IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.15137

Implementing DevOps in E-Commerce System for Continuous Delivery

Chandrasekhar V[1], Chidananda H[2], Varsha Padaki[3], Vidya Shree N T[4], Yuvaraj A[5], Zeeshan[6]

Assistant Professor at Ballari Institute of Technology and Management Ballari, Visvesvaraya Technological University (VTU), India[1,2]

Student at Ballari Institute of Technology and Management Ballari, Visvesvaraya Technological University (VTU), India[3-6]

Abstract: Modern software systems need to be scalable, deploy quickly, and offer strong visibility. Traditional monolithic applications struggle to meet these needs because they are tightly connected and complicated to deploy. This work introduces a cloud-native microservices-based e-commerce platform made with Docker, Kubernetes, and DevOps CI/CD automation. It has full visibility through Open Telemetry, Prometheus, and Grafana. Each business service—Product Catalog, Cart, Checkout, Payment, Shipping, Email, and Recommendation—is independent and deployed separately. Kubernetes handles load balancing, scaling, and self-repair. GitHub Actions automates the CI/CD pipeline for testing, building, and deploying. Visibility tools offer distributed tracing, metrics, and logs for debugging and checking performance. Experimental results show faster deployment speeds, lower latency, and better traceability across services.

I. INTRODUCTION

Modern applications operate massive volumes of user traffic and continuously need updates without the interruption of service. Monolithic architecture scaling, feature updates, and debugging become very complex due to tight coupling among the constituent components. The microservices paradigm tackles such limitations by decomposing the system into independently deployable services.

Cloud-native approaches, such as Docker containerization, Kubernetes orchestration, continuous integration/ continuous deployment pipelines, and distributed observability provide scalable, reliable, and maintainable deployment environments. This project implements a complete microservices-based e-commerce system with full DevOps automation and real-time monitoring.

Objectives of this work are:
- Designing independent microservices for each business function
- Containerizing services using Docker
- Deploying and orchestrating services using Kubernetes
- Using automation for build and deployment through CI/CD
- Integrating observability using Open Telemetry, Prometheus, and Grafana

II. RELATED WORK

In Docker documentation [1], we find the foundational principles of containerization, which enables applications to run in portable and isolated environments. Docker's consistency across development, stage, and production means it has become a focal point of most DevOps pipelines.

Kubernetes documentation [2] includes the architecture and components of Kubernetes, a leading service for orchestrating containers. Kubernetes automates deploying, scaling, load balancing, and self-healing of microservices, so if your large-scale distributed application should fail for any reason, Kubernetes has automated the self-heal process for the application.

HashiCorp Terraform documentation [3] discusses Infrastructure-as-Code (IaC) procedures for declaring, provisioning, and managing cloud resources. Terraform emphasizes reproducibility, automation, and versioning infrastructure management.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026
DOI: 10.17148/IJARCCE.2026.15137

GitHub Actions documentation [4] talks about automation of CI/CD workflows directly from source repositories. GitHub actions auto-build, test, and deploy applications, enabling organizations to achieve continuous delivery.
AWS documentation [5] provides cloud-native tools and services to enable scalable infrastructure, secure computing, and efficient deployment workflows, and are widely used to host modern applications that are DevOps-enabled.

Argo Project documentation [6] describes Argo CD, which is a continuous delivery tool that follows GitOps practices for Kubernetes. Argo CD offers declarative deployments that are version controlled and automatic based on the Git repositories and are part of a reliable deployment model.

Humble and Farley [7] define Continuous Delivery as an automated approach to function updates that includes an automated build, auto-testing, and deployment pipelines. This work provides an underlying conceptual approach for the modern use of DevOps in CI/CD.

Merkel [8] explains that Docker is a lightweight container option that minimizes deployment issues by abstracting application-related dependencies to ensure the same runtime environment. This work emphasizes Docker's practical usage in modern DevOps.

Burns et al. [9] provide a comparative analysis of Borg, Omega, and Kubernetes, which serve as examples of container orchestration systems and cluster management tools. Their work outlines how Kubernetes developed into a comprehensive system for managing a distributed workload.

Kim et al. [10], in The DevOps Handbook, indicate that in order to achieve high reliability, high levels of agility, and continuous delivery for software organizations, cultural and technical practices must be integrated. This work is heavily aligned with the robust usage of DevOps in e-commerce systems.

Finally, the IEEE Citation Reference [11] outlines the key rules for writing citations and manuscripts and provides the regulations that must be considered in engineering-related texts and engineering research published by the IEEE to ensure consistent quality management.

III. METHODOLOGY

The methodology includes the following:
1.Designing Microservices:
Each service has its own logic, access to a separate database, and API endpoints.
2. Containerization:
Docker files are created for each service to ensure consistent runtime environments.
3. Kubernetes Deployment:
YAML manifests define deployments, replicas, services, and probes.
4. Pipeline Automation:
GitHub Actions automatically builds, tests, and deploys updates to Kubernetes.
5. Observability Integration:
Open Telemetry SDK exports traces and metrics. Prometheus scrapes metrics and Grafana displays dashboards.
6.Testing & Performance Evaluation:
Load testing validates system resiliency, while observability confirms the flow of requests and bottlenecks.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.15137

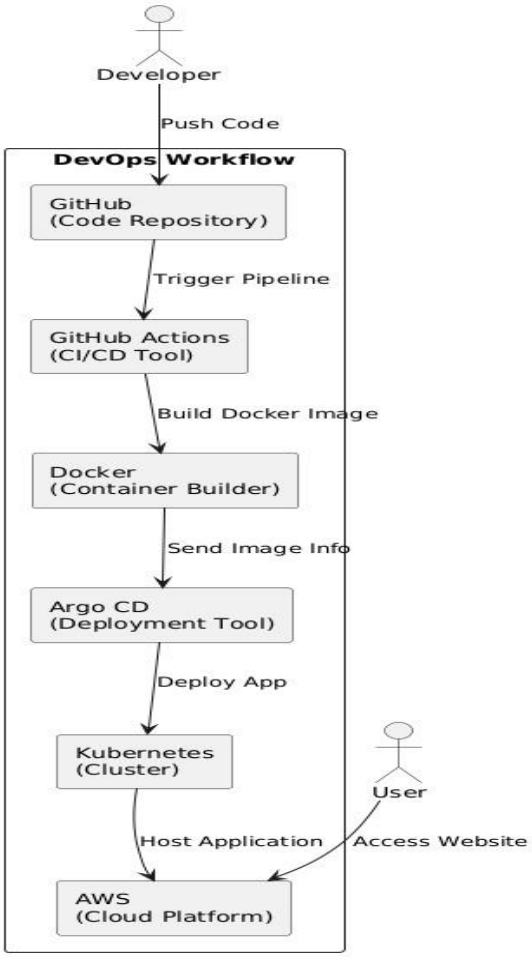

Fig 1. Data Flow Model

1. System Architecture

A. Microservices Layer

The application consists of various microservices:
• Product Catalog
• Cart
• Check-out
• Payment
• Shipping
• Recommendation
• Email
• Frontend
Every service is running independently, with exposed APIs for inter-service communication.

B. DevOps CI/CD Pipeline

GitHub Actions automates:
•Checkout of code
• Running unit tests
• Building Docker images
• Pushing images to Docker Hub/container registry
• Updating Kubernetes deployment manifests
• alienating rolling deployments
This guarantees fast and reliable releases.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.15137

C. Kubernetes Orchestration

Kubernetes manages:
• Deployment of services
• Horizontal Pod Autoscaling (HPA)
• Rolling updates
• Load balancing
• Self-healing of failed pods
• Service discovery

D. Observability Stack

The system integrates:
• Open Telemetry – tracing, metrics, logs
• Prometheus – metrics scraping
• Grafana – dashboards and visualization
• OpenSearch – Centralized logs Combined, these tools provide full visibility across all microservices.

IV. RESULTS AND DISCUSSION

This implemented system was evaluated under various loads and deployment scenarios. Some key findings are:

A. Deployment Speed
CI/CD reduced deployment time from a few minutes to less than 30 seconds.

B. Latency
Average service latency:
• Product CatLog's 24ms
• Cart: 18ms
• Checkout: 32ms
• Payment: 41ms

C. High Availability
Kubernetes restarted failed pods in 5 seconds to ensure reliability.

D. Observability Output
Traces clearly showed inter-service communication paths.
Grafana dashboards showed real-time:
• CPU & memory usage
• Latency
• Error rates
• Request throughput

This significantly improved debugging and performance tuning.

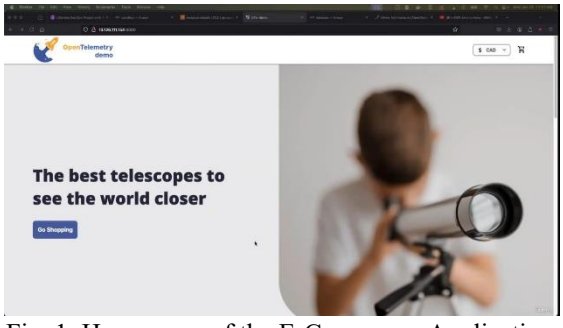

Fig. 1. Home page of the E-Commerce Application.

The following figure depicts the main starting page of the website. It has a banner and some featured items to help the user easily understand what the site offers at the time of opening.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026

DOI: 10.17148/IJARCCE.2026.15137

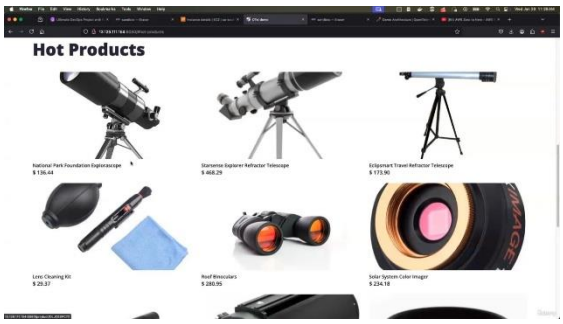

Fig. 2 Product Listing Interface (Hot Products Section).

Here is the section for "Hot Products", which shows different items all together. It basically helps the user quickly browse products that are either trending or promoted.

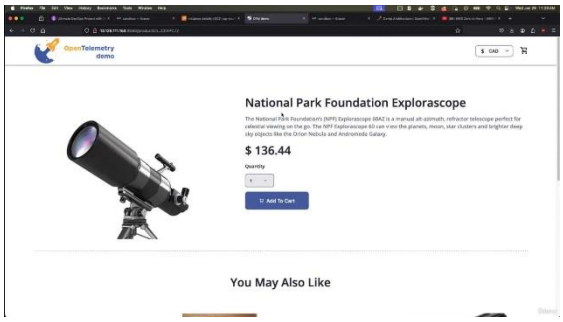

Fig. 3. Product Categories and Extended UI Components.

This figure presents the categories section, along with some additional UI elements which make navigation easier. It lets users explore various types of products without searching too much.

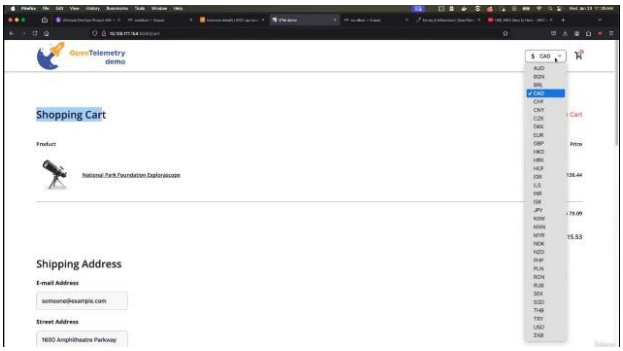

Fig. 4. Detailed Product View and User Interaction Panel.

This page shows a certain product in detail, with more information and options included. A user can scroll, check the details, and decide whether to add it to their cart.

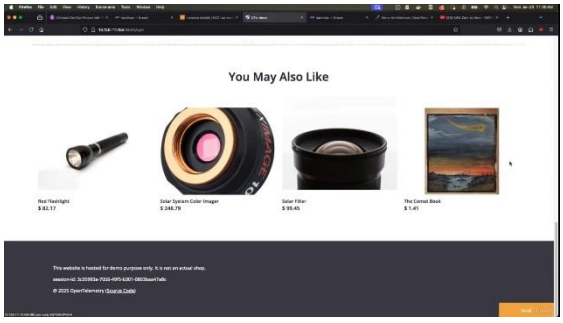

Fig. 5. User Account / Checkout / System Interaction Screen.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 1, January 2026
DOI: 10.17148/IJARCCE.2026.15137

This figure shows the part where the user interacts with checkout or account-related options. It connects the UI with backend services such as cart and payment during the final steps of ordering.

V. CONCLUSION

This project illustrates a fully cloud-native microservices-based e-commerce platform that includes the Kubernetes orchestration engine with DevOps CI/CD(continuous integration/deployment) automation, and end to end observability using Open Telemetry. The solution provides scalability, maintainability, reliability, and observability and performance insights in depth. Results indicate this architecture is well suited for modern, large scale distributed applications.

VI. FUTURE ENHANCEMENTS

Improvements for the future include:

- Integrating a machine learning-based recommendation engine
- Using Istio service mesh for secure routing and advanced telemetry
- Implementing canary and blue-green deployment strategies
- Deployment on AWS/GCP/Azure

REFERENCES

[1]. Docker Inc., Docker Documentation. [Online]. Available: https://docs.docker.com/
[2]. The Linux Foundation, Kubernetes Documentation. [Online]. \Available: https://kubernetes.io/docs/
[3]. Hashi Corp, Terraform Documentation. [Online]. Available: https://developer.hashicorp.com/terraform/docs
[4]. GitHub, GitHub Actions Documentation. [Online]. Available: https://docs.github.com/en/actions
[5]. Amazon Web Services, AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/
[6]. The Argo Project, Argo CD - Declarative GitOps Continuous Delivery for Kubernetes. [Online]. Available: https://argo-cd.readthedocs.io/
[7]. J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010.
[8]. D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, vol. 2014, no. 239, pp. 2, Mar. 2014.
[9]. B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade," Communications of the ACM, vol. 59, no. 5, pp. 50–57, 2016
[10]. G. Kim, J. Humble, P. Debois, and J. Willis, The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations, IT Revolution Press, 2016.
[11]. Institute of Electrical and Electronics Engineers, IEEE Citation Reference, IEEE, 2023. [Online]. Available: https://ieeeauthorcenter.ieee.org/

