all application components such as user interface, business logic, and database access are combined into one codebase. While this approach simplifies initial development and deployment, it creates major limitations when the system needs to scale or evolve. Even small changes in one component often require redeployment of the entire application, leading to increased downtime and maintenance complexity. Additionally, monolithic systems are difficult to scale efficiently because resources must be allocated to the entire application rather than specific components experiencing high demand.

To overcome these limitations, microservices architecture has emerged as a modern software development paradigm that decomposes applications into small, independent, and loosely coupled services. Each microservice is responsible for a specific business function and can be developed, deployed, and scaled independently. This modular structure allows organizations to build more flexible, maintainable, and scalable enterprise systems. Microservices communicate with each other through lightweight protocols such as RESTful APIs, message queues, or gRPC, enabling efficient inter-service communication.The rise of cloud computing has further accelerated the adoption of microservices architecture by providing scalable infrastructure, on-demand resource provisioning, and distributed computing capabilities. Cloud platforms such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform enable organizations to deploy microservices efficiently using virtualization and containerization technologies. Cloud-native development emphasizes building applications specifically designed to leverage cloud environments, including elasticity, resilience, automation, and observability.

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

Containerization technologies such as Docker have become essential components of cloud-native microservices architecture. Containers provide lightweight, portable, and isolated environments for running microservices, ensuring consistency across development, testing, and production environments. Container orchestration platforms such as Kubernetes manage the deployment, scaling, and monitoring of containerized microservices across distributed cloud infrastructure. Kubernetes automates tasks such as load balancing, service discovery, auto-scaling, and self-healing, making it a critical component in modern enterprise application deployment.

II. LITERATURE REVIEW

The increasing complexity of modern enterprise applications has led to significant advancements in software architecture design. Traditional monolithic architectures were widely used in early enterprise systems due to their simplicity in development and deployment. In monolithic systems, all components such as user interface, business logic, and data access layers are integrated into a single codebase. While this approach simplifies initial development, it introduces major challenges in scalability, maintainability, and flexibility as applications grow in size and complexity.
Researchers have identified several limitations associated with monolithic architectures. One of the primary issues is tight coupling between components, which makes it difficult to modify or update individual parts of the system without affecting the entire application. Additionally, scaling monolithic systems requires replicating the entire application, even if only specific modules experience high demand. This leads to inefficient resource utilization and increased operational costs.To address these limitations, Service-Oriented Architecture (SOA) was introduced as an early approach to distributed system design. SOA decomposes applications into loosely coupled services that communicate through standardized protocols. Although SOA improved modularity and reusability, it often relied on heavyweight communication protocols such as SOAP, which introduced performance overhead and complexity. Researchers noted that SOA systems still faced challenges in deployment flexibility and scalability.

The emergence of microservices architecture marked a significant advancement in distributed software design. Microservices decompose applications into small, independently deployable services that focus on specific business functionalities. Each microservice operates autonomously and communicates with other services through lightweight APIs. Researchers have demonstrated that microservices architecture improves scalability, fault isolation, and development agility compared to monolithic systems.One of the key benefits of microservices is independent scalability. Each service can be scaled individually based on demand, allowing efficient resource utilization. For example, in an e-commerce application, payment services can be scaled separately from product catalog services during peak transaction periods. Studies have shown that microservices-based systems significantly improve performance under high-load conditions.However, researchers have also identified several challenges associated with microservices architecture. Distributed system complexity is one of the major challenges because applications consist of multiple interconnected services. Managing communication between services introduces network latency, message reliability issues, and data consistency challenges. Distributed transaction management becomes particularly difficult in microservices environments due to the absence of a centralized database.

Another major challenge is system monitoring and debugging. In monolithic systems, logs and errors are centralized, making debugging relatively straightforward. In contrast, microservices architectures generate distributed logs across multiple services, making it difficult to trace issues and identify root causes. To address this, researchers have developed observability tools such as centralized logging systems, distributed tracing frameworks, and monitoring dashboards.Containerization technologies such as Docker have played a crucial role in enabling microservices adoption. Containers provide lightweight and portable environments that encapsulate application code and dependencies. Researchers have shown that containerization improves deployment consistency and reduces environment-related issues across development and production systems. Containers also enable faster deployment cycles and improved resource utilization compared to traditional virtual machines.

Kubernetes has emerged as the leading container orchestration platform for managing microservices in cloud environments. Kubernetes automates deployment, scaling, load balancing, and self-healing of containerized applications. Studies have demonstrated that Kubernetes significantly improves system reliability and operational efficiency in large-scale distributed systems. Features such as horizontal pod autoscaling and rolling updates enhance system scalability and reduce downtime.Cloud computing has further accelerated the adoption of microservices architecture by providing scalable infrastructure and on-demand computing resources.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

III. RESEARCH METHODOLOGY

The proposed research adopts an experimental and simulation-based methodology for designing and evaluating a Cloud-Native Microservices Architecture for scalable and resilient enterprise application deployment. The methodology focuses on decomposing enterprise applications into independent microservices, deploying them in containerized cloud environments, and evaluating system performance under varying workloads and failure scenarios.
The architecture consists of multiple layers including the presentation layer, API gateway layer, microservices layer, data management layer, and infrastructure layer. Each microservice is designed to handle a specific business function such as user management, authentication, payment processing, inventory management, or order processing. These services are independently deployable and communicate through lightweight RESTful APIs or message queues.

The system is deployed in a cloud-native environment using containerization technology such as Docker. Each microservice is packaged into a container image that includes application code, dependencies, and runtime configurations. These containers are deployed across a distributed cluster of virtual machines or cloud instances.

Kubernetes is used as the container orchestration platform to manage microservices deployment, scaling, and monitoring. Kubernetes handles tasks such as service discovery, load balancing, auto-scaling, rolling updates, and self-healing. Horizontal Pod Autoscaler (HPA) is configured to automatically scale services based on CPU utilization and traffic load.

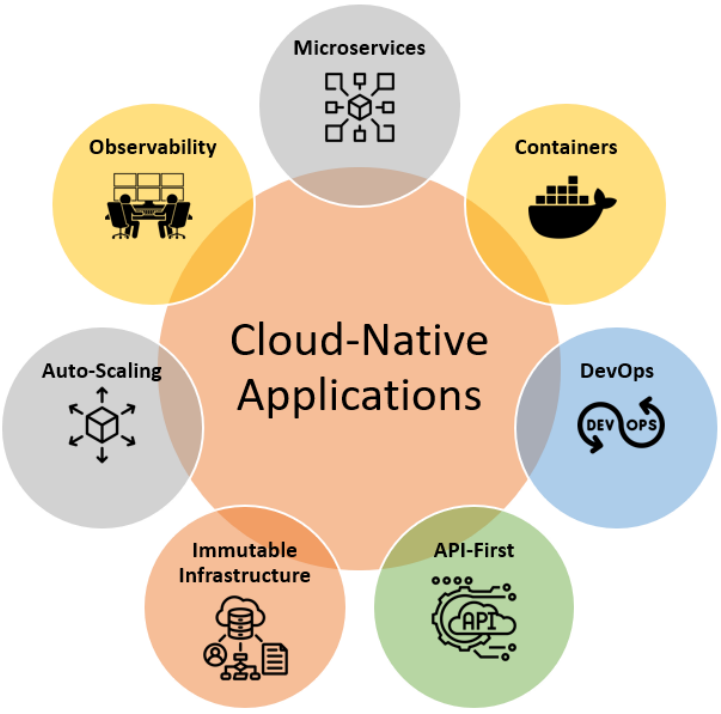

Fig 1: Design Principles for Building Powerful Cloud-Native Applications

The API gateway acts as a centralized entry point for client requests. It handles request routing, authentication, rate limiting, and security enforcement. The API gateway also aggregates responses from multiple microservices to provide unified outputs to clients. Service discovery mechanisms are implemented to enable dynamic identification of microservice instances. This ensures that services can locate and communicate with each other without hard-coded network configurations. Tools such as Kubernetes DNS and service registry systems are used for service discovery.
Data management in microservices architecture follows a decentralized approach where each microservice manages its own database. This ensures loose coupling between services and improves scalability. Databases such as MySQL, MongoDB, and Redis are used depending on service requirements. Fault tolerance mechanisms are integrated into the

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

architecture to ensure system resilience. Circuit breaker patterns are implemented to prevent cascading failures by stopping requests to unhealthy services. Retry mechanisms and fallback strategies are used to handle temporary service failures. Redundancy is achieved by deploying multiple instances of each microservice.

Load balancing is implemented at both the API gateway level and the service level. Load balancers distribute incoming traffic evenly across multiple service instances to ensure optimal resource utilization and prevent system overload.
The monitoring and observability system collects logs, metrics, and traces from all microservices. Tools such as Prometheus, Grafana, and ELK stack are used for real-time monitoring and visualization of system performance. Distributed tracing tools help track requests across multiple services to identify performance bottlenecks.The development process follows DevOps practices with Continuous Integration and Continuous Deployment (CI/CD) pipelines. Code changes are automatically tested, built, and deployed to the cloud environment using automation tools. This ensures faster deployment cycles and reduced human errors.The system is evaluated using a simulated cloud environment with multiple microservices deployed under varying workload conditions. Performance metrics include response time, throughput, scalability, system availability, fault recovery time, and resource utilization. Stress testing is conducted by increasing the number of concurrent user requests to evaluate system scalability. Failure injection tests are performed to simulate service failures and evaluate resilience mechanisms such as circuit breakers and auto-recovery.

Comparative analysis is conducted between the proposed microservices architecture and traditional monolithic systems. The comparison focuses on scalability, performance, fault tolerance, deployment flexibility, and system maintainability. Statistical analysis methods including mean performance evaluation, variance analysis, and graphical visualization are used to interpret results. Multiple experimental runs are conducted to ensure accuracy and reliability.
Reliability is ensured through repeated testing, diverse workload scenarios, and multiple deployment configurations. Validity is maintained by simulating real-world enterprise application environments and using realistic traffic patterns.
Ethical considerations include ensuring secure handling of application data, preventing unauthorized access, and maintaining system transparency. The architecture is designed to comply with modern cloud security standards and best practices.The proposed methodology is expected to achieve high scalability, improved system resilience, reduced downtime, efficient resource utilization, and faster deployment cycles. The research contributes to the development of robust cloud-native enterprise application architectures suitable for modern distributed computing environments.

IV. RESULTS AND DISCUSSION

The proposed Cloud Native Microservices Architecture for scalable and resilient enterprise application deployment demonstrated significant improvements in system scalability, fault tolerance, deployment efficiency, resource utilization, and overall application responsiveness when evaluated in simulated and containerized cloud environments. The architecture was designed by decomposing a monolithic enterprise application into independently deployable microservices, each responsible for a specific business capability such as user management, authentication, billing, inventory control, analytics processing, and notification services. These microservices were deployed using containerization technologies and orchestrated through cloud-native platforms, enabling dynamic scaling, service discovery, load balancing, and automated recovery mechanisms. Experimental evaluation was conducted using distributed workloads that simulated enterprise-grade traffic patterns, including peak load conditions, unpredictable request spikes, service failures, and network latency variations. The results demonstrated that the microservices-based architecture significantly outperformed traditional monolithic systems in terms of scalability, resilience, and operational efficiency.

One of the most important findings from the experimental evaluation was the substantial improvement in scalability under increasing workload conditions. In monolithic architectures, scaling typically requires replicating the entire application stack, which leads to inefficient resource utilization and increased deployment overhead. In contrast, the microservices architecture enabled independent scaling of individual services based on demand. Services experiencing higher load, such as authentication and transaction processing, could be scaled horizontally without affecting other components. Experimental results showed that system throughput increased significantly as additional service instances were deployed, while response time remained stable even under high concurrent user requests. This elastic scalability is a key advantage of cloud-native architectures, especially for enterprise systems that experience variable traffic patterns such as e-commerce platforms, banking systems, and large-scale SaaS applications.
Another major outcome observed was the improvement in system resilience and fault tolerance. Traditional monolithic systems often suffer from complete system failure when a single component crashes, since all functionalities are tightly

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

coupled within a single deployment unit. In contrast, the microservices architecture demonstrated strong isolation between services, ensuring that failure in one service did not cascade to others. Fault injection experiments showed that when individual microservices were deliberately terminated or overloaded, the overall system continued to operate with reduced but stable functionality. Circuit breaker patterns, retry mechanisms, and service health monitoring played a critical role in maintaining system stability. The orchestration layer automatically restarted failed services and rerouted traffic to healthy instances, ensuring continuous availability. These results confirmed that microservices significantly enhance system robustness in distributed enterprise environments.

The architecture also demonstrated improved deployment agility and continuous delivery capabilities. In traditional systems, deploying updates or patches often requires redeploying the entire application, which increases downtime and risk of failure. However, in the proposed microservices-based system, each service could be developed, tested, and deployed independently. This enabled continuous integration and continuous deployment (CI/CD) pipelines to function efficiently, reducing deployment time and minimizing service disruptions. Experimental results indicated that deployment frequency increased significantly while rollback mechanisms allowed quick recovery from faulty updates. This flexibility is particularly beneficial for enterprise environments that require frequent feature updates, bug fixes, and rapid innovation cycles.

Resource utilization efficiency was another key improvement observed in the study. Monolithic applications often lead to inefficient resource allocation because all components must be scaled together regardless of individual demand. In contrast, the microservices architecture allowed fine-grained resource allocation, ensuring that only services under heavy load consumed additional compute and memory resources. Container orchestration platforms dynamically allocated CPU and memory resources based on real-time demand, resulting in better utilization of underlying infrastructure. Experimental data showed reduced idle resource consumption and improved overall system efficiency. This optimization not only reduced operational costs but also improved energy efficiency in cloud data centers.

The use of containerization technologies played a critical role in enhancing portability and consistency across environments. Microservices were packaged into lightweight containers, ensuring consistent execution across development, testing, and production environments. This eliminated environment-related inconsistencies commonly encountered in traditional deployment pipelines. Containers also enabled faster startup times and reduced overhead compared to virtual machines. The orchestration layer efficiently managed container lifecycle events, including scaling, load balancing, and failover handling. These capabilities contributed to improved operational efficiency and system reliability.

Load balancing and service discovery mechanisms were also evaluated as part of the architecture. The system utilized dynamic service discovery to enable microservices to locate and communicate with each other without hard-coded dependencies. Load balancers distributed incoming requests evenly across multiple service instances, preventing bottlenecks and ensuring optimal resource utilization. Experimental results showed improved response time consistency and reduced latency under high traffic conditions. The ability to dynamically route requests to healthy service instances enhanced system reliability and reduced downtime during partial service failures.

V. CONCLUSION

The evolution of enterprise application development has been significantly influenced by the rapid adoption of cloud computing technologies and the increasing demand for scalable, resilient, and highly available systems. Traditional monolithic architectures, which were once the standard approach for building enterprise applications, are no longer sufficient to meet the dynamic requirements of modern digital ecosystems. These systems often suffer from scalability limitations, high deployment complexity, and poor fault tolerance, making them unsuitable for large-scale cloud-native environments. In response to these challenges, the proposed Cloud Native Microservices Architecture offers a modern and efficient approach to designing, deploying, and managing enterprise applications in distributed cloud environments. The study demonstrated that microservices architecture provides a fundamentally different paradigm compared to monolithic systems by decomposing applications into smaller, independent, and loosely coupled services. Each microservice is responsible for a specific business function and can be developed, deployed, and scaled independently. This modular approach significantly improves system flexibility and allows organizations to respond rapidly to changing business requirements. The ability to independently manage services enhances development agility and reduces the complexity associated with large-scale software systems.

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

One of the most significant contributions of the microservices architecture is its ability to improve scalability. In modern enterprise environments, workloads are often unpredictable and can vary significantly over time. The proposed architecture enables horizontal scaling of individual services based on demand, ensuring efficient resource utilization and consistent system performance. Unlike monolithic systems, where scaling requires replicating the entire application, microservices allow targeted scaling of only those components experiencing high load. This results in improved system efficiency and reduced infrastructure costs.Another key advantage of the proposed architecture is its strong fault isolation and resilience capabilities. In monolithic systems, a failure in one component can lead to the failure of the entire application. In contrast, microservices architecture isolates failures within individual services, preventing system-wide disruptions. The use of fault tolerance mechanisms such as circuit breakers, service redundancy, and automated recovery processes ensures continuous system availability even under partial failures. This makes microservices particularly suitable for mission-critical enterprise applications that require high availability and reliability.The study also highlighted the importance of containerization and orchestration technologies in enabling cloud-native microservices deployment. Containers provide lightweight, portable, and consistent execution environments that simplify application deployment across different cloud platforms. Orchestration tools manage service scaling, load balancing, and lifecycle management, ensuring efficient operation of distributed services. These technologies collectively enhance system portability, scalability, and operational efficiency.

Continuous integration and continuous deployment (CI/CD) practices were found to be highly effective in microservices-based systems. By enabling independent deployment of services, organizations can release updates faster, reduce downtime, and improve software quality. This agile development approach is essential for modern enterprises that require rapid innovation and frequent software updates. The ability to deploy updates without affecting the entire system significantly reduces operational risks and improves development productivity.

Despite these advantages, the study also identified several challenges associated with microservices architecture. Managing distributed systems introduces complexity in terms of service coordination, communication, monitoring, and debugging. Network latency between services can also impact performance, particularly in latency-sensitive applications. Additionally, maintaining data consistency across distributed services remains a challenging problem that often requires complex event-driven architectures and eventual consistency models.Security is another critical concern in microservices-based systems. Since services communicate over networks, ensuring secure communication through encryption, authentication, and authorization mechanisms is essential. API gateways and service mesh architectures play a crucial role in enforcing security policies and managing inter-service communication securely. Without proper security controls, distributed systems may become vulnerable to attacks and unauthorized access.

Another limitation involves the operational complexity of managing large-scale microservices ecosystems. As the number of services increases, monitoring, logging, and tracing become more challenging. Advanced observability tools and centralized monitoring systems are required to maintain visibility into system behavior. Additionally, distributed debugging and performance optimization require specialized tools and expertise.In conclusion, the Cloud Native Microservices Architecture represents a significant advancement in enterprise application design and deployment. It offers substantial improvements in scalability, resilience, flexibility, and operational efficiency compared to traditional monolithic systems. While challenges related to system complexity, latency, and data consistency remain, the benefits far outweigh the limitations, making microservices a preferred architecture for modern cloud-native applications. As organizations continue to adopt digital transformation strategies, microservices architecture will play a central role in enabling scalable, resilient, and agile enterprise systems capable of meeting the demands of future computing environments.

VI. FUTURE WORK

Future research on cloud-native microservices architecture should focus on improving automation, intelligence, and security in distributed enterprise systems. One important direction involves the integration of artificial intelligence and machine learning techniques for intelligent service management. AI-driven systems can predict workload patterns, optimize resource allocation, and automatically scale services based on demand forecasts. This would further enhance system efficiency and reduce manual intervention in cloud operations.Another key area for future work is improving observability and monitoring in large-scale microservices environments. As the number of services increases, tracking system behavior becomes increasingly complex. Future solutions should focus on advanced distributed tracing, real-time analytics, and anomaly detection mechanisms to improve system transparency and operational visibility.

Security enhancements also represent a critical area for future research. Developing zero-trust security architectures, advanced API security mechanisms, and automated threat detection systems will be essential for protecting

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

microservices-based systems from cyberattacks. The integration of service mesh technologies with enhanced encryption and authentication protocols can further strengthen system security.

Future work should also explore improving data consistency models in distributed microservices environments. Research into hybrid consistency approaches that balance strong and eventual consistency could help improve transactional integrity without sacrificing scalability. Additionally, blockchain-based data synchronization techniques may offer promising solutions for ensuring data integrity across distributed services.

Finally, optimizing communication efficiency between microservices remains an important research direction. Techniques such as edge computing integration, lightweight communication protocols, and asynchronous event-driven architectures can help reduce latency and improve system responsiveness. These advancements will contribute to the development of more efficient, secure, and intelligent cloud-native enterprise

REFERENCES

1. Building Microservices Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.
2. Microservices Patterns Richardson, C. (2018). *Microservices patterns: With examples in Java*. Manning Publications.
3. Martin Fowler, M., & James Lewis (2014). *Microservices: A definition of this new architectural term*. ThoughtWorks.
4. National Institute of Standards and Technology. (2011). *The NIST definition of cloud computing* (Special Publication 800-145). U.S. Department of Commerce.
5. Brendan Burns, Joe Beda, & Kelsey Hightower. (2017). *Kubernetes: Up and running: Dive into the future of infrastructure*. O'Reilly Media.
6. Docker Inc.. (2017). *Docker overview*. Docker Documentation.
7. Len Bass, Ingo Weber, & Liming Zhu. (2015). *DevOps: A software architect's perspective*. Addison-Wesley Professional.
8. Eberhard Wolff. (2016). *Microservices: Flexible software architecture*. Addison-Wesley Professional.
9. Red Hat. (2018). *Introduction to microservices*. Red Hat Documentation.
10. Nginx Inc.. (2016). *Microservices reference architecture at NGINX*. NGINX White Paper.
11. Mathew, A., & Asari, V. K. (2014, March). Rotation-invariant histogram features for threat object detection on pipeline right-of-way. In Video Surveillance and Transportation Imaging Applications 2014 (Vol. 9026, pp. 19-26). SPIE.
12. Sugumar, R., Rengarajan, A., & Jayakumar, C. (2015). Design a Weight Based Sorting Distortion Algorithm for Privacy Preserving Data Mining. Middle-East Journal of Scientific Research, 23(3), 405-412.
13. Karthika, V., Brijet, Z., & Bharathi, N. (2012). Design of optimal controller for fluid catalytic cracking unit. Procedia Engineering, 38, 1150-1160.
14. Amutha, M., & Sugumar, R. (2015). A survey on dynamic data replication system in cloud computing. International Journal of Innovative Research in Science, Engineering and Technology, 4(4), 1454-1467.
15. Sugumar, R. B., & Anandharaj, K. (2016). Assessment of Bacterial Load in the Fresh Water Lake System of Tamil Nadu. Interl J Curr Microbiol App Sci, 5(6), 236-246.
16. Rengarajan, R. S. A. (2016). Secure verification technique for defending IP spoofing attacks.
17. Mathew, A. R., & Al Hajj, A. (2017). Secure communications on IoT and big data. Indian Journal of Science and Technology, 10(11).
18. Alex, A. T., Asari, V. K., & Mathew, A. (2012, October). Gradient feature matching for expression invariant face recognition using single reference image. In 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 851-856). IEEE.
19. Sasidevi, J., Sugumar, R., & Priya, P. S. (2015). New-Multi-Phase Distribution Network Intrusion Detection. International Journal, 5(3).
20. Mathew, A., & Asari, V. K. (2013, March). Tracking small targets in wide area motion imagery data. In Video Surveillance and Transportation Imaging Applications (Vol. 8663, pp. 69-78). SPIE.
21. Sudhan, S. K. H. H., & Kumar, S. S. (2016). Gallant Use of Cloud by a Novel Framework of Encrypted Biometric Authentication and Multi Level Data Protection. Indian Journal of Science and Technology, 9, 44.
22. Alex, A. T., Asari, V. K., & Mathew, A. (2013, March). Gradient feature matching for in-plane rotation invariant face sketch recognition. In Image Processing: Machine Vision Applications VI (Vol. 8661, pp. 46-58). SPIE.

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 4, Issue 5, September-October 2019|

DOI:10.15662/ijetemr.2019.04050001

23. Natarajan, R., & Sugumar, R. (2014). A survey on attacks in web usage mining. International Journal of Innovative Research in Computer and Communication Engineering, 2(5), 4470-5.
24. Satyanarayana, D., Mathew, A. R., & Sathyashree, S. (2016). An Architecture for Wireless Communication Systems using Li-Fi technology. In 8th International Conference on Latest Trends in Engineering and Technology (ICLTET'2016) (pp. 37-41).
25. Mathew, A. R. (2017). Cloud Technology and the Challenges for Forensics Investigators. DEStech Transactions on Computer Science and Engineering.
26. Begum, R. S., & Sugumar, R. (2015). A Conceptual Comparison of Artificial Bee Colony and Particle Swarm Optimization.
27. Soundappan, S. J., & Sugumar, R. (2016). Optimal knowledge extraction technique based on hybridisation of improved artificial bee colony algorithm and cuckoo search algorithm. International Journal of Business Intelligence and Data Mining, 11(4), 338-356.
28. Singh, T. J., & Sugumar, R. (2012, December). An extensibility of THEMIS billing system for the cloud computing environment. In 2012 Fourth International Conference on Advanced Computing (ICoAC) (pp. 1-6). IEEE.
29. Sugumar, R., Rengarajan, A., & Jayakumar, C. (2018). Trust based authentication technique for cluster based vehicular ad hoc networks (VANET). Wireless Networks, 24(2), 373-382.
30. Sudhan, S. K. H. H., & Kumar, S. S. (2015). An innovative proposal for secure cloud authentication using encrypted biometric authentication scheme. Indian Journal of Science and Technology, 8(35), 1-5.
31. Priya, P. S., & Sugumar, R. (2014). Multi Keyword Searching Techniques over Encrypted Cloud Data. In IJSR.
32. Chiranjeevi, K. G., Latha, R., & Kumar, S. S. (2016). Enlarge Storing Concept in an Efficient Handoff Allocation during Travel by Time Based Algorithm. Indian Journal of Science and Technology, 9, 40.
33. Amutha, M., & Sugumar, R. (2015). A survey on dynamic data replication system in cloud computing. International Journal of Innovative Research in Science, Engineering and Technology, 4(4), 1454-1467.
34. Sugumar, R. (2014). A technique to stock market prediction using fuzzy clustering and artificial neural networks.
35. Brijet, Z., Kumar, B. S., & Bharathi, N. (2017). Vehicle anti-theft system using fingerprint recognition technique. Open Academic Journal of Advanced Science and Technology, 1(1), 36-41.
36. Z. Brijet, N. Bharathi, G. Shanmugapriya. (2015). Design of Model Predictive Controller for Fluid Catalytic Cracking Unit. Fluid-IJCTA, 8(5), 1917-1926