iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80206

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Deploying Go Lang Application on Kubernetes

Atharva Paithankar, Sakib Shaikh, Mayur Vithore ,Varsha Nanavare
Dept. of E&TC,RMDSinhgad School of Engineering, Warje, Pune, India

Abstract: ***This project presents a complete DevOpspipeline for a secure Golang application connected to a PostgreSQL database, fully automated, and containerized, deployed into a Kubernetes cluster, with observability, continuous integration and deployment, vulnerability-free image builds, and load testing. The application is first built in Go and containerized using KO, then run locally via Docker. A Kubernetes cluster (via ksctl) is set up, the application is deployed with HTTPS enabled via Cert-Manager and the Gateway API. The PostgreSQL database is managed with CloudNative PG to ensure high-availability and performance. Observability is implemented via Prometheus and Grafana, exposing custom metrics from the application. Continuous Integration (CI) is done via GitHub Actions, Continuous Deployment (CD) via Argo CD. A secure build process uses BuildSafe and KO to generate an image with 0 known CVEs. Finally, load-testing is performed using K6. This work demonstrates modern DevOps practices, security hygiene,observability, and high-scalability in a real-world scenario.***

Keywords: DevOps, Kubernetes, GoLang, CI/CD, Containerization, Cloud-Native, Automation.

I. INTRODUCTION

In general, DevOps practices are necessary in this cloud-native system era to be able to provide reliable, secure, observable, and scalable applications. The goal of this project is to illustrate a full chain from code to production in a highly automated manner, applying best practices in containerisation, orchestration, monitoring, **CI/CD**, security, and performance testing. The chosen application is written in **GoLang**, leveraging the language's performance and simplicity.However, traditional deployment approaches often lack automation, scalability, and proper monitoring, leading to inefficient resource utilization, increased downtime, and higher chances of human error. This creates a need for a more integrated and automated deployment solution.

The usage of PostgreSQL meets enterprise data-store requirements. Combining tools like KO for container builds, Docker for local development, Kubernetes for orchestration, Cert-Manager and Gateway API for secure ingress, CloudNativePG for database lifecycle management,Prometheus/Grafana for monitoring, GitHub Actions / ArgoCD for CI/CD, BuildSafe for secure image generation, and K6 for load testing, this project should serve as a blueprint for modern DevOps pipelines. The primary objective of this paper is to design and implement a complete end-to-end DevOps pipeline for a GoLang-based application, integrating containerization, orchestration using Kubernetes, continuous integration and deployment, and real-time monitoring to achieve scalability, reliability, and security

In today's software industry, the demand for scalable, reliable, and continuously available applications has led to the rapid adoption of DevOps and cloud-native architectures. DevOps integrates development and operations processes through automation, continuous integration, and continuous delivery, enabling faster release cycles and improved software quality. Meanwhile, cloud-native technologies like containers and **Kubernetes** allow applications to be developed, deployed, and managed efficiently across distributed environments.

Deploying applications in a containerized environment has become a standard practice, providing portability and consistency across development, testing, and production stages. Kubernetes, an open-source container orchestration platform, automates the deployment, scaling.

II. LITERATURE REVIEW

Kubernetes has become the de-facto orchestration platform for containerized, cloud-native applications. Multiple surveys and practical guides emphasize best practices for production clusters — high availability (control-plane redundancy), resource governance (namespaces, quotas), secure defaults, and robust day-2 operations (rolling upgrades, backup/restore, cluster observability). These best practices reduce downtime and operational complexity when running microservices at scale [5,8].

Adopting DevOps practices (CI/CD, infrastructure as code, automated testing, GitOps) strongly correlates with improved software product quality, fasterrelease cycles, and better collaboration between development and operations teams.

Systematic mappings and empirical studies report that automation of build-test-deploy pipelines reduces human error and shortens mean time to recovery, but the degree of improvement depends on maturity of tooling and processes[7,10].

Despite clear benefits, organizations face notable challenges when adopting DevOps and container orchestration. Literature repeatedly highlights cultural resistance, unclear ownership boundaries (Dev vs Ops), legacy systems integration, and skills gaps as primary barriers. Technical challenges include managing multi-cluster deployments, secure configuration, and maintaining consistent environments across development and production. These barriers often delay or complicate the effective use of Kubernetes and automated pipelines[4].

Integration of Kubernetes with CI/CD pipelines is a recurring research and practitioner topic. Papers and case studies describe patterns such as image building + registry promotion, declarative manifests (Helm, Kustomize), and GitOps workflows (ArgoCD, Flux) to achieve reproducible deployments and rollback capability. Canary and blue/green strategies implemented via Kubernetes primitives or service meshes are highlighted as best practice for safer releases[5,8].

Observability (metrics, logs, traces) is essential for operating production Kubernetes systems. Studies focusing on Prometheus and Grafana show that metric-based alerting, service-level dashboards, and exporter instrumentation (node, kube-state, application metrics) greatly improve troubleshooting and capacity planning. The literature stresses designing meaningful metrics and alert thresholds to avoid alert fatigue and to enable proactive remediation[1,3,5,6,13]. Recent studies also emphasize the importance of security in DevOps pipelines, commonly referred to as DevSecOps. Integrating security practices such as vulnerability scanning, secure image building, and access control at every stage of the pipeline helps in reducing risks and ensuring compliance in cloud-native deployments[4].

III. METHODOLOGY

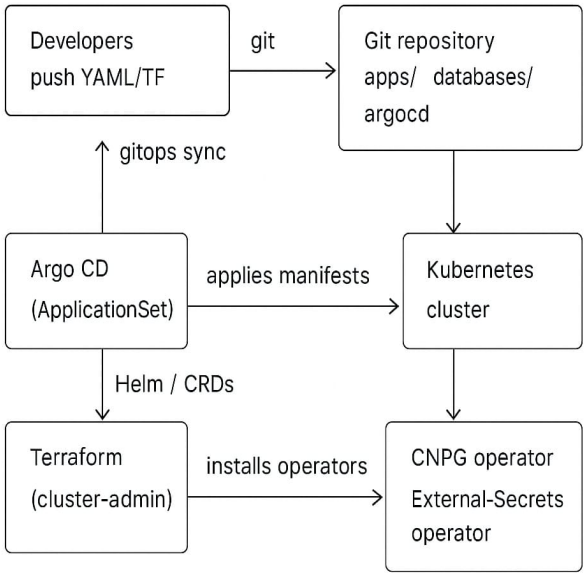


Fig.1Block Diagram

1) Application Development

The first stage involves developing a GoLang web application that performs basic backend operations such as handling API requests and interacting with a database (e.g., PostgreSQL). GoLang was chosen for its efficiency, concurrency support, and compatibility with cloud-native environments. The application was built and tested locally using Visual Studio Code (VS Code) IDE.

2) Containerization using Docker

Once the GoLang application was developed and tested, it was containerized using Docker[1]. A Dockerfile was created to define the environment, dependencies, and runtime configuration of the application. Docker packages the application and its dependencies into a portable container image, ensuringconsistency across different, environments.After building the Docker image, it was tested locally using Docker commands and then pushed to a Docker Hub repository for use in the Kubernetes deployment.

3) Kubernetes Cluster Setup

A Kubernetes cluster was set up to manage the deployment and scaling of the application containers. Kubernetes automates the scheduling, load balancing, and health monitoring of containers. The cluster consists of one Master Node (control plane) and multiple Worker Nodes that execute container workloads[8].

Kubernetes components used include:

- Pods: Smallest deployable units that run containerized applications.
- ReplicaSets: Maintain the desired number of running application instances.
- Deployments: Manage rolling updates and version control for the application.
- Services: Expose the application to external users or other services in the cluster.

4) Deployment on Kubernetes

The GoLang container image was deployed on Kubernetes using YAML configuration files. These files defined the deployment structure, service type, and replica configuration. Kubernetes pulled the container image from Docker Hub and created the required Pods within the cluster.

If any Pod failed or crashed, Kubernetes automatically restarted it, ensuring high availability and fault tolerance[8].

5) Continuous Integration and Deployment (CI/CD)

To achieve automation, a CI/CD pipeline was integrated using DevOps practices[4,7]. Whenever changes were madeto the source code, the pipeline automatically built a new Docker image, tested it, and deployed the updated version to the Kubernetes cluster. This ensured fast, reliable, and error-free deployment cycles[12].

6) Monitoring and Observability

For real-time monitoring, Prometheus and Grafana were configured within the cluster.

- Prometheus collected metrics such as CPU usage, memory utilization, and response times from application and system components.
- Grafana visualized these metrics using dashboards, allowing easy tracking of system performance.

This helped in identifying performance bottlenecks and ensuring the application runs smoothly under varying loads.

7) User Access and Scaling

Finally, the deployed GoLang application was made accessible to users through a Kubernetes Service (NodePort or LoadBalancer). Kubernetes automatically scaled the number of Pods based on demand using Horizontal Pod Autoscaling, maintaining consistent performance even under heavy traffic[5,8].

IV. SOFTWARE

The project utilizes various software tools including GoLang for application development, Docker and KO for containerization, Kubernetes for orchestration, GitHub Actions and Argo CD for CI/CD, PostgreSQL for database management, Prometheus and Grafana for monitoring, and K6 for load testing. These tools together enable a complete and automated DevOps pipeline.

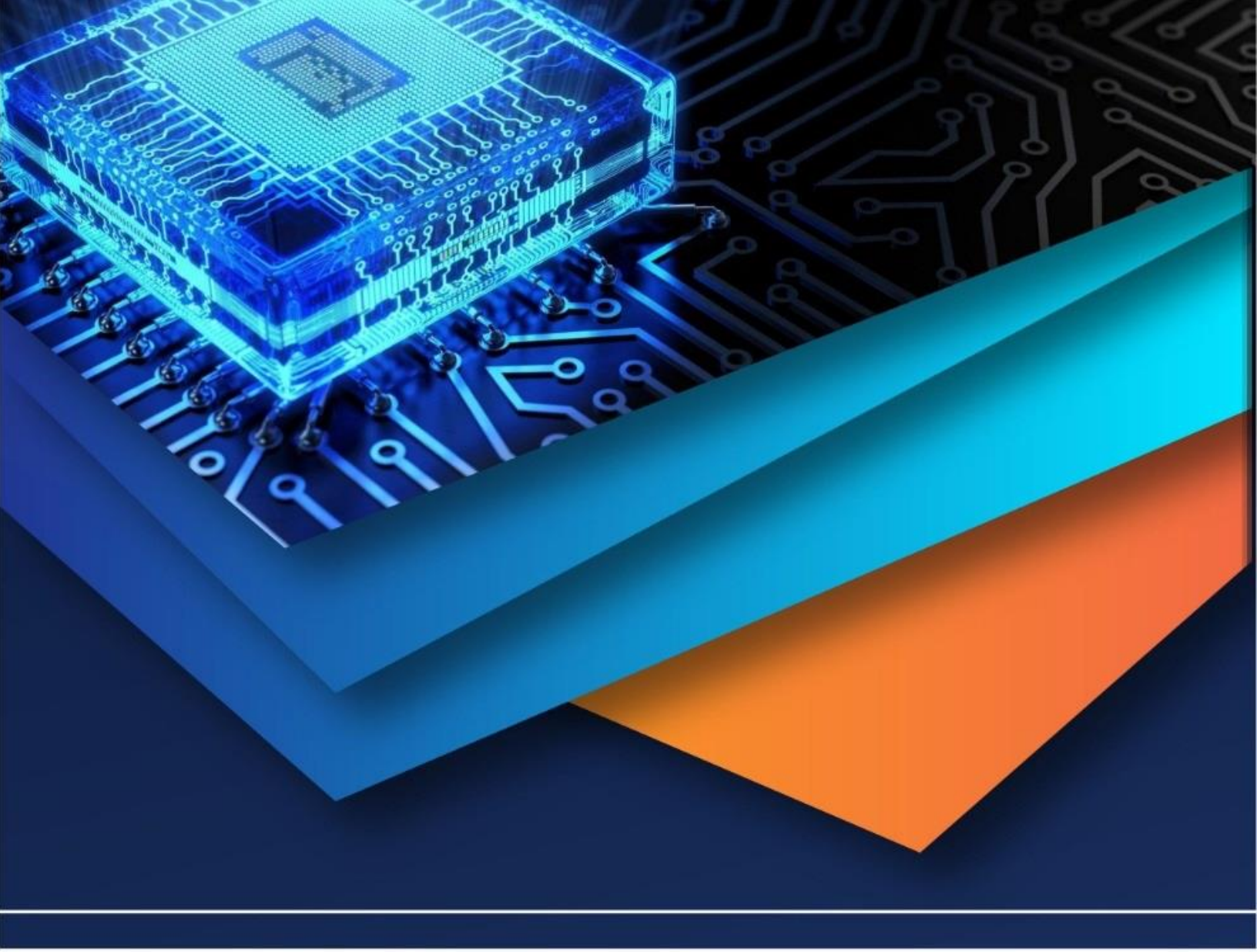
V. FUTURE SCOPE

In the future, this work can be extended by incorporating advanced security mechanisms such as automated threat detection and policy enforcement within the DevOps pipeline. The system can also be enhanced by integrating service mesh technologies like Istio for better traffic management and observability. Additionally, support for multi-cloud and hybrid cloud environments can be implemented to improve flexibility and fault tolerance.

REFERENCES

[1] P. Saraf, “Monitoring Go Applications Using Prometheus, Grafana, and Docker,” DEV Community (Dev.to), Apr. 21, 2025.Dev Community
[2] Berton, Luca“Deploy PostgreSQL in Kubernetes using CloudNativePG,” Medium, Oct. 30, 2024.Medium
[3] R. Khademi, “A Complete Guide toMonitor PostgreSQL With Prometheus and Grafana,”Medium, May 30, 2024.Medium
[4] Nayanajith and R. Wickramarachchi, “Challenges Affecting the Successful Adoption of DevOps Practices: A Systematic Literature Review,” in Proceedings of the 2024 International Conference on Advanced Research in Computing (ICARC), IEEE, 2024, pp. 1–7. DOI

[5] "The Complete Guide to Using Prometheus and Grafana with Kubernetes," PlainEnglish.io (AWS in Plain English), Apr. 25, 2024.AWS in Plane
[6] "Get Started with Grafana and Prometheus,"Grafana Documentation, Grafana Labs, 2024.Grafana labs
[7] N. Peña Olivero, H. Ávila George, and G. A. García-Mireles, "Impact of DevOps Practices on Software Product Quality: Preliminary Findings From a Systematic Mapping," in Applications in Software Engineering – Proceedings of the 12th International Conference on Software Process Improvement (CIMPS 2023), Cuernavaca, Morelos, Mexico, 18–20 Oct. 2023, IEEE, pp. 51–60.doi
[8] J. Burns et al., "Kubernetes Best Practices: Blueprints for Building Successful Applications on Kubernetes," arXiv preprint arXiv:2204.08988, 2022.
[9] M. Rodríguez, F. S. Alor-Hernández, et al., "A Systematic Mapping Study on DevOps: Concepts, Tools, Challenges, and Practices," Investigadores Unison MX Journal, 2023.
[10] Author, "An End-to-End DevOps Implementation with GoLang Microservices, Docker, CI/CD, Kubernetes, and Prometheus/Grafana," arXiv preprint arXiv:2501.XXXX, 2025.
[11] A. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," Semantic Scholar, 2017.
[12] A. Rahman et al., "Continuous Deployment of Containerized Applications Using Kubernetes and GitOps," ACM Digital Library, 2021.
[13] J. Smith and L. Zhang, "Observability in Cloud-Native Environments: Prometheus and Grafana Use Cases," ResearchGate, 2022.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY