JGTAI
Journal of Global Trends in Artificial Intelligence

Volume 1, No. 1, June 2025-2026
Available online at: www.jgtai.com

Cloud-Native Software Platforms: A Survey of Intelligent Resource Management and Reliability

Mr. Ram Pratap Singh
Department of Computer Science and Engineering
Lakshmi Narain College of Technology
Bhopal
rampratap@lnct.ac.in
rampratap.mca2011@gmail.com

***Abstract*—Cloud-native software platforms have revolutionized modern application development by leveraging microservices, containerization, Kubernetes orchestration, Continuous Integration/Continuous Deployment (CI/CD), serverless computing, and Infrastructure as Code (IaC). These technologies enable scalable, resilient, and efficient application deployment while supporting dynamic resource management across distributed cloud environments. However, the increasing complexity of cloud-native systems introduces challenges related to resource allocation, workload scheduling, reliability, fault tolerance, observability, and operational efficiency. This survey presents a comprehensive review of intelligent resource management and reliability techniques in cloud-native software platforms. It examines cloud-native architectures, resource allocation, monitoring, scheduling, auto-scaling, load balancing, AI/ML-based resource optimization, and performance enhancement strategies. The survey also reviews reliability mechanisms, including fault tolerance, high availability, self-healing, failure recovery, observability, and performance evaluation. Furthermore, it provides a comparative analysis of recent studies, identifies existing research challenges, and discusses future research directions toward developing intelligent, autonomous, scalable, secure, and highly reliable cloud-native software platforms.**

Keywords—Cloud-Native Platforms, Intelligent Resource Management, AI-Driven Resource Scheduling, Kubernetes Resource Optimization.

I. INTRODUCTION

Cloud-native software platforms have emerged as a transformative paradigm for developing, deploying, and managing modern distributed applications by leveraging microservices, containerization, Kubernetes orchestration, Continuous Integration/Continuous Deployment (CI/CD), serverless computing, and Infrastructure as Code (IaC). These technologies enable organizations to build scalable, resilient, and agile applications capable of adapting to dynamic workloads while improving operational efficiency and accelerating software delivery [1]. The widespread adoption of cloud-native platforms across domains such as healthcare, finance, e-commerce, telecommunications, and the Internet of Things (IoT) has significantly increased the demand for intelligent resource management and high system reliability to ensure continuous service availability and optimal resource utilization[2][3].

Efficient resource management is a fundamental requirement for cloud-native environments, where applications consist of loosely coupled microservices running in containerized infrastructures [4][5]. Technologies such as Kubernetes provide automated workload scheduling, dynamic resource allocation, horizontal auto-scaling, load balancing, and self-healing mechanisms that optimize infrastructure utilization while maintaining QoS and SLAs [6][7][8]. However, the distributed and dynamic nature of cloud-native platforms introduces challenges related to resource contention, performance variability, fault tolerance, security, observability, and operational complexity [9][10]. To address these challenges, modern cloud-native systems increasingly incorporate AI, ML, predictive analytics, and AIOps for intelligent resource provisioning, anomaly detection, predictive scaling, failure prediction, and automated system recovery[11].

This survey presents a comprehensive review of intelligent resource management and reliability techniques in cloud-native software platforms. It discusses the evolution of cloud-native architectures, including microservices, containerization, orchestration frameworks, serverless computing, and DevOps practices, together with AI-driven resource optimization and reliability enhancement mechanisms. This survey primarily aims to summarize the following:

- This survey presents a comprehensive overview of cloud-native software platforms, including cloud-native architecture, microservices, containerization, Kubernetes, serverless computing, and Infrastructure as Code (IaC).
- The survey reviews intelligent resource management techniques, including workload scheduling, container orchestration, auto-scaling, load balancing, AI-driven resource allocation, and performance optimization approaches.
- This work examines reliability enhancement mechanisms such as fault tolerance, self-healing, observability, distributed tracing, monitoring, logging, and resilience engineering for maintaining highly available cloud-native applications.
- The survey provides a comparative analysis of recent studies on intelligent resource management and reliability, highlighting state-of-the-art techniques, their advantages, limitations, and existing research gaps.
- The survey identifies key challenges and discusses future research directions for developing intelligent, autonomous, scalable, secure, and reliable cloud-native software platforms.

The remainder of this paper is organized as follows. Section II presents the architecture of cloud-native software platforms. Section III reviews intelligent resource management techniques. Section IV discusses reliability and resilience mechanisms. Section V presents a comparative review of recent studies. Finally, Section VI concludes the paper with key findings and future research perspectives.

II. Cloud-Native Software Platform Architecture

The architecture of cloud-native software platforms is a cutting-edge method for creating, releasing, and overseeing distributed applications that are integral to the cloud. To create scalable, resilient, and portable applications, cloud-native platforms use DevOps approaches, containerization, Kubernetes orchestration, microservices, and the cloud, as opposed to traditional monolithic designs. These platforms help companies automate application deployment, optimize resource usage, and maintain high availability and the capacity for continuous innovation. In addition, cloud-native architectures integrate CI/CD pipelines, cloud-native service mesh, observability tools and IaC to drive operational efficiency, simplify infrastructure management and build system reliability [12]. The cloud-native software platform architecture can be segmented into cloud-native computing fundamentals, microservices and containerization, Kubernetes and container orchestration, and CI/CD using IaC software, based on literature studied.

A. Cloud-Native Computing Fundamentals

Cloud-native computing is a pattern of software development and deployment that takes full advantage of cloud environments with scalable, resilient, loosely coupled architecture. It focuses on the automation, elasticity, continuous delivery, and portability aspects through technologies like containers, microservices, orchestration platforms, and declarative infrastructure management [13]. Cloud-native systems offer quicker deployment cycles, higher resource efficiency, and more agile operations than traditional application architectures [14]. The Cloud Native Computing Foundation (CNCF) outlines four fundamental principles of containerization, microservices, declarative APIs, and automation, that enable organisations to quickly create and operate highly available cloud applications. These principles are used to build intelligent resource management and reliable cloud-native software platforms [15].

B. Microservices and Containerization

The fundamental building blocks of cloud-native software platforms are microservices and containerization. The microservice architecture breaks down large applications into small, standalone services that can be developed, deployed, and scaled independently, enhancing flexibility, fault isolation, and maintainability [16]. Containers bundle applications, their dependencies, and other resources into a package that can be deployed and run anywhere. Docker containers are lightweight and portable, and Docker containers can be deployed to ease the deployment of applications, and container registries enable efficient images distribution. Together with microservices, containers deliver much better consistency during deployment, scalability, resource efficiency, and resilience of applications, and are indispensable to modern cloud-native platforms.

C. Kubernetes and Container Orchestration

The growing adoption of cloud-native applications, which are deployed across distributed infrastructures, has made efficient and effective container orchestration a critical piece in the demand for application deployment, scaling, networking, and fault recovery [17]. Kubernetes has become the de facto orchestration platform through automation of container scheduling, workload distribution, service discovery, load balancing, rolling updates, and self-healing. It arranges each container in a Pod and controls each Pod via a control plane and worker nodes, which ensures applications are always highly available even under fluctuating workloads [18]. Kubernetes also has intelligent resource allocation and horizontal auto-scaling functionality, which allows cloud-native platforms to deliver high performance while maximizing resource utilization.

III. Intelligent Resource Management

Effective and efficient management of resources is one of the core features of a cloud-native software platform that supports the effective use of resources while keeping application performance, scalability and reliability intact. Cloud-native apps are made up of distributed microservices deployed across a fluid infrastructure that is managed by Kubernetes [19]. Therefore, resource management is not just about provisioning but involves real-time monitoring, intelligent scheduling, auto-scaling, load balancing, AI-driven optimization, and performance tuning. Today's cloud-native platforms embed observability capabilities, orchestration frameworks, and Artificial Intelligence (AI) capabilities to automatically adjust resource allocations based on the nature of the workload, the service-level goals, and the infrastructure environment.

A. Resource Allocation, Monitoring and Scheduling

Resource Allocation, monitoring, and scheduling are essential to ensuring that cloud-native applications have access to the right computing resources, and that they are effectively used, delivering high utilization, low latency, and reliable service guarantees [20]. Kubernetes and other orchestration platforms keep monitoring the available resources, the infrastructure and the workload demand to allocate containers and microservices to the best possible location. Real-time monitoring systems also help with scheduling decisions by giving real-time visibility on CPU usage, memory usage, storage occupancy, network usage, and application performance. These mechanisms contribute to increased scalability, minimized resource contention, and effective cloud-native functionality.

1) Resource Allocation

Resource allocation is the process of allocating resources like CPU, memory, storage, GPUs and network bandwidth for containerized applications based on workload demand. Cloud-native platforms, as opposed to the traditional static allocation approach, dynamically allocate resources using Kubernetes schedulers, container orchestration frameworks and declarative resource specifications [21]. Resource requests, limits, namespaces, and quotas are used to distribute resources fairly among multiple microservices without causing resource starvation.

2) Resource Monitoring

Resource monitoring continuously monitors infrastructure resources, application performance, and system health to

enable smart infrastructure management decisions. Observability frameworks are used in cloud-native environments to gather metrics, logs, traces, and events from a distributed set of microservices. Tools like Prometheus, Grafana, OpenTelemetry, Loki and Jaeger give real-time insights into CPU usage, memory consumption, network throughput, storage usage, container health, and application latency.

3) Resource Scheduling

Resource scheduling is the process of deciding the location and timing of containers, pods and application workloads against the available computing resources [22]. Prior to assigning the Pods to the worker nodes, Kubernetes scheduling algorithms consider the capacity of the nodes, availability of resources, affinity rules, taints, tolerations, priority classes, and workload constraints. Previous to that, in today's cloud-native scheduling, the rule-based approach has been extended by using multi-objective optimization, energy-aware scheduling, QoS-aware scheduling and AI-assisted scheduling algorithms [23]. Heterogeneous scheduling for CPU-GPU clusters, edge-cloud scheduling, serverless scheduling, and predictive scheduling via workload forecasting are also explored in recent research.

B. Auto-Scaling and Load Balancing

Auto-scaling and load balancing capabilities allow cloud-native platforms to dynamically adjust resources based on workload fluctuations and keep applications available and responsive [24]. Auto-scaling automatically adjusts computing resources based on the demand of the application, while load balancing distributes incoming requests between multiple instances of the service to avoid overloading the service and to increase the response time.

1) Auto-Scaling

Auto-scaling elastically adjusts the number of application replicas and computing resources based on workload fluctuations. Kubernetes also provides HPA, VPA, and Cluster Autoscaler for managing application scalabilities[25].

Compared to the traditional threshold-based approach, intelligent auto-scaling uses predictive analytics, reinforcement learning, LSTM models, and workload forecasting to proactively allocate resources before they are needed, preventing performance degradation. AI-driven auto-scaling significantly improves resource utilization, reduces operational costs, and enhances application responsiveness under dynamic workloads.

2) Load Balancing

Load balancing distributes application requests across multiple containers or Pods or across microservices to maximize throughput and minimize response time. Kubernetes Services, Ingress Controllers, Service Mesh technologies, and cloud-native load balancers provide for workload distribution, and for service discovery, traffic routing, and fault isolation.

AI-driven decision-making, real-time monitoring, and workload forecasting are combined in modern adaptive load balancing to intelligently reroute traffic based on resource usage and application status. Intelligent load balancing decreases latency, enhances fault tolerance, to stop resource hotspots and plays a vital role in cloud-native reliability [26].

C. AI/ML-Based Resource Optimization

AI and ML have become essential technologies for intelligent resource optimization in cloud-native software platforms. AI/ML-based methods leverage past workload trends, infrastructure metrics, and application performance to guide intelligent resource allocation decisions—an approach that differs from traditional rule-of-thumb methods [27]. Cloud-native environments produce a significant amount of operational data from microservices, containers, Kubernetes clusters and monitoring systems that allows for predictive allocation of resources, intelligent scheduling of workloads, anomaly detection, and capacity planning [28]. Therefore, ML models such as supervised learning, reinforcement learning, and DL are capable of predicting the resource demand more accurately and optimizing the utilization of CPU, memory, storage and network [29].

Additionally, AI optimization is embedded in Kubernetes orchestration and AIOps features to include predictive auto-scaling, intelligent placement of containers, workload migration, and automated fault detection [30][31]. The techniques dynamically rebalance resources based on fluctuations in workload, minimizing resource waste and enhancing system scalability, Quality of Service (QoS), Service Level Agreement (SLA) adherence, and reliability. This means that with AI/ML resource optimization, a cloud-native software platform can provide autonomous, efficient, and highly scalable resource management.

IV. Reliability and Resilience Techniques

Reliability and resilience are essential properties of cloud-native software platforms that guarantee that applications are available, responsive, and resilient to failures and varying workloads, as well as to infrastructure failures. Cloud-native applications run in distributed microservices spread across containerized environments, where failure of hardware, software, network, and contention of resources are predictable [32]. As a result, today's cloud native platforms incorporate fault-tolerant design, high-availability systems, automatic self-healing, intelligent failure recovery and extensive observability to ensure ongoing service operation. Together, these technologies—such as Kubernetes, service mesh, distributed monitoring, and AI-powered operational analytics—can help detect failures in advance, trigger automatic remediation, and intelligently manage resources. The surveyed literature suggests that reliability and resilience techniques can be grouped into fault tolerance and high availability, self-healing and failure recovery, observability, and reliability metrics with performance evaluation.

A. Fault Tolerance and High Availability

In cloud native software platforms, fault tolerance and high availability are fundamental principles for ensuring reliable operation of software systems under component failure. Because a cloud-native application is spread out across clusters of containerized resources, a failure can happen at the infrastructure level, application level, container level, or network level. Modern cloud-native offerings, then, include redundancy, replication, failover, load balancing and intelligent orchestration to reduce service downtime and keep applications always available. Kubernetes is a key component that automatically deploys the application in the desired state, restarts failed containers, and redistributes workloads. Fault tolerance and high availability work together to increase

operational continuity, minimize downtime, and maintain QoS in dynamic workload scenarios.

1) Fault Tolerance

Fault tolerance is the ability of the cloud-native software platform to still provide services when hardware failure, software bugs, container crashes, network problems or outages of the infrastructure occur. Fault-tolerant systems are intended to detect, isolate and mitigate failures, not avoid them outright, but to keep failures from impacting application availability. Redundancy, replication, checkpointing, circuit breakers, retry, and distributed consensus are all used in the cloud-native platforms to ensure continuous availability. Kubernetes also supports fault tolerance with some other capabilities such as ReplicaSets, StatefulSets, health checks and Pod replacement, which automatically restarts new Pods when existing ones fail.

2) High Availability

The goal of high availability is to keep cloud-native applications up and running with minimal downtime even if components fail or when scheduled for maintenance. High availability is accomplished by using redundancy, load balancing, failover, distributed storage, multi-zone deployments, and container orchestration on cloud-native platforms [33]. Service mesh technologies offer intelligent traffic routing and failover mechanisms, and Kubernetes automatically manages application health and redistributing workloads across healthy nodes on failure. The load balancer sends a client request to one of several instances of the application to avoid resource contention and reduce response time.

B. Self-Healing and Failure Recovery

Self-healing and failure recovery techniques allow cloud-native software platforms to autonomously identify failures, recover impacted services, and resume normal operations with minimal human involvement. Cloud-native environments use Kubernetes controllers and orchestration frameworks, anomaly detection, and automated remediation workflows to keep applications healthy continuously, without relying on manual troubleshooting. Such mechanisms can help achieve these objectives, minimizing service disruptions, optimizing operations, and strengthening system resilience in highly dynamic distributed environments.

1) Self-Healing

Self-healing is the ability of the cloud-native software platforms to detect abnormal system behaviour and reestablish healthy states of the applications without manual involvement. Kubernetes regularly checks the health of containers using liveness and readiness probes, and automatically restarts Pods that have failed, replaces unhealthy containers, and reschedules workloads to healthy nodes [34]. Modern self-healing frameworks take these features one step further with integration of an Anomaly Detection component that uses AI, a Policy component that dictates remediation, and an Observability component like Prometheus and Grafana, enabling automated remediation based on real-time operational data. Intelligent self-healing solves Mean Time to Recovery (MTTR), lowers operational complexity and keeps services running in dynamic workload and infrastructure.

2) Failure Recovery

Failure recovery is about the restoration of application functionality after failure has occurred, to reduce the downtime and maintain the consistency of the application. Cloud-native software platforms use features such as automated failover, workload migration, checkpoint restoration, backup and recovery, distributed data replication, and disaster recovery to quickly restore impacted services [35]. Controllers continually compare the state of the applications against their desired configuration, and service mesh technology can reroute traffic from service instances that are down. Furthermore, AI-powered predictive maintenance and failure prediction models enhance recovery efficiency by proactively anticipating potential failures before they cause service disruption, optimizing recovery operations and ensuring platform reliability.

C. Observability (Monitoring, Logging, and Tracing)

Observability offers a holistic view of the internal functioning of cloud-native software platforms, gathering and analyzing operational data from distributed applications. Observability is a more comprehensive approach than traditional monitoring, using metrics, logs, traces and events to enable proactive anomaly detection, root cause analysis (RCA), performance tuning and reliability management. Cloud-native platforms increasingly integrate Prometheus, Grafana, OpenTelemetry, Jaeger, and Loki to provide end-to-end visibility across microservices, containers, Kubernetes clusters, and distributed infrastructure. These capabilities enable operators and intelligent management systems to rapidly identify failures, optimize resource utilization, and maintain high system reliability.

1) Monitoring

System health is assessed through the continuous collection of infrastructure and application metrics through monitoring. These metrics include CPU utilization, memory consumption, network throughput, response time, error rates, and resource availability. Cloud-native platforms utilize Prometheus, Grafana, Kubernetes Metrics Server, and cloud-native dashboards to provide real-time operational insights. Continuous monitoring supports proactive anomaly detection, intelligent auto-scaling, capacity planning, SLA compliance, and automated remediation by supplying accurate runtime information for operational decision-making.

2) Logging

Logging records detailed information about application events, system activities, exceptions, and operational behavior throughout the cloud-native environment. Centralized logging platforms such as Loki, Elasticsearch, Fluentd, and Kibana aggregate logs from distributed microservices, enabling developers and operators to investigate failures, audit system behavior, and diagnose application errors efficiently. Structured logging combined with AI-assisted log analytics further improves anomaly detection, root-cause analysis, and security monitoring in large-scale cloud-native deployments.

3) Tracing

Tracing follows the execution path of user requests as they propagate across multiple microservices, containers, and cloud services. Distributed tracing platforms such as OpenTelemetry and Jaeger record request flows, latency measurements, dependency relationships, and service interactions, allowing developers to identify bottlenecks, communication failures, and latency hotspots. Tracing significantly enhances debugging, performance optimization, and fault localization in highly distributed cloud-native applications by providing end-to-end visibility into complex service interactions [36].

D. Reliability Metrics and Performance Evaluation

Reliability metrics and performance evaluation provide quantitative measures for assessing the effectiveness of cloud-native software platforms under varying workload conditions and operational failures. These metrics help evaluate system availability, fault tolerance, recovery efficiency, scalability, and overall service quality while supporting continuous optimization and SLA compliance. Modern cloud-native environments combine infrastructure-level measurements with application-level performance indicators and AI-driven operational analytics to provide comprehensive reliability assessment.

1) Reliability Metrics

Reliability metrics quantify the ability of cloud-native software platforms to provide continuous and dependable service. Common metrics include availability, Mean Time Between Failures (MTBF), Mean Time to Recovery (MTTR), fault recovery rate, service uptime, failure rate, error rate, and SLA compliance. Recent cloud-native research also considers prediction stability, decision consistency, resilience indicators, and anomaly detection accuracy as complementary measures for evaluating intelligent cloud-native systems. These metrics assist organizations in identifying reliability bottlenecks, optimizing recovery mechanisms, and improving long-term operational stability [37].

2) Performance Evaluation

Performance evaluation measures how efficiently cloud-native software platforms utilize computational resources while maintaining responsiveness, scalability, and reliability. Common evaluation metrics include response time, throughput, latency, resource utilization, scalability, workload distribution efficiency, and application availability under varying user loads. Experimental evaluation typically employs benchmarking tools and workload simulators to assess Kubernetes scheduling efficiency, auto-scaling performance, serverless deployments, and distributed microservices. Performance evaluation enables continuous optimization of intelligent resource management strategies while validating the effectiveness of reliability mechanisms in large-scale cloud-native software platforms [38].

Table I summarizes the major reliability and resilience techniques employed in cloud-native software platforms. It highlights the description, representative technologies and methods, and key advantages of each technique for improving application availability, fault tolerance, automated recovery, observability, and overall system reliability.

TABLE I. RELIABILITY AND RESILIENCE TECHNIQUES IN CLOUD-NATIVE SOFTWARE

Reliability Technique	Description	Representative Technologies/Methods	Key Advantages
Fault Tolerance	Enables applications to continue operating despite hardware, software, or network failures by detecting, isolating, and mitigating faults.	ReplicaSets, StatefulSets, Replication, Circuit Breaker, Retry Mechanisms, Health Probes	Continuous service availability, fault isolation, improved resilience
High Availability	Ensures continuous application availability through redundancy, failover, load balancing, and distributed deployments.	Kubernetes High Availability, Multi-Zone Deployment, Load Balancer, Failover Clustering, Service Mesh	Minimal downtime, improved uptime, SLA compliance
Self-Healing	Automatically detects failures and restores healthy application states without manual intervention.	Kubernetes Self-Healing, Pod Restart, ReplicaSet Controller, Node Recovery, Automated Remediation	Reduced MTTR, autonomous recovery, improved reliability
Failure Recovery	Restores services after failures using recovery, backup, replication, workload migration, and disaster recovery mechanisms.	Backup & Restore, Disaster Recovery (DR), Checkpointing, Workload Migration, Automated Failover	Rapid recovery, business continuity, reduced service disruption
Monitoring	Continuously collects infrastructure and application metrics to monitor system health and resource utilization.	Prometheus, Grafana, Kubernetes Metrics Server, Alertmanager	Real-time visibility, anomaly detection, proactive maintenance
Logging	Collects and stores application and infrastructure logs for debugging, auditing, and fault diagnosis.	Loki, Elasticsearch, Fluentd, Kibana (EFK/ELK Stack)	Root-cause analysis, security auditing, troubleshooting
Distributed Tracing	Tracks requests across multiple microservices to analyze dependencies, latency, and communication paths.	OpenTelemetry, Jaeger, Zipkin	End-to-end visibility, latency analysis, performance optimization
Reliability Metrics	Measures the dependability and availability of cloud-native platforms using quantitative reliability indicators.	Availability, MTBF, MTTR, Failure Rate, Error Rate, SLA/SLO Compliance	Reliability assessment, performance benchmarking, operational improvement
Performance Evaluation	Evaluates system efficiency, scalability, responsiveness, and resource utilization under varying workloads.	Response Time, Throughput, Latency, Resource Utilization, Scalability Testing, Benchmarking	Performance optimization, capacity planning, QoS improvement

E. Challenges and Future Research Directions

Despite significant advancements in cloud-native software platforms, challenges remain in intelligent resource management, reliability, security, scalability, and autonomous operations. The future generation of cloud-native systems require AI-powered, sustainable, and energy-efficient solutions to meet these challenges.

The major challenges in the cloud-native software platforms and future directions for research to design intelligent, scalable, secure, energy efficient, and highly reliable cloud-native systems using advanced resource management and autonomous operation techniques are summarized in Table II.

TABLE II. CHALLENGES AND FUTURE RESEARCH DIRECTIONS IN CLOUD-NATIVE SOFTWARE PLATFORMS

Open Challenge	Description	Future Research Direction
Scalability of Cloud-Native Applications	Managing large-scale microservices, containers, and Kubernetes clusters while maintaining performance, reliability, and efficient resource utilization.	Develop autonomous, distributed, and AI-driven resource management frameworks for large-scale cloud-native environments.

Intelligent Resource Management	Optimizing CPU, memory, storage, network, and accelerator resources under highly dynamic and heterogeneous workloads.	Design adaptive AI/ML-based resource allocation, workload scheduling, and predictive resource provisioning techniques.
Reliability and Fault Tolerance	Maintaining continuous application availability despite hardware failures, software defects, network disruptions, and infrastructure outages.	Develop predictive fault detection, self-healing architectures, and AI-assisted resilience mechanisms for cloud-native platforms.
Observability and Root Cause Analysis	Collecting and correlating metrics, logs, and traces across distributed microservices to rapidly identify system failures and performance bottlenecks.	Develop intelligent observability frameworks integrating AIOps, distributed tracing, and automated root cause analysis.
AI-Driven Resource Optimization	Ensuring AI-based resource management decisions remain accurate, explainable, secure, and adaptable under changing workload conditions.	Investigate trustworthy, explainable, and reinforcement learning-based optimization techniques for autonomous cloud-native operations.

V. Literature Review

The following section summarizes the related research of the cloud-native software platforms, emphasizing on the application of intelligent resource management and reliability improvement approaches. Finally, it outlines current challenges, research gaps and future directions for the development of intelligent, scalable and reliable cloud-native software platforms. The recent research summarized in Table 1 has enhanced understanding of AI-based resource management, container orchestration, cloud portability, and reliability improvements.

S. S. Gaddipati (2026) discusses architectural frameworks for self-healing, predictive auto-scaling, intelligent data governance and real-time anomaly detection of AI agents in distributed cloud environments. In addition, potential problems such as agent alignment, latency limits, security aspects, and interoperability with heterogeneous cloud environments are addressed. Experimental studies show that results of using agentic AI-based platforms are much better than using traditional rule-based automation in terms of throughput, resources utilization, and mean time to recovery [39].

Z. Ding et al. (2025) discusses a new architecture for radar information processing software based on cloud native technologies, including cloud-native technologies like "containerization", "microservices", and "elastic scaling". The study verifies its substantial enhancements in resource utilization, system flexibility, and reliability by comparing its functionality and performance with traditional radar information processing architectures. It offers new insights and methods for optimizing modern radar information processing systems [40].

R. Karanam (2024) presents a framework that integrates Terraform with containerization, orchestration, and IaC to simplify migrations to the cloud. research and practical implementation show that a serverless multi-cloud lakehouse can effectively manage varied analytics workloads, ensure compliance, and optimize performance. The results demonstrate that by implementing these techniques, cloud native lake houses become the go-to solution for data-driven businesses in terms of cost effectiveness, scalability, and governance [41].

P. Kumar and R. Gujjala (2023) presents a fresh approach that uses containerization and orchestration technologies to isolate cloud providers' dependencies, allowing businesses to become truly independent of their cloud providers and achieve true cloud portability. Serverless multi-cloud lake house architectures can provide better agility, scalability, and resilience than typical monolithic data platforms, according to a thorough review of performance indicators, cost optimization methodologies, and operational complexity reduction [42].

L. Wen et al. (2022) demonstrate that the trend prediction algorithm is utilized in real-time monitoring systems to anticipate the allocation and scheduling of microservice resources based on historical data, typically at a level that is 30.28 percent greater than prior to use, when coupled with the DL model in intelligent operation and maintenance technology. Communication operators' networks can benefit from cloud native technology's easier and more cost-effective deployment and operation methods [43].

H. Gadde (2022) explores the implementation of AI-enhanced adaptive resource allocation strategies tailored for cloud-native databases. By leveraging ML algorithms and data analytics, approach dynamically adjusts resource allocation based on realtime workload demands, ensuring high availability, scalability, and performance consistency. Experimental results demonstrate that adaptive resource allocation system reduces resource waste by up to 30% while improving response times by 25% compared to traditional static allocation methods [44].

TABLE III. Literature Review On Cloud-Native Resource Management And Reliability Techniques

Author(s) & Year	Focus Area	Methodology/Technology	Key Findings	Limitations
S. S. Gaddipati (2026)	Agentic AI for cloud-native platforms	Self-healing, predictive auto-scaling, intelligent data governance, anomaly detection	AI-driven platforms improved throughput, resource utilization, and reduced mean time to recovery compared with rule-based systems.	Challenges remain in agent alignment, latency, security, and interoperability across heterogeneous cloud environments.
Z. Ding et al. (2025)	Improve radar information processing using cloud-native architecture	Containerization, microservices, elastic scaling, cloud-native software-defined architecture	Enhanced resource utilization, system flexibility, scalability, and reliability compared with traditional radar architectures.	Focuses on radar information processing and lacks evaluation for general-purpose cloud-native applications.
R. Karanam (2024)	Cloud-native data platforms	Containerization, Kubernetes, Infrastructure-as-Code (Terraform),	Improved scalability, governance, compliance, and cost efficiency for analytics workloads.	Focuses mainly on data analytics platforms rather than intelligent resource management and reliability.

		serverless multi-cloud lakehouse		
P. Kumar & R. Gujjala (2023)	Multi-cloud portability	Containerization, orchestration, serverless architecture	Achieved cloud portability, vendor independence, scalability, agility, and resilience over traditional monolithic systems.	Does not incorporate AI-driven resource allocation or predictive reliability mechanisms.
L. Wen et al. (2022)	Intelligent operation and maintenance	Deep learning-based resource prediction and microservice scheduling	Resource scheduling efficiency improved by 30.28%, enabling economical deployment of cloud-native communication networks.	Evaluation is limited to communication operator networks with insufficient discussion of fault tolerance and reliability.
H. Gadde (2022)	Adaptive resource allocation	AI-enabled machine learning for dynamic resource allocation	Reduced resource waste by 30% and improved response time by 25% compared with static allocation methods.	Primarily evaluates database resource allocation without addressing distributed cloud reliability or multi-cloud environments.

VI. Conclusion

Cloud-native software platforms have transformed modern software development by enabling scalable, resilient, and agile applications through microservices, containerization, Kubernetes orchestration, serverless computing, and IaC. This survey reviewed the fundamental architecture of cloud-native platforms and examined intelligent resource management techniques, including resource allocation, monitoring, scheduling, auto-scaling, load balancing, AI/ML-based optimization, and performance enhancement. Furthermore, it discussed the concepts of reliability and resilience, including fault tolerance, high availability, self-healing, failure recovery, observability, and reliability testing, to make sure that services are always available in distributed systems. The comparative literature review revealed the increasing adoption of AI, ML, and AIOps for autonomous resource management and predictive system maintenance. However, issues of scalability, security, interoperability and complexity of operation exist. Cloud-native platforms of the future will be built on intelligent automation, explainable AI, and adaptive orchestration, ultimately offering highly reliable, efficient, secure and self-managing cloud computing environments.

References

[1] V. Ugwueze, "Cloud Native Application Development: Best Practices and Challenges," *Int. J. Res. Publ. Rev.*, vol. 5, pp. 2399–2412, 2024, doi: 10.55248/gengpi.5.1224.3533.

[2] S. R. Somula, "Mitigating Estimation Challenges in Large-Scale Projects: An Integrated Framework for Enhanced Accuracy and Adaptability," *IPHO-Journal Adv. Res. Sci. Eng.*, vol. 03, no. 10, pp. 1–9, 2025, doi: https://doi.org/10.5281/zenodo.17462211.

[3] V. K. Sharma and A. K. S, "Hierarchical Cloud-IoT Architecture for AI-Powered Intelligent Disaster Response," in *2025 7th International Conference on Innovative Data Communication Technologies and Application (ICIDCA)*, IEEE, Oct. 2025, pp. 603–610. doi: 10.1109/ICIDCA66325.2025.11280408.

[4] C. Shekhar Pareek, "Chaos Testing: A Proactive Framework for System Resilience in Distributed Architectures," *Int. J. Sci. Res.*, vol. 13, no. 11, pp. 851–855, Nov. 2024, doi: 10.21275/SR241110081650.

[5] J. B. Mehta, "AI-DRIVEN TEST ENGINEERING FOR CLOUD-NATIVE SYSTEMS," *Int. J. Data Sci. IoT Manag. Syst.*, vol. 5, no. 1, 2026.

[6] S. Banerjee, "Intelligent Cloud Systems: AI-Driven Enhancements in Scalability and Predictive Resource Management," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 4, no. 3, pp. 266–276, Dec. 2024, doi: 10.48175/IJARSCT-22840.

[7] S. Thalary and N. K. Kuntamukkala, "Operationalizing Software Invariants: A DevOps-Driven Approach to Reliability in Cloud-Native Systems," *Int. J. Emerg. Trends Comput. Sci. Inf. Technol.*, vol. 3, no. 4, pp. 157–168, 2022, doi: 10.63282/3050-9246.IJETCSIT-V3I4P115.

[8] M. R. C. Mukkolakkal, "Deploy, Calibrate, Monitor, Heal -- No Human Required: An Autonomous AI SRE Agent for Elasticsearch," *arvix.org*, Apr. 2026, [Online]. Available: http://arxiv.org/abs/2604.03933

[9] K. K. Mohammed, "Leadership Practices of Data Engineering for AI and Machine Learning," *Int. J. Sci. Res. Eng. Trends*, vol. 12, no. 1, pp. 1–5, 2026.

[10] S. Singh, "Advancing Wireless Communications with Open Radio Access Network," in *2025 6th International Conference on Data Intelligence and Cognitive Informatics (ICDICI)*, IEEE, Jul. 2025, pp. 682–687. doi: 10.1109/ICDICI66477.2025.11135206.

[11] A. Joon, "Smart Predictive Maintenance: AI-Driven Adaptation for Industrial Equipment," in *2025 IEEE North-East India International Energy Conversion Conference and Exhibition (NE-IECCE)*, IEEE, Jul. 2025, pp. 1–6. doi: 10.1109/NE-IECCE64154.2025.11183108.

[12] V. K. R. K. Koppireddy, "Passwordless Authentication: A Modern Approach to Secure Access," *J. Inf. Syst. Eng. Manag.*, vol. 10, no. 63s, pp. 1–6, 2025.

[13] K. Jangiti, "Design and Validation of a Machine Identity Governance Framework for AI Agents in Multi-Cloud Environments," in *SoutheastCon 2026*, IEEE, Feb. 2026, pp. 1–6. doi: 10.1109/SoutheastCon63549.2026.11476363.

[14] H. P. Cyril, N. D. Bhandarwar, S. Kumara, and S. Mathur, "Securing Containerised Applications Using Kubernetes-Based Orchestration in Software Development," in *2026 International Conference on Intelligent Computing, Networks, and Security (IC-ICNS)*, IEEE, Mar. 2026, pp. 1–6. doi: 10.1109/IC-ICNS68863.2026.11537887.

[15] B. Goyal, "Understanding cloud-native AI: The foundation of scalable platform architecture," *World J. Adv. Eng. Technol. Sci.*, vol. 15, no. 1, pp. 822–827, Apr. 2025, doi: 10.30574/wjaets.2025.15.1.0251.

[16] V. R. Gudelli, "Containerization Technologies : ECR and Docker for Microservices Architecture," *Int. J. Innov. Res. Manag. Pharm. Sci.*, vol. 11, no. 3, pp. 1–10, 2023.

[17] Y. Muni, "Advanced Multi-Protocol Label Switching (MPLS)-Enabled Networks: A Survey of Emerging Approaches Via Traffic Engineering," *Eng. Technol. J.*, vol. 11, no. 01, Jan. 2026, doi: 10.47191/etj/v11i01.22.

[18] M. K. Abhishek, D. R. Rao, and K. Subrahmanyam, "Framework to Deploy Containers using Kubernetes and CI / CD Pipeline," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 4, pp. 522–526, 2022.

[19] N. Adik, "The Rise of Open and Free Networks: A Community-Driven Paradigm for Decentralized Connectivity," in *2025 IEEE First International Conference on Innovations in Engineering and Next-Generation Technologies for Sustainability (ICINVENTS)*, IEEE, Nov. 2025, pp. 1–9. doi: 10.1109/ICINVENTS64613.2025.11402224.

[20] S. D. Veeravalli, S. Balla Venkata, A. Ashirova, R. M S, S. Appachu Devanira Poovaiah, and D. Turaev, "Breaking the Swarm Paradigm: Single Candidate Optimization for Cloud Task Allocation," in *2025 International Conference on Innovations in Intelligent Systems: Advancements in Computing, Communication, and Cybersecurity (ISAC3)*, IEEE, Jul. 2025, pp. 1–7. doi: 10.1109/ISAC364032.2025.11156559.

[21] S. K. Malaraju and S. K. Madishetty, "AI-Augmented Compiler Optimization for Energyefficient Software Execution on Embedded Systems," in *2025 14th International Conference on System Modeling & Advancement in Research Trends*

(SMART), IEEE, Nov. 2025, pp. 1–6. doi: 10.1109/SMART66937.2025.11389313.

[22] R. K. Kanneganti, "Security and Compliance Issues in Cloud-Based Deployments of Content and Workflow Management Systems," *Eastasouth J. Inf. Syst. Comput. Sci.*, vol. 2, no. 01, pp. 131–138, Aug. 2024, doi: 10.58812/esiscs.v2i01.1122.

[23] H. Liu, X. Wang, and C. Wang, *A Cloud-Native Heterogeneous Resource Scheduling Platform Supporting GPU Scheduling*, vol. 1, no. 1. Association for Computing Machinery, 2025. doi: 10.1145/3725899.3725923.

[24] R. Dandigam, R. T. Thutari, and T. Vaidya, "LLM-Augmented Agentic Consensus Swarms for Autonomous Edge Security," in *2026 14th International Symposium on Digital Forensics and Security (ISDFS)*, IEEE, Mar. 2026, pp. 1–8. doi: 10.1109/ISDFS69419.2026.11459110.

[25] R. Azmeera and J. Hyatt, " Software Errors, Product Functionality, and Organizational Cascades: Evidence from Developer-Lived Experience ," 2026. doi: 10.2139/ssrn.6924439.

[26] A. O. Ogundipe, "Adaptive Load Balancing and Auto Scaling Algorithms for Resource Optimization in Distributed Microservices based Cloud Applications," *Int. J. Sci. Archit. Technol. Environ.*, pp. 101–111, Aug. 2024, doi: 10.63680/ijsate0524111.06.

[27] M. Kalal, "Lean-Driven SAP Production Planning Optimization Using Machine Learning for Inventory and Throughput Efficiency," in *2026 14th International Symposium on Digital Forensics and Security (ISDFS)*, IEEE, Mar. 2026, pp. 1–6. doi: 10.1109/ISDFS69419.2026.11459029.

[28] A. S. Rao, "Orchestrating Efficiency : AI-Driven Cloud Resource Optimization for Enhanced Performance and Cost Reduction," *Int. J. Res. Publ. Rev.*, vol. 4, no. 12, pp. 2007–2009, 2023.

[29] R. Taware, S. K. Anumula, H. R. Chennam, and O. H. Kundurthy, "Exploring AI-Based Self-Configuring Slice Allocation Across Hybrid Access Networks," in *2025 Third International Conference on Networks, Multimedia and Information Technology (NMITCON)*, IEEE, Aug. 2025, pp. 1–6. doi: 10.1109/NMITCON65824.2025.11188257.

[30] R. Reddy Kethireddy, "Smart AI-Enabled Orchestration for Resource Optimization in the Cloud Environment," *Int. J. Sci. Res.*, vol. 13, no. 1, pp. 1822–1829, Jan. 2024, doi: 10.21275/SR24115214559.

[31] V. K. R. K. Koppireddy, "AI-Driven Cybersecurity Integration: A Comprehensive Framework for Enterprise Security Automation and Threat Management," *Int. J. Adv. Res. Eng. Technol*, vol. 16, no. 1, pp. 189–200, 2025.

[32] V. Chaturvedi, "Modern Software Development with Java , Spring Boot , and Python : A Survey of Frameworks and Best Practices," vol. 3, no. 4, pp. 188–197, 2023, doi: 10.56472/25832646/JETA-V3I8P121.

[33] C. Luca, "High Availability , Fault Tolerance , and Disaster Recovery Strategies," 2025, *Research Gate*.

[34] O. Johnphill, A. S. Sadiq, O. Kaiwartya, and M. A. Taheir, "Cross : a cloud _ native approach to automated remediation and self _ healing in cyber _ physical systems," *Cybersecurity*, vol. 9, no. 1, p. 124, 2026.

[35] O. D. Olufemi, A. O. Ejiade, O. Ogunjimi, F. Ogochuckwu, and Ikwuogu, "AI-enhanced predictive maintenance systems for critical infrastructure: Cloud-native architectures approach," *World J. Adv. Eng. Technol. Sci.*, vol. 13, no. 2, pp. 229–257, Dec. 2024, doi: 10.30574/wjaets.2024.13.2.0552.

[36] P. S. Yadav, "Real-Time Insights In Distributed Systems: Advanced Observability Techniques For Cloud-Native Enterprise Architectures," *Int. J. Core Eng. Manag.*, vol. 7, no. 07, pp. 24–34, 2023.

[37] J. Miller, D. Wilson, M. Johnson, and C. Andrew, "Performance and Reliability Assessment of Cloud-Native Intelligent Systems," *Artif. Intell. J.*, vol. 3, no. 3, pp. 1–7, 2022, doi: 10.5281/ZENODO.18234631.

[38] M. B. Irfansyah, B. Waheed, I. Winarno, and A. Alimudin, "Performance evaluation of serverless cloud-native API deployment: a case study on a mobile health application," *TELKOMNIKA (Telecommunication Comput. Electron. Control.*, vol. 24, no. 1, p. 34, Feb. 2026, doi: 10.12928/telkomnika.v24i1.27261.

[39] S. S. Gaddipati, "Agentic Ai-Driven Autonomous Data Platforms For Scalable Cloud-Native Architectures," *Int. J. Innov. Stud.*, vol. 10, no. 1, pp. 992–1006, 2026.

[40] Z. Ding, J. Wang, C. Hong, and K. Song, "Research of Cloud-Native-Based Software-Defined Radar Information Processing Architecture," in *2025 7th International Conference on Electronic Engineering and Informatics (EEI)*, IEEE, Nov. 2025, pp. 843–846. doi: 10.1109/EEI67650.2025.11334864.

[41] R. Karanam, "Modern Cloud-Native Lakehouse Architectures: Multi-Cloud Approaches For Advanced Analytics And Business Intelligence," *Int. J. Comput. Eng. Technol.*, vol. 15, no. 1, pp. 163–180, 2024, doi: doi.org/10.34218/IJCET_15_01_016.

[42] P. K. Reddy Gujjala, "The Future of Cloud-Native Lakehouses: Leveraging Serverless and Multi-Cloud Strategies for Data Flexibility," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 9, no. 5, pp. 868–882, Oct. 2023, doi: 10.32628/CSEIT239093.

[43] L. Wen, H. Qian, and W. Liu, "Research on Intelligent Cloud Native Architecture and Key Research on Intelligent Cloud Native Architecture and Key Technologies Based on DevOps Concept Technologies Based on DevOps Concept," *Procedia Comput. Sci.*, vol. 208, pp. 590–597, 2022, doi: 10.1016/j.procs.2022.10.082.

[44] H. Gadde, "AI-Enhanced Adaptive Resource Allocation in Cloud-Native Databases," *Rev. Intel. Artif. en Med.*, vol. 13, no. 1, pp. 443–470, 2022.