Cloud-Based College Event Management System Using Docker

Mohsin Kalim Ansari[1], Pranjal Harish Chopade[2], Soham Murlidhar Nhavkar[3], Sanika Popat Talekar[4]
[1,2,3.4]*Department of Computer Engineering, P. K. Technical Campus, Pune*

Abstract- Educational institutions face increased complexity in campus event coordination, from seminars to festivals. Traditional event management systems struggle to adapt. This paper presents a cloud-based system using Docker containerization. The system uses microservices to enable independent scaling and consistency. Docker containers package application logic, dependencies, and configurations, which allow quicker deployment across platforms. A responsive frontend, RESTful backend services, and a distributed database are integrated with container orchestration. System testing showed better resource use, faster deployment, and higher scalability than typical hosting. This research adds to containerized applications in educational technology and advises institutions on modernizing event coordination, as Docker transforms service and cuts costs.

Keywords — Cloud Computing, Docker Containerization, Event Management System, Microservices Architecture, Educational Technology, Scalable Systems, DevOps

I. INTRODUCTION

College campuses host numerous activities like conferences, recruitment, programs, and meetings. Coordinating these events needs technology that handles workloads, user needs, and participation. Many colleges use typical event management systems on servers. These systems cause operational problems and scale poorly, needing intervention during usage increases. Code changes from steps to production create inconsistencies.

Cloud computing and containerization have changed application deployment. Docker aids developers in packaging with its runtime environment. This solves compatibility issues. Cloud platforms like AWS and Google provide scalable resources. There is no maintaining hardware that sits idle.

Combining Docker with cloud infrastructure benefits college event management. Institutions can scale registration, reporting, and notification modules by dividing applications into microservices. This improves resilience by separating faults and speeds up development. Docker containers are more lightweight than VMs, which betters resource use and lowers costs.

This research covers the design and assessment of a cloud-based system for educational institutions. We used Docker containerization to fix the issues of deployment models. This paper talks about design choices, strategies, metrics, and observed benefits. The goal is to show that containerized platforms offer advantages and to guide institutions considering improvements.

II. LITERATURE REVIEW

The combination of cloud computing, containerization, and educational management attracts research. Academics study how technologies can modernize campuses. This section reviews existing literature, noting contributions and gaps that shaped our study.

Hadi and colleagues made a prototype for higher education in 2025. Their work showed that flexible scaling and cost are advantages of cloud deployment [1]. Freire studied Kubernetes microservices in architectures. His research showed improvements in balancing and scaling [2]. These studies show cloud platforms support technology, but they do not cover frameworks for event management.

Traditional systems in education have limits. Syarif and Pizaini mentioned key issues in 2022: architectures, codebases, and support for processes [3]. They proposed event-driven microservices for admission and services. Riyanto's team used Docker

containers to expand on this work. They reached over 40,000 HTTP requests in classroom application tests [4]. These results support containerized microservices for workloads, but mainly cover instructional systems.

Empirical studies compared containerization and virtualization to measure Docker's. Khaldi did an analysis of Docker-based university IT labs in 2025. He reported a 64.6% drop in RAM, 50.8% in CPU use, and 87.8% disk savings against virtual machines [7]. Elbelgehy and colleagues approved these results when studying Docker Swarm virtual labs. They saw CPU use around 13% and memory footprints of 103 MB [8]. Vaillancourt's team showed Docker's benefits by running workflows across computing environments [6]. These studies support Docker's ability, yet study laboratory and computational contexts instead of event coordination systems.

Moving from monolithic to microservices involves chances. Wang and Ma suggested decomposing, containing, and deploying to Kubernetes in 2019 [11]. Their structure improved stability while traffic management, scaling, and control. Researchers saw discussion of strategies, pipeline implementations, and patterns in publications [1]. Syarif and Pizaini stated that there were not enough large-scale evaluations of microservices [3]. Studies are geared toward laboratories and systems, leaving event platforms unexplored.

Security and reliability for containerized systems are getting attention. Dubec and colleagues used assignment systems in 2023. They emphasized sandboxing and queues to contain workloads while scaling [12]. Diouf's team fixed vulnerabilities by adding mechanisms into Kubernetes, showing continuity under failures [13]. These studies are focused on safety in systems. Security frameworks for event contexts are underdeveloped. The literature builds a base for educational systems while creating chances for research on event platforms that unite practices, and security.

III. PROPOSED SYSTEM ARCHITECTURE AND METHODOLOGY

Our cloud-based management system uses a microservices design run through Docker containers and hosted on cloud infrastructure. This enables scalable, maintainable coordination of college events. This section talks about the system's design parts, the stack, and the methodology that fixes limits found in typical approaches.

The architecture has three levels: presentation, application, and data. Each level is containerized separately to allow development, testing, and deployment. The presentation level uses a web interface built with JavaScript frameworks. This frontend is packaged inside a Docker container that serves assets and sends API requests to backend services. Contact with services is through RESTful APIs, ensuring coupling and independence. Users use dashboards for their roles: students, faculty, organizers, and administrators see functions for their credentials.

The application level runs logic through microservices, with each service for a domain. Our event service handles creation, changing, and deletion, managing details. A registration service handles enrollment, registration, waitlists, and capacity limits through database transactions. The notification service uses message queues to send email and alerts about updates, confirmations, and reminders. This ensures communication without blocking transaction flows. An authorization service using JWT security approves credentials and enforces access controls across system endpoints. Each microservice runs in a container, exposing APIs. Services connect through HTTP or message brokers like RabbitMQ or Kafka for interactions.

The data level uses database cases to store information and maintain system state. We employ PostgreSQL containers for data for user profiles and event records, ensuring transaction guarantees for processes. For caching data and session management, a Redis container gives in-memory storage, reducing database query loads. File storage for content uses cloud-native storage services like Amazon S3 or Azure Blob Storage, which work with applications through SDK libraries. Our database container uses volume mounts to ensure data across containers, mapping paths to storage that survives restarts.

Docker's containerization ability serves as the key to our strategy, giving advantages. Each container has an environment with application code, libraries, and

configuration files. This removes the problem common in typical deployments. Docker images are built through Dockerfiles to explain construction steps, ensuring builds across steps. The layered image architecture enables storage and transmission through caching, which speeds up cycles. Container orchestration through Docker Compose or Kubernetes automates service. This includes container health checks, restarts, and updates.

The cloud deployment model improves Docker's benefits through resource and services. Cloud platforms assign resources dynamically based on metrics: CPU use, memory, queue depths. The system scales container cases during high-traffic periods. Load balancers send requests across container replicas, ensuring instances become bottlenecks while giving tolerance when containers fail checks. Kubernetes services like Amazon EKS, Azure AKS, or Google GKE infrastructure complexity, handling operations, upgrades, and patching. This allows teams to focus on logic over overhead.

Our development and deployment flow follows DevOps principles, uniting integration and deployment pipelines that automate testing and processes. Source code triggers pipelines that compile code, run integration tests, make Docker images, and send images to registries. Deployment pipelines pull images and rolling updates to clusters, maintaining system availability through replacement strategies. Infrastructure as Code tools define cloud resources, enabling provisioning, supporting disaster recovery, and deployments. The outlined methodology ensures the management system achieves availability, and adaptation to through containerization and cloud practices.

IV. IMPLEMENTATION AND RESULTS

Converting our design into a system involved deploying on Amazon Web Services (AWS) cloud infrastructure with Docker containers run through Amazon Elastic Kubernetes Service (EKS). This section explains the implementation, decisions, procedures, and results that validate the approach.

We established a Docker-based environment that enabled iteration and testing before cloud deployment. Developers used Docker Compose to define applications through YAML files specifying service definitions, network setups, volume mappings, and variables. The frontend React application compiled into builds served by an Nginx container. We set it up with compression and browser caching to lower bandwidth consumption and better load times. Backend microservices run in Node.js with the Express framework ran within Docker images. We picked to cut transfer times and storage costs. PostgreSQL database containers initialized with migration scripts, ensuring structures across. Redis containers were set up with persistence policies that balanced with data needs.

Container orchestration through Kubernetes gave management abilities for reliable operation. Our EKS cluster had worker nodes to ensure availability and tolerance. Kubernetes Deployment resources defined specifications for each microservice: replica counts, resource requests, liveness and readiness checks, and update strategies. We set up autoscalers to track CPU and memory metrics, scaling replica counts between thresholds based on use. The registration service scaled from 3 to 10 replicas during enrollment when CPU use went over 70%. This distributed load across instances and maintained response times. Service resources with LoadBalancer provisioned AWS Elastic Load Balancers to distribute traffic across pod replicas, while Ingress controllers managed access and SSL termination for HTTPS connections.

Performance evaluations compared our containerized cloud deployment against a baseline virtual machine, We used Apache JMeter for testing, simulating user scenarios with user traffic. The containerized system showed resource. Under moderate load (500 users), average CPU use was 45% compared to 78% for the VM, Resource distribution showed similar advantages: the containerized deployment needed 6.2 GB total memory across services versus 12.8 GB for the VM —a 51.6% drop. Response time showed latency of 320 milliseconds for event operations in the containerized system versus 580 milliseconds for the VM.

Deployment velocity metrics showed Docker's on. The containerized system completed deployment cycles—from code commit to production—in about 8 minutes through automated pipelines. VM-based

deployments needed 45-60 minutes with server provisioning. Rolling updates executed without service, containers with new versions while maintaining replica counts. This contrasts with VM approaches that call for maintenance and service. Container times averaged 3-5 seconds from image pull to ready state, enabling responses to demand. VM boot called for 2-3 minutes before application.

Cost analysis showed economic advantages to the containerized cloud approach. The system's ability to scale during low-activity periods reduced compute costs by about 40% against VM infrastructure sized for capacity. Container allowed hosting services on compute instances. Node use reached 65-75% compared to 30-40% for VM deployments, maximizing infrastructure returns. Kubernetes services removed the need for personnel to handle cluster management, patching, and maintenance. While this introduced service fees, the cost of ownership remained.

Security included network policies restricting service communication to defined paths, through AWS Secrets Manager, and container image scanning to identify vulnerabilities before deployment. For monitoring and observability, we set up the system with Prometheus for metrics collection, Grafana for dashboards, and the ELK stack for logging. This gave visibility. These implementation collectively that Docker containerization with cloud delivers improvements in performance, agility, and for event management systems. It validates the and value of our.

V. CONCLUSION AND FUTURE SCOPE

This presented a cloud-based event system using Docker to fix limits in college event platforms. Our architecture that microservices with technology deliver improvements across dimensions: scalability, efficiency, resource, and agility. The showed quantifiable benefits—51.6% drop in memory consumption, 44.8% in response, and deployment speed from 45-60 minutes to about 8 minutes through automated pipelines. These outcomes Docker's for technology infrastructure and offer institutions a for modernized service.

The microservices design for educational contexts with workloads and tasks. Scaling registration, notification, and event services independently enabled resource with demand patterns. We avoided while maintaining performance during periods. Container removed environment in. This reduced and accelerated cycles. with cloud platforms gave infrastructure that adapted to, efficiencies through scaling, and eliminating from hardware.

Despite the, considerations. Our implementation focuses on requirements and. Security was through practices, but we threat and frameworks. The system's on for orchestration suggests for automation through resource prediction and scaling policies.

Future research possibilities for extending the system's and the. Kubernetes orchestration represents evolution from Docker Compose, offering features scheduling algorithms, mechanisms, and networking policies. Integrating service technologies enhance, security, and traffic through distributed tracing, authentication, and access controls. CI/CD pipelines that testing, scanning, and delivery deployments, safe deployment, and.

Artificial intelligence and machine offers potential for event. analytics could predict event based on patterns, sentiment, and calendars, enabling planning and resource. language processing could automate event, generate, and provide search that enhances user. systems could suggest events to students based on academic, past, and, visibility.

Incorporating monitoring and frameworks beyond metrics insights into system and. tracing through illuminate request flows across, identifying, and patterns. Application performance solutions could track user, performance with responsiveness to efforts. Chaos practices that introduce failures would system find weaknesses before they in.

strategies represent for enhancing system and vendor Designing applications of deployment would flexibility in provider based on cost, features, or. approaches that combine infrastructure for data with resources for elastic workloads could address data maintaining.

In, this Docker-based as a for educational event systems, delivering improvements in, and agility. Our practical for institutions on transformation, cloud-

native technologies offer. As and cloud are, educational institutions that these to leverage future. Their technology will remain, and with missions of campus. The forward continuous, automation, and of to the value to students, faculty, and.

REFERENCES

[1] N. K. Hadi, A. S. Mohammed, and H. T. Salim, Enhancing Software Reusability in Higher Education Applications through Microservices Architecture, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, vol. 16, no. 1, pp. 324-339, 2025.

[2] G. M. Freire, A Distributed Architecture of Reactive Microservices Orchestrated by Kubernetes: Case Study on Load Balancing in Local Cloud, Ph.D. dissertation, Dept. Computer Science, Univ. São Paulo, São Paulo, Brazil, 2025.

[3] U. Syarif and P. Pizaini, Penerapan Event-Driven Microservices pada Aplikasi Layanan Penerimaan Peserta Didik Baru, JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika), vol. 7, no. 3, pp. 891-902, Aug. 2022.

[4] D. J. Riyanto, P. Pizaini, and N. S. Herman, Implementasi Service Choreography Pattern Arsitektur Microservice Classroom Akademik Menggunakan Docker, JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika), vol. 7, no. 3, pp. 968-978, Aug. 2022.

[5] Campus Cloud: Empowering University Management Web Application with Cloud-Hosted Docker Technology on AWS, Indian Scientific Journal of Research in Engineering and Management, vol. 7, no. 5, pp. 1-8, May 2023.

[6] P. Z. Vaillancourt, J. M. Wozniak, S. Chard, B. Blaiszik, I. Foster, and K. Chard, Self-Scaling Clusters and Reproducible Containers to Enable Scientific Computing, in Proc. IEEE Int. Conf. Cloud Computing Technology and Science, 2020, pp. 119-126.

[7] A. Khaldi, Revolutionizing University IT Labs: A Docker-Based Approach for Next-Generation Learning Environments, in Proc. IEEE Advanced Computing and Data Science Applications, Aug. 2025, pp. 1-6.

[8] A. G. A. Elbelgehy, S. Abdelrazek, H. M. El Bakry, and A. A. Saleh, Performance Evaluation of Virtual Cloud Labs Using Hypervisor and Container, Indian Journal of Science and Technology, vol. 13, no. 48, pp. 4715-4728, Dec. 2020.

[9] A. Lakkadwala and P. Lakkadwala, Scalability and Stability in Cloud-Native Applications: Lessons from Docker and Kubernetes Deployments, in Advances in Cloud Computing and Distributed Systems, Springer, 2024, pp. 45-67.

[10] Y. Wang and D. Ma, Migrating Monolithic Applications to Microservices with Kubernetes: A Systematic Approach, in Proc. IEEE Int. Conf. Software Engineering and Service Science, 2019, pp. 234-239.

[11] G. M. Diouf, G. Noel, and N. Suri, On Byzantine Fault Tolerance in Multi-Master Kubernetes Clusters, Future Generation Computer Systems, vol. 110, pp. 1026-1037, 2020.

[12] J. Dubec, J. Balažia, R. Bencel, and M. Chovanec, Docker-Based Assignment Evaluations in E-Learning, in Proc. IEEE Int. Conf. Knowledge and Innovation Technologies, Oct. 2023, pp. 78-83.

[13] M. A. F. Marques, R. Santos, and P. Silva, Container Lifecycle Traceability Using Blockchain Technology, Journal of Cloud Computing: Advances, Systems and Applications, vol. 11, no. 2, pp. 45-58, 2022.