ISSN:2790-1688 Volume-14-(2025)

A Study on Continuous Integration and Automated Deployment of Software Based on DevOps

Yuyan Gai [1], Yingjie Wu [1], Qingxi Liu [1, *], Yixin Pan [2]

[1] Software College, Harbin Institute of Information Engineering, Harbin, 150431, Heilongjiang, china

[2] School of Art and Design, Harbin Institute of Information Engineering, Harbin, 150431, Heilongjiang, china

Abstract. This paper systematically discusses the core technology of continuous integration, the implementation of automated deployment and its key technical practices in DevOps environment, including code management and version control, automated testing, containerization and orchestration technology, monitoring and performance optimization. Research shows that by building an efficient CI/CD pipeline, software delivery efficiency can be significantly improved, deployment risk can be reduced, and system scalability can be enhanced. Combined with the current technology trends, this paper looks forward to the future development direction of DevOps, in order to provide theoretical support and practical guidance for the continuous optimization of software engineering.

Keywords: continuous integration; automated deployment; DevOps; containerization; code management.

1. Introduction

In the field of modern software engineering, Continuous Integration (CI) and Automated Deployment (Automated Deployment) have become key technologies to improve the efficiency and quality of software delivery. The traditional software development model often faces the problems of code integration conflicts, long deployment cycle, complex environment configuration, etc., which leads to low delivery efficiency. the introduction of DevOps breaks the barriers between development and operation and maintenance, realizes the automation of the whole process from code submission to production deployment, and effectively improves the maintainability and scalability of the system. This study focuses on continuous integration and automated deployment of software in DevOps environment, and discusses its theoretical foundation, key technologies and practical path in depth.

2. DevOps Continuous Integration Theory and Technical Architecture

2.1 DevOps Concepts and Development

DevOps (Development and Operations) is a fusion of software development (Development) and operations (Operations) concepts and practices, designed to break down the traditional barriers between development and operations, through automation and collaboration to improve the efficiency and quality of software delivery. DevOps emphasizes Continuous Integration (CI), Continuous Delivery (CD) and Automated Operations (Automated Operations) to build an agile and efficient software engineering system.

The rise of DevOps stems from the popularity of Agile development methodologies and the need for rapid iteration and high-quality delivery in the enterprise. The traditional software development model often has the problem of development, testing, and deployment running in isolation, resulting in long code integration cycles, lagging response from operations and maintenance, and slow software updates. In order to solve these pain points, DevOps introduces automation tool chain, realizing the whole process automation from code submission to production environment deployment, and significantly improving the delivery efficiency.Google, Amazon, Netflix and other technology leading enterprises took the lead in adopting DevOps mode, and through the infrastructure as Code (Infrastructure as Code, IaC), continuous monitoring and feedback and other technologies. Google, Amazon, Netflix and other technology leaders took the lead in adopting the DevOps model, and through infrastructure as code (Infrastructure as Code (IaC)), continuous monitoring and feedback and other technical means to achieve efficient operation and maintenance, to promote industry change. From the perspective of technology development, DevOps has gone through multiple stages of evolution. Initially, it only focused on version control and continuous integration, and then introduced automated testing, containerized deployment, microservice architecture and other technologies to improve the maintainability and scalability of software. In recent years, with the deep integration of cloud computing and artificial intelligence, DevOps is developing in the direction of intelligence and adaptability, and new technologies such as machine learning-based anomaly detection and intelligent operation and maintenance further enhance the automation capabilities of DevOps. In the future, DevOps will continue to evolve and become the core paradigm driving the efficient development of software engineering.

2.2 Key technologies for continuous integration

Continuous Integration (CI) is the core part of DevOps system, aiming to ensure software quality and development efficiency through high-frequency code merging and automated testing. Its key technologies cover source code management, build automation, test automation, dependency management and environment isolation to ensure the stability and consistency of code changes. Source Code Management (SCM) is the foundation of Continuous Integration, which usually adopts distributed version control system (e.g. Git) for code hosting and combines with branching strategies (e.g. Git Flow, Trunk-Based Development) to improve collaboration efficiency. At any point in time, the state of the codebase can be represented as:

$$S_t = f\left(S_{t-1}, \Delta C_t\right)$$

where S_t represents the state of the codebase at time t, ΔC_t is the set of code changes in the current commit, and f(·) is a code merge function that ensures that the merged codebase remains buildable. Build Automation implements source code compilation, dependency resolution, and artifact packaging through build tools (e.g., Maven, Gradle) to ensure that the software can correctly generate executable units. The build process can be abstracted as:

$$B = G\left(S, D\right)$$

where B represents the build product, S is the source code, D is the dependencies, and G(·) represents the build process. Test Automation (TAM) is the quality assurance aspect of continuous integration, covering unit testing, integration testing and regression testing. Unit testing is usually

automated using frameworks such as JUnit, TestNG, etc. for validation, and its core goal is to ensure:

$$\forall c_i \in C, \quad T(c_i) = pass$$

Where C is the code changeset and $T(c_i)$ represents the test result of changing c_i , requiring all test cases to pass. Dependency Management and Environment Isolation techniques, such as Docker containers, virtual environments, and artifact repositories (Nexus, Artifactory), are used to resolve environment consistency issues and prevent dependency conflicts or runtime incompatibilities.

2.3 Continuous Integration Technology Architecture Model

In the continuous integration process, after the developer submits the code, the version control system (e.g., Git) triggers the CI pipeline, and the build server (e.g., Jenkins, GitLab CI/CD) automatically performs the build tasks, including code pulling, dependency management, compilation and packaging, and other operations. Subsequently, the system enters the automated testing phase, running unit tests, integration tests and code quality analysis to ensure software stability. Tested builds are stored in artifact repositories (e.g., Nexus, JFrog Artifactory) for subsequent deployment. Environment management leverages containerization (Docker, Kubernetes) and infrastructure-as-code (Terraform, Ansible) technologies to ensure consistency across different phases of the environment, avoiding the problem of "running locally but failing in production". In addition, monitoring systems (e.g. Prometheus, ELK) track build status in real-time and provide visual feedback to quickly pinpoint problems.

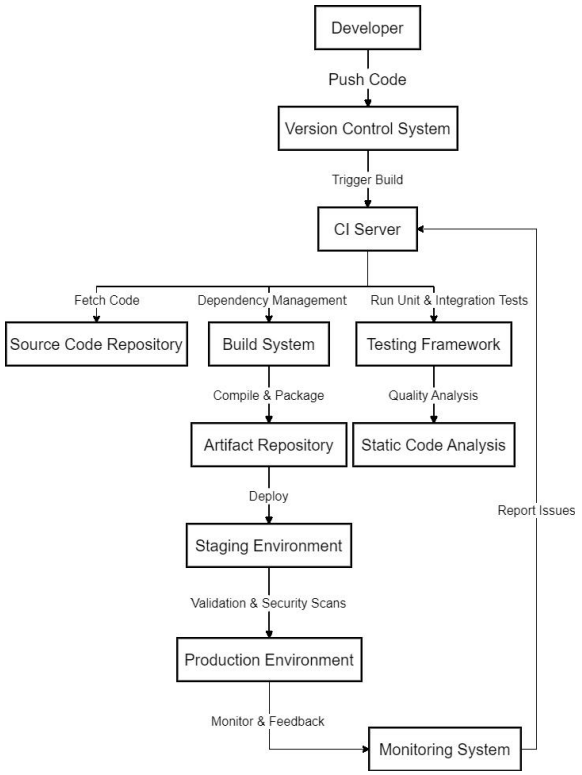

Figure 1. Continuous integration technology architecture model diagram

3. Automated Software Deployment Methodology and Realization

3.1 Automated deployment process design

Automated deployment is the core part of Continuous Delivery (CD) in DevOps system, which aims to achieve efficient and reliable software release through standardized processes and automation tools. The process mainly includes key steps such as code packaging, artifact management, environment preparation, deployment execution and rollback mechanism to enhance system maintainability and scalability. Automated deployment starts with build products (e.g., Docker images, JAR/WAR packages), which are generated by the continuous integration pipeline and stored in the artifact repository. Subsequently, an environment management system (e.g. Ansible, Terraform) automatically configures the target environment to ensure that runtime dependencies are consistent with the infrastructure. The deployment execution phase typically uses container orchestration (e.g., Kubernetes) or automated scripts (e.g., Shell, Python) to complete application distribution, and reduces downtime through Blue-Green Deployment, Rolling Update, and other strategies. The automated deployment system should have a built-in monitoring and rollback mechanism to detect the health status of the application in real time and quickly roll back to a stable version in case of anomalies to ensure business continuity. Efficient automated deployment process can significantly improve software delivery efficiency, reduce the cost of manual intervention, and enhance system stability and resilience.

3.2 Containerization and orchestration technologies

Containerization is one of the core technologies for automated deployment, providing a lightweight, isolated runtime environment that improves software portability and consistency. Containers have faster startup and lower resource consumption than traditional virtual machines, and rely on OS-level virtualization technologies such as Linux Namespaces and Cgroups to achieve process-level resource isolation. Assuming that a container instance C_i runs on host H, its resource allocation can be expressed as follows:

$$R(C_i) \subseteq R(H)$$

where $R(C_i)$ represents the set of resources available to the container C_i and R(H) is the total host resource set, ensuring that containers do not interfere with each other. In practice, Docker is the most widely used container engine with an image mechanism that allows applications and their dependent environments to be packaged into standardized units for cross-platform deployment. Container image consists of a multi-tier file system, which is efficiently reused through tiered storage (UnionFS), and the image construction can be represented as:

$$I = \sum_{n=1}^{N} L_n$$

Where I is the final image and L_n is the nth tier of the image, ensuring that different applications share the base tier to optimize storage space. Container orchestration is used to manage large-scale container deployments to ensure high availability and elastic scaling of applications.Kubernetes is

currently the most mature container orchestration system, which manages container instances through Pods and uses controllers (e.g., Deployment, StatefulSet) to achieve replica management and rolling upgrades. The auto-scaling mechanism of Kubernetes can be represented as:

$$N_{t+1} = f(N_t, U_t)$$

Where N_t is the number of container instances at moment t, U_t represents the current resource utilization rate, and function f decides the expansion or contraction operation based on a preset threshold. The combination of containerization and orchestration technology enables the software to be efficiently deployed in distributed environments and improves the stability and scalability of the system through automatic fault recovery and elastic scaling mechanisms.

3.3 Deploying pipeline builds

Deployment Pipeline is the core mechanism of automated deployment in DevOps system, which is responsible for coordinating code delivery, build, test, deployment and monitoring to ensure efficient and stable release of software version. Deployment Pipeline usually uses CI/CD tools (e.g., Jenkins, GitLab CI/CD, ArgoCD) for automated process management, and realizes smooth delivery of software from development to production by executing tasks in stages. A standard deployment pipeline consists of multiple phases, including code pull (Pull Code), build (Build), test (Test), release (Release), deployment (Deploy) and monitoring (Monitor). The execution process of the setup Pipeline is:

$$P = \{S_1, S_2, \ldots, S_n\}$$

Where P stands for the complete Pipeline and S_i is the i-th execution stage, the state transition of the Pipeline can be expressed as:

$$S_{i+1} = f(S_i, C_i)$$

Where C_i is the execution condition of the current phase and f is the task execution function. During the deployment process, Pipeline supports parallel task execution, such as running automated tests and security scans concurrently to reduce deployment latency.Pipeline adopts a triggering mechanism (e.g., based on code changes or timed triggers) to ensure continuous delivery and combines it with strategies such as rolling upgrades and blue-green deployments to ensure the stability of the production environment.

Figure 2. Structure of the deployed Pipeline

4. DevOps Continuous Integration and Deployment Key Technology Practices

4.1 Code Management and Version Control

Code management and version control is the foundation of DevOps continuous integration and deployment, which directly affects the development collaboration efficiency and software delivery quality. An efficient code management system can ensure the traceability of code changes, the standardization of team collaboration, and the controllability of system version, avoiding conflicts and inconsistencies in the development process. Currently, the mainstream version control tools include Git, Mercurial and Subversion (SVN), of which Git has become the preferred choice in DevOps environments due to its distributed architecture, branch management capability and efficient merging mechanism.Git adopts distributed storage and supports local commits and remote synchronization, which improves the flexibility of code management. In a Git environment, code changesets (commits) are usually uniquely identified by a hash value H(C):

$$H(C)=SHA\text{-}1(C)$$

Where C represents the content of the code change and H(C) is the unique hash value of that change, which is used to ensure code integrity and version traceability. In DevOps practice, common code management strategies include Git Flow, Trunk-Based Development, and Forking Workflow, which performs versioning through the Main branch, Development branch, and multiple Feature Branches. Git Flow manages versioning through Main, Develop, and multiple Feature Branches, and is suitable for software products with periodic releases; Trunk-Based Development emphasizes that developers submit code directly to the main branch and control feature uptime through Feature Flags to improve delivery efficiency; Forking Workflow is mainly used in open source collaboration projects, and achieves code review through the Fork and Pull/Request mechanisms. Forking Workflow is mainly used in open source collaboration projects , through the derived code base (Fork) and Pull Request mechanism to achieve code review and merge . Combined with continuous integration (CI) tools (e.g., Jenkins, GitLab CI/CD), automated builds and tests can be triggered after the code is submitted to ensure that each change does not destroy system stability. At the same time, the version control system supports Tag mechanism to mark important versions, combined with artifact repositories (such as Nexus, JFrog Artifactory) to store the build products, to realize the reproducibility of the software and fast rollback capabilities.

4.2 Automated Testing and Quality Assurance

Automation testing is the core means to guarantee software quality in DevOps system, ensuring the stability and reliability of code changes through standardized testing process and automatic execution mechanism. The automated testing system usually includes Unit Test, Integration Test, System Test and Regression Test, each of which verifies the behavior of the software at different levels of granularity. In a Continuous Integration (CI) environment, the testing process is often tied to code commits, triggering an automated Testing Pipeline. Once the tests are executed, test reports are automatically generated and key quality metrics such as Test Pass Rate, Code Coverage, Defect Rate, etc. are analyzed to ensure the quality of delivery.

TABLE I. Quality table for automated testing of the system

testing phase	Pass rate (%)	Coverage (%)	Defect rate (%)	Average execution time (s)
unit test	98.5	92.3	1.2	15

integration test	96.2	85.7	2.5	40
system testing	94.8	80.1	3.8	90
regression test	97.3	88.5	1.8	50

Combining code quality inspection with Static Code Analysis (Static Code Analysis) tools (such as SonarQube) and adopting Test-Driven Development (TDD) strategies can further improve code maintainability and reliability. The construction of an automated test system helps to reduce software defects, improve delivery efficiency, and support the efficient operation of DevOps Continuous Integration and Continuous Delivery (CI/CD) processes.

4.3 Monitoring and Performance Optimization

Monitoring and performance optimization is a key part of DevOps system to ensure system stability, availability and efficient operation. Modern monitoring systems usually use a combination of active and passive monitoring, covering multiple levels such as Infrastructure Monitoring, Application Performance Monitoring (APM), and Log Analysis, to Realize real-time observation of system operation status and abnormal warning. In a DevOps environment, common monitoring metrics include CPU utilization, memory usage, response time, error rate, and so on. Performance optimization usually relies on bottleneck analysis, cache optimization, asynchronous processing and load balancing to ensure system stability in highly concurrent scenarios.

TABLE II. Key monitoring data table of the system

Indicator name	current value	thresholds	state of affairs	note
CPU utilization rate	72%	85%	normalcy	Load Acceptable
memory footprint	68%	80%	normalcy	Sufficient memory
Response time (ms)	180	250	normalcy	Below alert value
error rate	0.9%	1.5%	normalcy	No apparent abnormality
Average number of requests (QPS)	850	1000	normalcy	Load balancing is working properly

Combined with Prometheus, Grafana, ELK (Elasticsearch, Logstash, Kibana) and other monitoring tools, it can achieve multi-dimensional data collection and visualization analysis, and improve fault early warning capability through AI-driven intelligent anomaly detection technology. Through continuous optimization and monitoring automation, DevOps teams are able to identify system bottlenecks in a timely manner, optimize resource allocation, and improve overall service quality.

5. Conclusion

This research systematically analyzes software continuous integration and automated deployment under the DevOps system, and deeply discusses the key technologies of continuous integration, automated deployment methods and their applications in practice. By introducing core technologies such as code management and version control, automated testing and quality assurance, containerization and orchestration technologies, DevOps achieves an efficient software delivery mechanism and improves the collaborative efficiency of development and operation and maintenance. Continuous integration ensures the stability of code changes through automated build, testing and code analysis, while automated deployment combined with containerization and

orchestration technology builds a flexible and scalable software delivery system. At the same time, with the help of monitoring and performance optimization technology, it can realize real-time health detection and fault warning of the system to ensure business continuity. Research shows that the comprehensive implementation of DevOps system not only improves the software delivery efficiency, but also enhances the stability and security of the system. In the future, with the development of cloud computing, artificial intelligence and edge computing, DevOps technology will further evolve in the direction of intelligence and adaptive. How to make use of intelligent operation and maintenance (AIOps), adaptive deployment strategy and automated test optimization algorithm will become the key direction for further improvement of continuous delivery system, providing stronger support for efficient innovation of software engineering.

References

[1] Gollapudi K P ,Subbian G R .Cloud Migrated Continuous Testing in DevOps: A Game-Changer for P&C Insurers[J]. Computer Science,2025,18(3):239-249.

[2] Dragomirescu A O ,Crăciun C P ,Bologa R A .Enhancing Invoice Processing Automation Through the Integration of DevOps Methodologies and Machine Learning[J].Systems,2025,13(2):87-87.

[3] Benjamin J ,Mathew J .Enhancing continuous integration predictions: a hybrid LSTM-GRU deep learning framework with evolved DBSO algorithm[J]. Computing,2024,107(1):9-9.

[4] Chava A .CI/CD and Automation in DevOps Engineering[J].Asian Journal of Research in Computer Science,2024,17(11):73-80.

[5] Eswararaj D ,Koppada R L ,Bodala S R .DevOps Implementation: Essential Tools, Best Practices, and Solutions to Overcome Challenges for Seamless Development and Operations Integration[J].Asian Journal of Research in Computer Science,2024,17(10):26-36.

[6] Amaro R ,Pereira R ,Silva D M M .Mapping DevOps capabilities to the software life cycle: a systematic literature review[J].Information and Software Technology,2025,177107583-107583.

[7] Aouni E F ,Moumane K ,Idri A , et al. A systematic literature review on Agile, Cloud, and DevOps integration: challenges, benefits[J].Information and Software Technology,2025,177107569-107569.

[8] Paulino D ,Netto T A ,Brito T A W , et al.WebTraceSense-A Framework for the Visualization of User Log Interactions[J].Eng,2024,5(3). 2206-2222.

[9] FluriJ ,FornariF ,PustulkaE .On the importance of CI/CD practices for database applications[J].Journal of Software: Evolution and Process,2024,. 36(12):e2720-e2720.

[10] Arian G .Aspects of partially automated DevOps cycles and requirements for tracking in aerospace applications[J].e+i Elektrotechnik und Informationstechnik,2024,141(3-4):167-170.

[11] Chattopadhyay R .Pega Pipeline Management: Revolutionizing DevOps Practices[J].):2206-2222.

[12] Cuadra J ,Hurtado E ,Sarachaga I , et al.Enabling DevOps for Fog Applications in the Smart Manufacturing domain: a Model-Driven based Platform Engineering approach[J].Future Generation Computer Systems,2024,157360-375.

[13] Ali M J .DevOps and continuous integration/continuous deployment (CI/CD) automation[J].Advances in Engineering Innovation,2023,4(1):38-42.

[14] Yogesh R ,Giuseppe B ,Nathaniel F .Effective DevOps with AWS:Implement continuous delivery and integration in the AWS environment[M].Packt Publishing Limited:2018-09-28.

[15] Rohin T ,Jhalak M .Mobile DevOps:Deliver continuous integration and deployment within your mobile applications[M].Packt Publishing Limited. 2018-03-29.