World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(REVIEW ARTICLE)

Check for updates

Architecting cloud-native enterprise platforms: Scalable and secure design patterns using AWS, Kubernetes, and CI/CD Pipelines

Yasmeen Syed *

Independent Researcher, USA.

World Journal of Advanced Engineering Technology and Sciences, 2026, 18(01), 415-424

Publication history: Received on 24 November 2025; revised on 26 December 2025; accepted on 30 January 2026

Article DOI: https://doi.org/10.30574/wjaets.2026.18.1.0003

Abstract

This study aims to explore the architectural principles and design patterns required to build scalable, resilient, and secure cloud-native enterprise platforms. It focuses on leveraging modern technologies such as AWS cloud services, Kubernetes orchestration, and CI/CD pipelines. The objective is to identify best practices that enhance system performance, fault tolerance, and operational efficiency. Additionally, the research investigates how enterprises can transition from monolithic systems to microservices-based architectures. The study also emphasizes aligning architectural decisions with business agility and innovation. Ultimately, it seeks to provide a structured framework for designing next-generation enterprise systems.

The research adopts a qualitative and analytical approach by reviewing existing literature, industry case studies, and architectural frameworks. It examines cloud-native patterns such as microservices, containerization, and infrastructure as code. AWS services like EC2, S3, Lambda, and IAM are analyzed in conjunction with Kubernetes components. CI/CD pipeline implementations using tools like Jenkins, GitHub Actions, and ArgoCD are evaluated. Comparative analysis is conducted to assess performance, scalability, and security aspects. Architectural Figures and design models are used to illustrate workflows and system interactions.

The findings demonstrate that cloud-native architectures significantly improve scalability, deployment speed, and system resilience. Kubernetes enhances container orchestration and resource utilization, while AWS provides robust infrastructure and managed services. CI/CD pipelines enable rapid and reliable software delivery with reduced human intervention. Security patterns such as zero-trust architecture and IAM policies strengthen system protection. The integration of these technologies results in highly modular and fault-tolerant systems. Organizations adopting these patterns achieve improved operational efficiency and reduced downtime.

Cloud-native enterprise platforms represent a paradigm shift in system design and deployment strategies. The integration of AWS, Kubernetes, and CI/CD pipelines enables scalable, secure, and automated environments. Organizations must adopt standardized design patterns to fully leverage these technologies. Continuous monitoring, automation, and security integration are critical for long-term success. The study concludes that adopting cloud-native principles enhances innovation and competitive advantage. Future research can explore AI-driven optimization in cloud-native environments.

Keywords: Cloud-Native Architecture; AWS; Kubernetes; CI/CD Pipelines; Microservices; Scalability; Security; DevOps; Containerization; Infrastructure as Code

* Corresponding author: Yasmeen Syed.

1. Introduction to Cloud-Native Architecture

Cloud-native architecture represents a transformative approach to building and running applications that fully exploit the advantages of cloud computing. Unlike traditional architectures, cloud-native systems are designed to operate in distributed, dynamic environments such as public, private, or hybrid clouds. These systems rely heavily on microservices, containers, and orchestration platforms to ensure scalability and resilience. By decoupling application components, organizations can develop, deploy, and scale services independently, significantly improving agility and innovation. This paradigm also aligns with DevOps practices, promoting continuous integration and delivery.

A fundamental aspect of cloud-native architecture is its reliance on containerization technologies such as Docker, which package applications and their dependencies into lightweight, portable units. These containers can run consistently across different environments, eliminating the "it works on my machine" problem. Additionally, orchestration tools like Kubernetes automate deployment, scaling, and management of containerized applications. This ensures optimal resource utilization and system reliability. Cloud-native systems also emphasize declarative APIs, allowing infrastructure and applications to be defined and managed programmatically.

Another critical feature is elasticity, where systems can automatically scale resources up or down based on demand. This capability is particularly important for enterprise platforms that experience fluctuating workloads. Cloud-native architectures leverage managed cloud services to reduce operational overhead and improve efficiency. Furthermore, they integrate observability tools to monitor system health and performance in real time. These characteristics make cloud-native architecture essential for modern enterprises seeking digital transformation.

2. Evolution from Monolithic to Microservices Architecture

Traditional monolithic architectures consist of a single, tightly coupled codebase where all components are interconnected. While this approach simplifies initial development, it becomes increasingly difficult to maintain, scale, and update as the application grows. Any change requires redeploying the entire system, which increases downtime and risk. Moreover, scaling specific components independently is not feasible, leading to inefficient resource usage. These limitations have driven the shift toward microservices architecture in modern enterprise systems.

Microservices architecture breaks down applications into smaller, independent services that communicate via APIs. Each service is responsible for a specific business function and can be developed, deployed, and scaled independently. This modularity enhances fault isolation, meaning that failure in one service does not necessarily impact the entire system. It also enables teams to work in parallel, accelerating development cycles. Technologies such as RESTful APIs and message queues facilitate communication between services.

The transition to microservices also introduces challenges such as increased complexity in service management, data consistency, and network communication. However, these challenges are mitigated through service discovery, API gateways, and container orchestration platforms. Microservices align well with cloud-native principles, enabling organizations to build highly scalable and resilient systems. As enterprises adopt this architecture, they gain flexibility, faster time-to-market, and improved system reliability.

3. Core Components of AWS for Enterprise Platforms

Amazon Web Services (AWS) provides a comprehensive suite of cloud services that form the backbone of cloud-native enterprise platforms. These services are categorized into compute, storage, networking, database, and security components. AWS enables organizations to build scalable and cost-effective applications without investing in physical infrastructure. Services like EC2 provide virtual machines, while S3 offers highly durable object storage. Together, these services support a wide range of enterprise workloads.

Amazon Web Services offers serverless computing through AWS Lambda, allowing developers to run code without managing servers. This reduces operational complexity and improves scalability. AWS also provides managed database services such as RDS and DynamoDB, which ensure high availability and automatic backups. Networking services like VPC enable secure and isolated environments for applications. These components work together to create a robust cloud infrastructure.

Security is a critical aspect of AWS, with services like IAM (Identity and Access Management) controlling user permissions and access levels. AWS also integrates monitoring tools such as CloudWatch to track system performance

and detect anomalies. By leveraging these services, enterprises can build secure, scalable, and resilient platforms. The flexibility of AWS allows organizations to adopt hybrid and multi-cloud strategies. This makes AWS a preferred choice for cloud-native application development.

Table 1 AWS Services and Their Functions

Service	Category	Function	Use Case
EC2	Compute	Virtual servers	Scalable applications
S3	Storage	Object storage	Data backup
Lambda	Serverless	Event-driven compute	Automation
RDS	Database	Managed DB	Transaction systems
IAM	Security	Access control	Identity management

Table 1 presents an overview of essential services offered by Amazon Web Services and their roles in building cloud-native enterprise platforms. It categorizes services into compute, storage, database, serverless, and security domains, representing how each contributes to system functionality. For instance, EC2 enables scalable virtual computing resources, while S3 provides durable object storage for data management. AWS Lambda supports serverless execution, allowing applications to run without infrastructure management, and RDS offers managed relational database services for transactional workloads. IAM ensures secure access control through authentication and authorization mechanisms. Collectively, the table emphasizes how these services integrate to deliver scalability, flexibility, and security in cloud-native architectures.

4. Kubernetes for Container Orchestration

Kubernetes has become the de facto standard for managing containerized applications in cloud-native environments. It automates the deployment, scaling, and management of containers, ensuring high availability and efficient resource utilization. Kubernetes organizes containers into pods, which are the smallest deployable units. These pods run on nodes within a cluster, forming a highly scalable infrastructure. The control plane manages the overall system, ensuring that desired states are maintained.

One of Kubernetes' key features is its ability to perform automatic scaling based on resource usage. Horizontal Pod Autoscaling adjusts the number of pods in response to traffic demand. Additionally, Kubernetes provides self-healing capabilities by automatically restarting failed containers and rescheduling them on healthy nodes. This ensures minimal downtime and improved system reliability. Load balancing and service discovery further enhance application performance and accessibility.

Kubernetes also integrates seamlessly with cloud providers like AWS, enabling hybrid deployments and multi-cloud strategies. It supports declarative configuration through YAML files, allowing developers to define infrastructure as code. This simplifies deployment and version control processes. Moreover, Kubernetes enhances security through role-based access control (RBAC) and network policies. As a result, it plays a crucial role in modern cloud-native architectures.

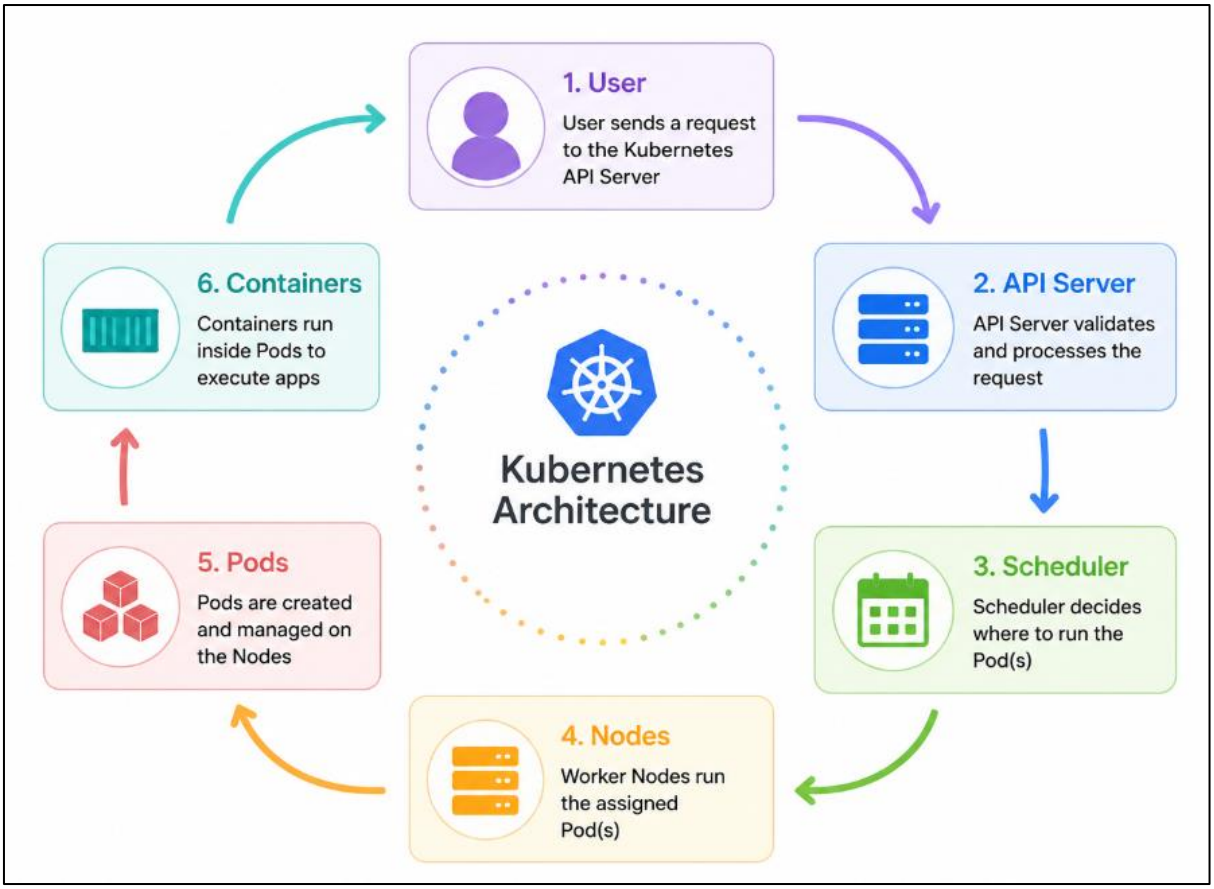

Figure 1 Kubernetes Architecture

The Kubernetes architecture diagram (Figure 1) shows how the control plane manages worker nodes. The API Server is the main entry point for user requests, and the Scheduler selects the best node to run workloads. Worker nodes execute applications inside Pods, which contain one or more containers. The Controller Manager ensures the system stays in the desired state, while etcd stores cluster data. The diagram follows a top-down flow: requests enter through the API Server, are scheduled, and run inside Pods on worker nodes. Arrows indicate communication between components, highlighting Kubernetes' automated orchestration.

5. CI/CD Pipelines in Cloud-Native Systems

CI/CD (Continuous Integration and Continuous Deployment) pipelines play a key role in automating how software is built and deployed in cloud-native systems. They allow developers to frequently update code, run automated tests, and deploy applications reliably. By handling repetitive tasks automatically, CI/CD reduces mistakes and speeds up development. This is especially useful in microservices architectures, where different services are updated separately.

A CI/CD pipeline usually includes stages like code commit, build, test, deployment, and monitoring. Tools such as Jenkins, GitHub Actions, and Argo CD help automate these steps by connecting with version control systems. Automated testing ensures the code works correctly before deployment. Techniques like blue-green and canary deployments reduce downtime and lower the risk of failures, leading to a better user experience.

Security is now an important part of CI/CD pipelines through DevSecOps practices. This includes automated security checks, vulnerability scanning, and code analysis. Adding security early in the pipeline helps detect problems sooner. CI/CD pipelines also allow quick rollback if something goes wrong, making systems more reliable. Overall, they are essential for modern cloud-native development.

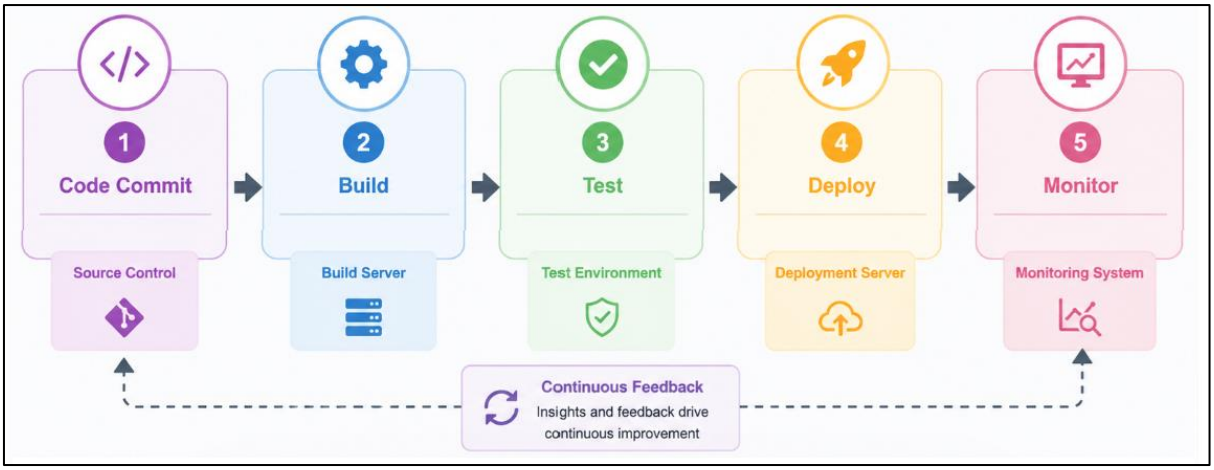

Figure 2 CI/CD Pipeline

The CI/CD pipeline Figure 2 represents the automated software delivery lifecycle. It begins with code commit in a version control system, followed by the build stage, where the application is compiled. Next is the testing phase, which includes automated unit and integration tests. After successful testing, the application moves to the deployment stage, where it is released to staging or production environments. Finally, monitoring and feedback loops ensure continuous improvement. The Figure should be drawn as a linear or cyclic flow with arrows connecting each stage, emphasizing automation, continuous feedback, and iterative development.

6. Design Patterns for Scalability

Scalability is a fundamental requirement for enterprise platforms, and cloud-native architectures provide several design patterns to achieve it. Horizontal scaling is one of the most common approaches, where additional instances of a service are deployed to handle increased load. This is supported by load balancers that distribute traffic evenly across instances. Auto-scaling groups dynamically adjust resources based on demand, ensuring optimal performance and cost efficiency.

Another important pattern is the use of stateless services, where application instances do not store session data locally. Instead, data is stored in external databases or distributed caches. This allows instances to be added or removed without affecting system functionality. Content Delivery Networks (CDNs) are also used to distribute content globally, reducing latency and improving user experience. These patterns ensure that systems can handle high traffic volumes efficiently.

Event-driven architecture is another scalability pattern that enables asynchronous communication between services. By using message queues and event streams, systems can process tasks independently and scale components as needed. This reduces bottlenecks and improves system responsiveness. Additionally, database sharding and replication techniques enhance data scalability. By combining these patterns, cloud-native platforms achieve high performance, resilience, and scalability.

7. Security Design Patterns in Cloud-Native Platforms

Security in cloud-native platforms is a foundational requirement rather than an afterthought. Modern enterprise systems operate in distributed and dynamic environments, making them more vulnerable to cyber threats. Cloud-native security design patterns emphasize a “security-first” approach, integrating protection mechanisms at every layer of the architecture. One of the most widely adopted models is the zero-trust architecture, which assumes that no entity inside or outside the network should be trusted by default. Every request must be authenticated, authorized, and encrypted before access is granted.

Identity and access management plays a crucial role in enforcing security policies. Services like AWS Identity and Access Management allow fine-grained control over user permissions, ensuring that only authorized users can access specific resources. Role-based access control (RBAC) and least privilege principles are commonly implemented to minimize security risks. Encryption is another critical component, both at rest and in transit, using protocols such as TLS and AES. These measures protect sensitive data from unauthorized access and breaches.

Additionally, cloud-native platforms employ security automation through DevSecOps practices. Automated vulnerability scanning, intrusion detection systems, and compliance checks are integrated into CI/CD pipelines. Network security policies, firewalls, and service mesh further enhance protection by controlling traffic flow between services. Continuous monitoring and logging help detect anomalies in real time. By combining these strategies, organizations can build robust and secure cloud-native systems capable of withstanding evolving cyber threats.

8. Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a key enabler of cloud-native architecture, allowing infrastructure to be provisioned and managed using code rather than manual processes. This approach enhances consistency, repeatability, and scalability in system deployment. Tools like Terraform and AWS CloudFormation enable developers to define infrastructure configurations in declarative or imperative formats. These configurations can be version-controlled, tested, and reused across environments.

IaC eliminates configuration drift, a common issue in traditional infrastructure management where environments become inconsistent over time. By using code to define infrastructure, organizations ensure that development, testing, and production environments are identical. This reduces deployment errors and improves system reliability. Additionally, IaC supports automated provisioning, enabling rapid scaling of resources in response to demand. It also integrates seamlessly with CI/CD pipelines, further enhancing automation.

Another advantage of IaC is improved collaboration between development and operations teams. Infrastructure definitions can be reviewed, modified, and deployed using standard software development practices. This aligns with DevOps principles, fostering a culture of shared responsibility. Security can also be embedded into IaC templates through policy-as-code frameworks. Overall, IaC is essential for managing complex cloud-native environments efficiently and securely.

9. Observability and Monitoring

Observability plays a crucial role in keeping cloud-native systems running smoothly. In simple terms, it helps teams understand what's happening inside an application by looking at what it produces like data, logs, and system behavior. Since modern applications are often built using microservices and run across distributed environments, it's not always easy to pinpoint problems just by looking at the surface. That's where observability becomes essential it gives engineers a clear window into system health and performance.

At the heart of observability are three key components: metrics, logs, and traces. Metrics are numerical values that show how a system is performing over time, such as CPU usage, memory consumption, or request latency. Logs, on the other hand, are detailed records of events that happen within the system like errors, warnings, or user activities. Then there are traces, which follow a request as it travels through different services in a distributed system. When you combine these three, you get a complete picture of what's going on, making it much easier to diagnose issues.

To make observability practical, organizations rely on powerful tools. For example, Prometheus is widely used for collecting and storing metrics, while Amazon CloudWatch provides a comprehensive solution for monitoring cloud resources and applications. These tools allow teams to track performance in real time, set up alerts, and visualize data through dashboards. Engineers can quickly spot unusual patterns like sudden spikes in latency or error rates and take action before users are affected. For distributed systems, tracing tools like Jaeger help track how requests move between services, making it easier to identify bottlenecks or failures.

But observability isn't just about collecting data it's about using that data effectively. Teams use logs and traces to perform root cause analysis when something goes wrong. Instead of guessing, they can trace the exact path of a failure and fix it faster. This is especially important in large-scale systems where a small issue in one service can impact many others.

Another growing trend in observability is the use of machine learning. Advanced systems can now automatically detect unusual behavior, predict potential failures, and even suggest solutions. This proactive approach helps organizations prevent issues before they escalate, rather than just reacting to them after the fact.

10. Resilience and Fault Tolerance Mechanisms

Resilience and fault tolerance are essential for ensuring that cloud-native systems remain operational despite failures. In distributed environments, failures are inevitable due to network issues, hardware faults, or software bugs. Cloud-native architectures address this challenge through various design patterns that enhance system reliability. One such pattern is the circuit breaker, which prevents cascading failures by stopping requests to a failing service.

Retry mechanisms and timeouts are also commonly used to handle transient failures. These techniques allow systems to recover automatically without human intervention. Load balancing distributes traffic across multiple instances, ensuring that no single component becomes a bottleneck. Additionally, failover strategies redirect traffic to backup systems in case of failure. These mechanisms ensure continuous availability of services.

Another important aspect is redundancy, where multiple instances of critical components are deployed across different regions or availability zones. This geographic distribution minimizes the impact of localized failures. Chaos engineering practices are also employed to test system resilience by intentionally introducing failures. By implementing these strategies, organizations can build robust systems capable of maintaining high availability and performance under adverse conditions.

11. DevSecOps Integration

DevSecOps is an evolution of DevOps that integrates security practices into every stage of the software development lifecycle. Instead of treating security as a separate phase, DevSecOps embeds it within CI/CD pipelines, ensuring continuous security validation. This approach reduces vulnerabilities and improves compliance with regulatory standards. Security testing tools are automated to scan code, dependencies, and configurations for potential risks.

CI/CD tools such as Jenkins and GitHub Actions support the integration of security checks into development workflows. These tools enable automated testing, code analysis, and deployment processes. Security policies are enforced through automated gates, preventing insecure code from reaching production. This proactive approach significantly reduces the risk of security breaches.

DevSecOps also promotes collaboration between development, operations, and security teams. By sharing responsibilities, organizations can address security issues early in the development cycle. Continuous monitoring and feedback loops ensure that security measures remain effective over time. Compliance requirements are automatically enforced, reducing manual effort. Overall, DevSecOps enhances both security and efficiency in cloud-native systems.

Table 2 DevOps vs DevSecOps

Aspect	**DevOps**	**DevSecOps**
Focus	Speed	Security + Speed
Security	Post-development	Integrated
Automation	CI/CD	CI/CD + Security
Risk Handling	Reactive	Proactive

Table 2 compares DevOps and DevSecOps, focusing on their approach to software development, deployment, and security integration. DevOps primarily emphasizes speed, collaboration, and automation through continuous integration and continuous deployment pipelines, enabling rapid delivery of applications. In contrast, DevSecOps extends this model by embedding security practices throughout the development lifecycle, ensuring that vulnerabilities are identified and mitigated early. While DevOps treats security as a post-development activity, DevSecOps adopts a proactive approach by integrating automated security testing, compliance checks, and risk management into CI/CD workflows. The table highlights the evolution from reactive to proactive security, demonstrating how DevSecOps enhances both system reliability and protection in modern cloud-native environments.

12. Future Trends in Cloud-Native Architectures

Cloud-native architectures continue to evolve with emerging technologies and industry trends. One significant trend is the adoption of serverless computing, where developers focus solely on writing code while the cloud provider manages infrastructure. Services like AWS Lambda are driving this shift, enabling cost-efficient and scalable application development. Another trend is the integration of artificial intelligence and machine learning into cloud-native systems for predictive analytics and automation.

Edge computing is also gaining traction, bringing computation closer to data sources to reduce latency and improve performance. This is particularly important for applications such as IoT, autonomous vehicles, and real-time analytics. Multi-cloud and hybrid cloud strategies are becoming increasingly popular, allowing organizations to avoid vendor lock-in and enhance system resilience. These approaches enable seamless integration across different cloud platforms.

Furthermore, advancements in service mesh technologies and API management are improving communication between microservices. Security innovations such as confidential computing and zero-trust frameworks are enhancing data protection. Automation and self-healing systems are becoming more sophisticated, reducing the need for manual intervention. As these trends continue to develop, cloud-native architectures will play a central role in shaping the future of enterprise IT.

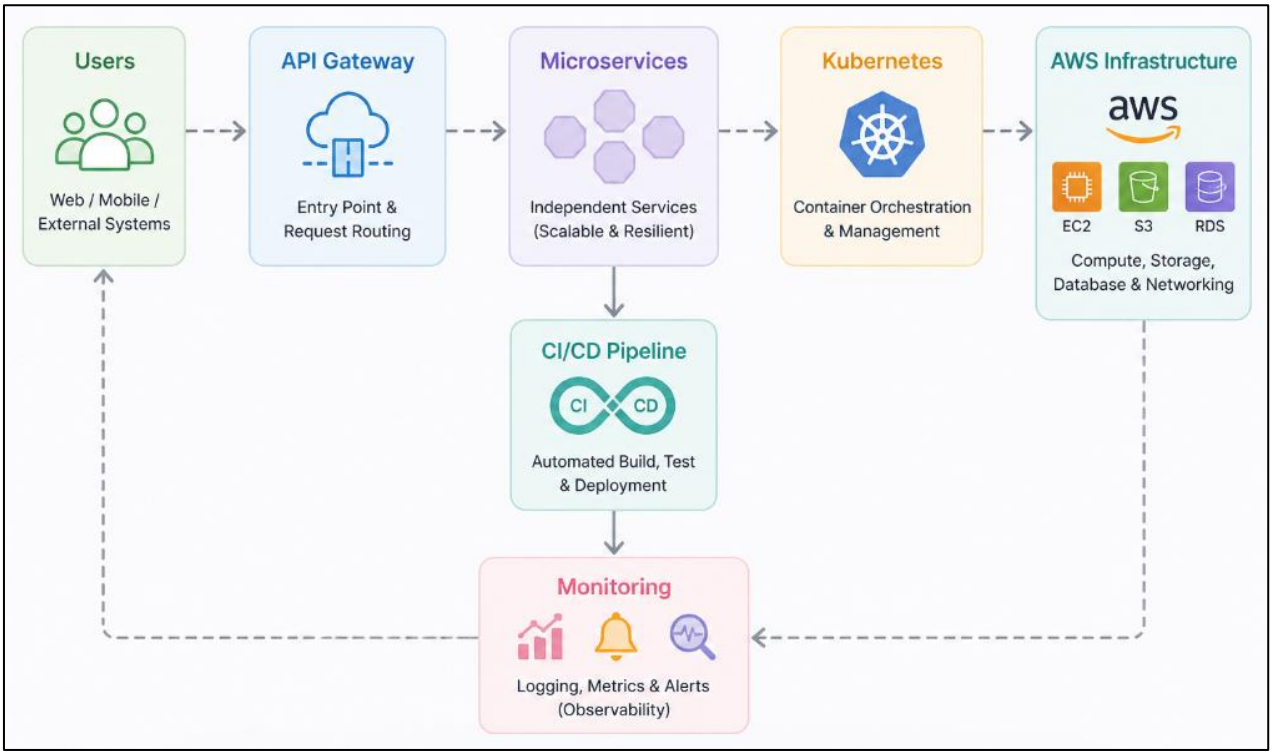

Figure 3 Cloud-Native Ecosystem

The cloud-native ecosystem Figure 3 shows how different components interact in a modern enterprise platform. At the front end, users access the system through an API Gateway, which routes requests to various microservices. These services are deployed and managed within a Kubernetes cluster, which runs on cloud infrastructure such as Amazon Web Services. Supporting layers include the CI/CD pipeline for continuous delivery and a monitoring system for observability. The Figure should depict a layered or flow-based structure with arrows showing request flow from users to backend services, and supporting systems connected below to indicate automation and monitoring.

13. Conclusion

The research on cloud-native enterprise platforms highlights a significant paradigm shift in how modern applications are designed, deployed, and managed. By integrating microservices architecture, containerization, and dynamic orchestration, organizations can achieve unprecedented levels of scalability and flexibility. The adoption of cloud

platforms such as Amazon Web Services enables enterprises to leverage on-demand resources, reducing infrastructure costs while improving performance. Furthermore, Kubernetes-based orchestration ensures efficient container management, enhancing system reliability and fault tolerance. These advancements collectively contribute to the evolution of highly resilient and adaptive enterprise systems.

Another critical outcome of this study is the role of CI/CD pipelines in accelerating software delivery and maintaining system consistency. Automation of build, test, and deployment processes minimizes human error and enhances productivity. The integration of DevSecOps practices ensures that security is embedded throughout the development lifecycle, addressing vulnerabilities proactively. Infrastructure as Code further strengthens this ecosystem by enabling consistent and repeatable deployments. Together, these technologies create a cohesive framework that supports continuous innovation and rapid adaptation to changing business requirements.

Security and resilience emerge as foundational pillars in cloud-native architectures. The implementation of zero-trust models, identity and access management, and encryption techniques ensures robust protection against cyber threats. Additionally, observability and monitoring tools provide real-time insights into system performance, enabling proactive issue resolution. Fault tolerance mechanisms such as circuit breakers, failover strategies, and redundancy enhance system availability. These practices ensure that enterprise platforms can maintain high performance even under adverse conditions, thereby improving user experience and operational stability.

In conclusion, cloud-native architecture, supported by AWS, Kubernetes, and CI/CD pipelines, offers a comprehensive solution for building scalable, secure, and efficient enterprise platforms. Organizations that adopt this design patterns gain a competitive advantage through improved agility, faster time-to-market, and enhanced system reliability. As emerging technologies such as AI, edge computing, and serverless architectures continue to evolve, cloud-native systems will remain at the forefront of digital transformation. Future research can further explore intelligent automation, sustainability in cloud computing, and advanced security frameworks to enhance the capabilities of cloud-native enterprise systems.

References

[1] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. IEEE Softw. 33, 3 (May 2016), 42–52. https://doi.org/10.1109/MS.2016.64

[2] Burns B, Grant B, Oppenheimer D, Brewer E, and Wilkes J Borg, omega, and kubernetes: lessons learned from three container-management systems over a decade *Queue* 2016, 14(1):70-93. https://doi.org/10.1145/2898442.2898444

[3] Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R. and Safina, L. (2017) Microservices: Yesterday, Today, and Tomorrow. In: Mazzara, M. and Meyer, B., Eds., Present and Ulterior Software Engineering, Springer, 195-216.

[4] Merkel, D. (2014) Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, 2014, No. 2.

[5] C. Pahl, "Containerization and the PaaS Cloud" in IEEE Cloud Computing, vol. 2, no. 03, pp. 24-31, May-June 2015, doi: 10.1109/MCC.2015.51.

[6] Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., Zambrano, A., & Lang, M. (2016). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In *Proceedings - 2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2016* (pp. 179-182). Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/CCGrid.2016.37

[7] Chen, L. (2018). Microservices: Architecting for continuous delivery and DevOps. IEEE International Conference on Software Architecture, 39–48.

[8] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. IEEE Software, 35(3), 24–35. DOI:10.1109/MS.2018.2141039

[9] Kang, H., Le, M., & Tao, S. (2016). Container and microservice driven design for cloud infrastructure DevOps. IEEE International Conference on Cloud Engineering, 202–211. DOI:10.1109/IC2E.2016.26

[10] Kratzke, N., & Quint, P. C. (2017). Understanding cloud-native applications after 10 years of cloud computing. Journal of Systems and Software, 126, 1–16. DOI: https://doi.org/10.1016/j.jss.2017.01.001

[11] Pahl, C., Jamshidi, P., & Zimmermann, O. (2018). Architectural Principles for Cloud Software. *ACM Transactions on Internet Technology (TOIT), 18*, 1 - 23.

[12] Newman, S. (2015). Building microservices: Designing fine-grained systems. O'Reilly Media.

[13] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7–18.

[14] Lewis, J. and Fowler, M. (2014) Microservices, a Definition of This New Architectural Term.

[15] Turnbull, J. (2014). The Docker book: Containerization is the new virtualization. James Turnbull Publishing. First Edition.