Journal of Artificial Intelligence and Big Data, 2021, 1, 1356
www.scipublications.com/journal/index.php/jaibd
DOI: 10.31586/jaibd.2021.1356

Review Article

Best Practices of CI/CD Adoption in Java Cloud Environments: A Review

Avinash Attipalli [1,*], Raghuvaran Kendyala [2], Jagan Kurma [3], Jaya Vardhani Mamidala [4], Varun Bitkuri [5], Sunil Jacob Enokkaren [6]

[1] Department of Computer Science, University of Bridgeport, USA
[2] Department of Computer Science, University of Illinois at Springfield, USA
[3] Computer Information Systems, Christian Brothers University, USA
[4] Department of Computer Science, University of Central Missouri, USA
[5] Software Engineer, Stratford University, USA
[6] Solution Architect, ADP, USA

*Correspondence: Avinash Attipalli (Attipalli.avinash@gmail.com)

Abstract: The continuous integration (CI) and continuous delivery/deployment (CD) methods are key tools in the field of modern software development, and they assist in the rapid, reliable and quality delivery of software. These DevOps methods are automated, and the code development, testing, and deployment processes are streamlined, which reduces the risk of integration, enhances productivity, and minimizes human labor. To implement CI/CD, Java cloud applications can utilize cloud-native services such as AWS Code Pipeline, Azure DevOps, and Google Cloud Build, as well as tools like Jenkins, GitLab CI/CD, GitHub Actions, CircleCI, Travis CI, and Bamboo. Basic concepts of CI/CD include automation, regular integration, testing, intensive testing, constant feedback, and process improvement. Some of the major pipeline phases include deployment, monitoring, testing, artefact management, build automation, and source code management. Despite clear benefits, challenges remain, including infrastructure complexity, dependency management, test reliability, and cultural barriers, particularly in large-scale or enterprise Java projects. This work provides a thorough analysis of CI/CD procedures and resources, including frameworks, best practices, and challenges for Java cloud applications. It highlights strategies to optimize adoption, improve software quality, and accelerate delivery cycles.

Keywords: Continuous Integration (CI), Continuous Delivery (CD), Continuous Deployment, DevOps, CI/CD Pipelines, Cloud-Native Services, Software Delivery

How to cite this paper:
Attipalli, A., Kendyala, R., Kurma, J., Mamidala, J. V., Bitkuri, V., & Enokkaren, S. J. (2021). Best Practices of CI/CD Adoption in Java Cloud Environments: A Review. *Journal of Artificial Intelligence and Big Data, 1*(1), 167–180.
DOI: 10.31586/jaibd.2021.1356

Received: August 30, 2021
Revised: November 28, 2021
Accepted: December 24, 2021
Published: December 27, 2021

1. Introduction

Continuous Delivery (CD) and Continuous Integration (CI) relate to deployment, a modern software development approach that emphasizes consistent and reliable incremental code changes. Automated build and test procedures ensure the correctness and stability of code being merged into the repository. The CD procedure then includes the quick and easy delivery of these verified modifications. The process of moving minor code changes from development environments to production is streamlined by the CI/CD pipeline. Businesses are spending a lot of money to develop and deliver high-quality software faster because the software market is getting more competitive. CI/CD and CD, sometimes referred to as continuous practices, enable the creation and faster software feature delivery without sacrificing quality [1]. CDE and CD emphasize that by maximizing automation assistance, CI can better integrate work-in-progress every day, value may be released to consumers fast and reliably [2].

The evolution of software development is driven by the premise that faster access to innovation enhances productivity [3]. Innovation is centred on the Web, and quick developments made possible by the Cloud have benefits over conventional technological cycles. Employees respond better to a steady flow of reasonable enhancements than to big, disruptive adjustments [4]. Bite-sized increments of gradual iterations significantly lessen the difficulties associated with change management, whereas traditional software upgrades often force employees through cumbersome relearning cycles [5]. Cloud computing is fundamentally a philosophy and architectural concept that separates applications, operating systems, and hardware, simplifying management while enabling flexibility [6]. It provides a straightforward concept for on-demand network access to a shared pool of reconfigurable computer resources that may be quickly made available and released without the involvement or control of the provider [7]. Three service models, four deployment models, and five key features make up the cloud model, which places an emphasis on availability [8].

The biggest information source nowadays is the World Wide Web. Java programming forms the core of professional foundational courses in most computer science programs and offers strong versatility. Follow-up courses often include JSP, J2EE, and Android programming. Strengthening fundamental Java skills in practice enables students to master programming techniques more effectively, and educational reforms in this direction have shown fruitful results [9]. Java-based tools for version management, auto-code generation, and HTML interface creation help both developers and end-users manage and deploy Java applications efficiently [10].

This survey highlights how CI/CD practices in Java cloud applications enable rapid, reliable, and automated software delivery, improving productivity and software quality while supporting frequent code changes. Cloud-based pipelines enhance scalability and flexibility for distributed teams, though challenges such as legacy system integration and security remain. Overall, CI/CD is a key driver of agility and efficiency in modern software development.

1.1. Organisation of the Paper

This paper is structured as follows: Section II reviews the Understanding of CI/CD, Section III, CI/CD Tools and Frameworks for Java, Section IV outlines Best Practices and Challenges in CI/CD for Java Cloud Application, Section V presents a literature review, and Section VI brings important conclusions and future directions to a close.

2. Fundamentals of CI/CD

2.1. CI and cloud-based CD are two examples of modern software development methodologies that seek to improve the reliability, speed, and quality of software delivery

These techniques are crucial to DevOps procedures because they enable frequent release of changes and continuous integration. Definition of Continuous Integration (CI), Continuous Delivery/Deployment (CD)

Continuous Integration (CI): In the process of developing software, called "continuous integration," team members regularly integrate their work and automate the processes of creation, testing, and validation. This helps detect and fix bugs early, improving overall software quality. Approximately 70% of students noted that CI enhanced their overall development workflow, as shown in Figure 1. The three main components of CI server, version control system, and source repository. According to Amazon Web Services (AWS), CI adoption is hampered by the need to automate builds, automate testing, maintain a single source repository, and update a shared codebase often. Build automation, code stability, analytics, allowing CD, quicker releases, cost savings, increased productivity, and higher-quality code are some of the main advantages of CI.

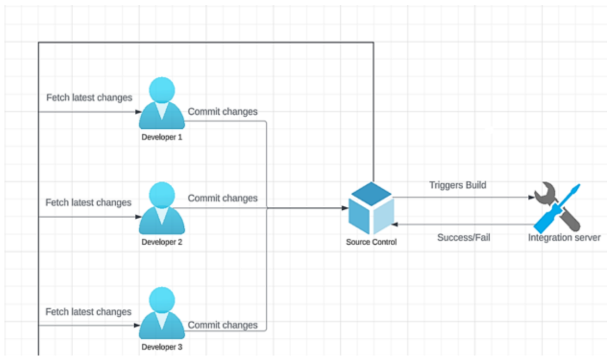

Figure 1. Continuous Integration (CI) workflow

Continuous Deployment/Delivery (CD): Code updates are automatically created, tested, and delivered as part of the process for developing CD software. CD focuses on maintaining code in a deployable state that may be released at any time, whereas CD refers to the automatic delivery of code to production. The goals of both approaches are to increase development efficiency, software quality, and release speed [11]. CD makes it possible to quickly and safely roll out experiments, configuration changes, problem repairs, and new features. Due to benefits like increased productivity, dependable releases, enhanced speed to market, producing the ideal product, and ensuring customer satisfaction, investment in CD is rising in multi-customer project courses where it is assessed for usage, experiences, and benefits (Figure 2):

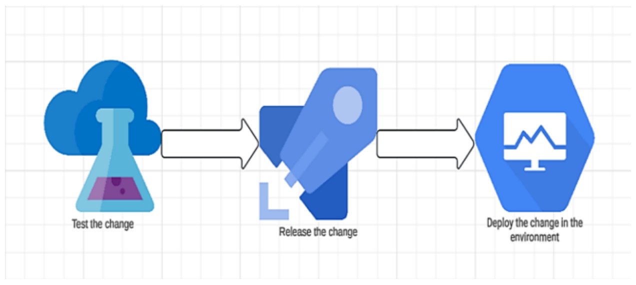

Figure 2. Continuous Deployment Workflow (CI)

2.2. Key benefits and challenges of CI/CD

A fundamental component of contemporary software development is CI/CD, offering automation, increased productivity, improved quality, and faster delivery. However, its adoption also brings challenges that organizations must manage effectively.

- **Automate the Software Release Process:** CD provides a means for the team to verify automatically generated and tested code [12], and ready for production release to provide a secure, quick, robust, and effective software delivery.
- **Improve Developer Productivity:** Developers can concentrate on producing new features and creating high-quality code instead of managing integrations or deployments thanks to CI/CD, which removes tedious manual chores, simplifies deployment, and untangles complicated dependencies.
- **Improve Code Quality:** Automated testing integrated into the pipeline, bugs and issues are identified early before they grow into critical problems. Frequent testing and continuous feedback loops enhance stability, security, and reliability, ultimately leading to higher overall code quality.
- **Deliver Updates Faster:** CI/CD has a great effect on the speed of release. Businesses are able to react swiftly to client demands, security threats and

market requirements [13]. One can now roll out features and fixes in a matter of days or even a few hours rather than weeks or months.

- **Reduced Integration risk:** The frequent integrations would minimize the possibility of significant conflicts in the codebase. System problems are identified fast and debugging becomes simpler and rollbacks are not allowed to be extensive.

2.2.1. Key Challenges of CI/CD:

- **Initial Setup Complexity:** The deployment of CI/CD pipelines takes a lot of time, talent, and effort. Organizations have to come up with pipelines that are designed to fit their architecture, technology stack, and business needs.
- **Infrastructure and Tooling Costs:** CI/CD can tend to require a strong infrastructure and third-party solutions that can raise the cost. Scalable build server maintenance, cloud services and testing environments may be costly.
- **Security Concerns:** The continuous deployment may unknowingly cause weak points in the pipeline unless security procedures are included. It is important to make sure that the builds are secure, dependencies are controlled, and access controls.
- **Test Reliability and Maintenance:** Automated tests are essential, yet flakey tests or shoddily designed tests may generate false positives/negatives, slowing the delivery. It is an ongoing task to have a good test suite.
- **Cultural and Skill Barriers:** CI/CD implementation typically necessitates a shift in team culture. Automation and constant cooperation should be welcomed, which may be challenging in traditional setups involving developers, testers, and operations.

2.3. Core Principles and Stages of CI/CD Pipelines

Software development, testing, and deployment are automated using CI/CD pipelines to provide faster, more reliable, cheaper [14]. They encompass such significant stages as deployment, monitoring, testing, artefact management, build automation, and source code management.

2.3.1. Core Principles of CI/CD Pipelines

Here are the core principles of CI/CD pipelines are as follows:

- **Automation:** Deployment process, testing and building process Automating to minimize human error and increase productivity. The use of automation technologies enhances the efficiency of repetitive operations and also provides consistency in these tasks, and minimizes chances of errors.
- **Frequent Integration:** This is through frequent code updates to detect and correct issues in the early stages. This notion supports small, incremental changes, which are easier to spot and deal with problems.
- **Testing:** Including automated testing at different phases to guarantee that the code is both functional and of the appropriate calibre. The range of testing includes end-to-end, integration, and unit tests.
- **Feedback:** Engaging developers with prompt feedback that allows them to correct the issues as fast as possible. The feedback loops help the developers to understand the impact of their changes and make the necessary corrections as fast as possible.
- **Continuous Improvement:** Refining procedures with time to make them more reliable and efficient. To do so, the CI/CD processes should be revised on a regular basis and enhanced to meet the changing needs and technology.

2.3.2. Core stages of CI/CD Pipelines

Here are the core stages of CI/CD pipelines are as follows:

- **Source Code Management (SCM):** The pipeline entails the use of a source code repository (e.g., Git, SVN, Mercurial), which acts as the core collaboration centre. The pipeline is executed with each commit or pull request, ensuring that integration is both gradual and continuous.
- **Build Automation:** The stage of building and packaging the source code in the build stage leads to deployable artefacts. In the case of Java applications, it is typical to use Maven, Gradle or Ant build tools. Automation of building creates a consistent versioned and managed dependency.
- **Automated Testing:** The purpose of this stage is to evaluate each unit's code quality, integration and system, regression testing. Dynamical frameworks (e.g., JUnit, TestNG, Mockito) ensure functionality and reliability and the quality of the code is evaluated using tools of static analysis (e.g., SonarQube, Checkstyle). Passing the test offers timely feedback and serves as a quality assurance for deployment.
- **Artefact Management:** Compiled artefacts are kept in artefact repositories like AWS S3, Nexus Repository, or JFrog Artifactory. This enables traceability, reuse, and version control of deployable packages across environments.
- **Deployment Automation:** In this stage, artefacts are deployed to testing, staging, or production environments. Deployment automation often relies on containerization (such as Docker) and orchestration (e.g., Kubernetes, Helm) to achieve scalability and consistency.
- **Monitoring and Feedback:** The final stage integrates logging, monitoring, and feedback mechanisms. Tools like Grafana, ELK Stack, and Prometheus, or Splunk, track performance and system health, while feedback loops ensure rapid detection and rollback of faulty releases.

2.4. Role and importance of Java in the Cloud Environment

Java has emerged as a leading language for cloud environments due to its platform independence, mature ecosystem, and strong support for enterprise applications. Frameworks such as Spring Boot, Spring Cloud, Quarkus, and Micronaut facilitate the development of cloud-native and microservices architectures, while JVM optimizations like GraalVM and OpenJ9 enhance startup times and memory efficiency for containerized and serverless deployments. Java seamlessly integrates with public and AWS, Azure, Google Cloud, OpenShift, and Cloud Foundry are examples of private cloud systems, supporting deployment automation, orchestration, and service discovery for scalable, resilient, and distributed systems [15]. However, challenges remain, such as performance overheads, JVM tuning in resource-constrained environments, and the complexity of migrating legacy monolithic applications. Despite these, Java's evolving ecosystem and lightweight, serverless-ready frameworks continue to make it a robust choice for enterprise-grade cloud applications.

3. CI/CD Tools and Frameworks for Java

Java applications are supported by several CI/CD systems and tools that automate the deployment, testing, and building processes. CircleCI, GitHub Actions, Jenkins, and GitLab CI/CD are popular options, and Bamboo integrates with Java ecosystems, supports distributed builds, and enables scalable delivery pipelines. These tools help streamline workflows, improve feedback cycles, and enhance reliability in Java cloud application deployments.

3.1. Popular Continuous integration and deployment (CI/CD) tools for Java projects

Numerous tools and platforms play a major role in facilitating CI/CD, is used in cloud Java applications. Scalability, dependability, and efficiency are guaranteed by these technologies, which provide automation in the creation, evaluation, and implementation of applications [16]. This section reviews both popular open-source CI/CD tools widely used in Java projects and cloud-native CI/CD services offered by major cloud providers, followed by a comparative analysis.

3.1.1. Jenkins

Jenkins is an open-source, cross-platform CI/CD technology with a Java foundation that facilitates real-time testing, reporting, continuous integration, and delivery. Installing it requires downloading the .war file and executing it from the console. Jenkins follows a master–worker architecture: the master schedules jobs, delegates execution to workers (agents), monitors their status, and aggregates results on the dashboard [17]. Workers, deployed as Java executables on remote machines, execute assigned jobs and can be dynamically added or removed, ensuring workload distribution and scalability.

Salient features of Jenkins

- Compatible with Linux, macOS, and Windows. It is better for large organizations and startups because it is free and open-source.
- GitHub has 22.1K stars and 8.4K forked repositories. Incredibly extensible. With more than 1,500 plugins, it has a flourishing plugin ecosystem and the greatest community in its class.
- Connects to well-known cloud platforms including Digital Ocean, AWS, Google Cloud, Azure, and others.
- It may be used to accomplish difficult CD needs and operate in simultaneously.
- The installer, which comes in .war format, is a standalone Java application that functions immediately.

3.1.2. GitLab CI/CD

The GitLab source code management (SCM) system and GitLab CI/CD are closely related. Code testing, development, and deployment may be automated with its robust and adaptable framework. Pipelines specified in an area how GitLab CI/CD works. At a repository's root is the GitLab-ci.yml file [18]. This file outlines the various pipeline phases, as well as the specific jobs or activities that must be completed at each level.

Salient Features of GitLab CI

- GitLab CI provides developer APIs so that outside developers may further integrate their solutions.
- It is compatible with widely used operating systems such as Linux, macOS, and Windows. The web application for GitLab CI features an intuitive user interface.
- To reduce build time, GitLab CI distributes a single build over several computers in parallel.
- GitLab CI's caching system saves time while tasks are executing. Can deactivate the cache on particular tasks and distribute the cache between their branches as well as between branches. Given the abundance of GitLab CI choices, the caching method may be used as needed.
- GitLab CI jobs can execute in a parallel or a sequential fashion. Additionally, it provides the ability to specify a unique pipeline. Using GitLab CI instead of a program like Jenkins or CircleCI is easy.
- GitLab CI's shell executor makes it simple to use by allowing builds to be started.

3.1.3. GitHub Actions

GitHub Actions is a relatively new CI/CD tool that was introduced in 2018. It may quickly and simply construct custom SDLC workflows in GitHub repository by using GitHub Actions [19]. This can be achieved by utilizing various tasks or actions that run automatically in response to specific events. Developers would want to utilize this tool for the additional reasons listed below.

Salient Features of GitHub Actions

- This includes developing, sharing, reusing, and forking software development processes.
- Since GitHub and it are completely connected, can manage it from one location.
- It has added support for Docker, enabling multi-container testing.
- It may generate its own CI template or select from a variety of existing ones.
- Provide each of the repositories 2000 free build minutes every month.

3.1.4. CircleCI

CircleCI is among the best technologies for CI/CD implementation in large-scale, open-source projects. CircleCI Server is its on-premises version, whereas CircleCI Cloud is a cloud-based solution. The languages that can be developed on Windows, Linux, and macOS are supported. Its pipelines use a unique YAML syntax, and it is easy to set up. CircleCI was recognized by Forrester Wave as a pioneer in Cloud Native continuous integration in 2019.

- Easily configured and compatible with a number of well-known version-controlling platforms, including Bitbucket, GitHub, and more.
- It comes pre-configured with support for most of the most widely used programming languages.
- Builds can be divided and distributed over several containers to shorten the build time.
- CircleCI's parallel testing feature allows tests to run concurrently across several executors.
- Timing data may be used to split tests, which further minimizes The duration required for the execution of the test.
- CircleCI Server is compatible with popular cloud computing platforms, including AWS, Google Cloud, Azure, and others.
- CircleCI Orbs are reusable code segments that expedite the integration of third-party technology and enable the automation of repetitive activities.

3.1.5. Travis CI

Travis CI was a pioneer in the market for CI/CD pipeline systems, much like Jenkins. Open-source projects were initially permitted to use it until it was shifted to support closed-source projects. Travis CI, which is developed in Ruby and is perfect for projects hosted on Bitbucket or GitHub, is one of the most effective CI/CD solutions for enterprise-level and open-source projects [20]. Like CircleCI, Travis CI offers a variety of options for businesses and the open-source community that want to use Travis CI on their private cloud or self-hosted infrastructure. The differences between the two leading CI/CD technologies are highlighted in a blog post that contrasts CircleCI with Travis CI.

Salient features of Travis CI

- Travis CI is compatible with 30 different programming languages, including Python, Java, C#, Julia, and more.
- Travis CI Enterprise is a simply, connectable self-hosted platform with Bitbucket and GitHub for businesses seeking more secure and private solutions.
- CI/CD pipelines integrate seamlessly with GitHub Enterprise products and employ a proprietary YAML syntax.
- Travis CI's Cloud version is appropriate for small-team businesses and open-source projects.

- It offers runtimes for popular operating systems, including Linux, macOS, and Windows.
- Travis CI's build matrix functionality enables conducting Parallel builds that rely upon a wide range of runtime, language, and environment combinations.
- Travis CI Enterprise facilitates interaction with well-known cloud computing platforms, including Azure, AWS, Google Cloud, Kubernetes, and others.

3.1.6. Bamboo

Bamboo is a well-liked CI tool. The company that created Jira, Atlassian, created this corporate offering, which offers a single window for carrying out builds, testing, and releases. It integrates easily with Bitbucket and Jira, two well-known SCM technologies. It is also compatible with widely used operating systems, such as Linux, macOS, and Windows. Like other well-known CI/CD pipeline systems, Bamboo is compatible with several different programming languages and technologies, such as Git, SVN, AWS, and others.

Salient Features of Bamboo

- Bamboo makes it easy to switch from Jenkins is an open-source CI/CD application on its platform.
- It is integrated with Bitbucket and Jira software by default.
- Bamboo is compatible with a number of well-known technologies and platforms, including Docker and AWS.
- By running builds on distant build agents, it is able to achieve parallel builds. Multiple concurrent agent test batches and up to 100 remote build agents are available.
- There are two versions of Bamboo: self-hosted and cloud-based. The builds can then be triggered using Bamboo according to the updates made to the repository, and the push alerts can be delivered using Bitbucket.

The CI/CD landscape is primarily defined by a trade-off between flexibility and integration, with each major tool catering to a specific organizational need. Jenkins stands as the veteran, offering unparalleled customization and a massive plugin ecosystem (over 1,800) that allows it to integrate with virtually any technology stack or self-hosted environment; however, this power comes at the cost of a steep learning curve and significant maintenance overhead. In contrast, GitLab CI/CD and GitHub Actions represent the modern, all-in-one approach, providing deeply integrated CI/CD directly within their respective Git platforms, simplifying setup, leveraging YAML-based pipeline-as-code, and offering built-in features like security scanning (especially strong in GitLab). CircleCI differentiates itself as a cloud-native solution renowned for its speed, robust support for Docker, and excellent parallelism, making it a favorite for microservices and teams prioritizing fast feedback loops. Bamboo is the tool of choice for organizations already deeply invested in the Atlassian ecosystem, offering seamless, out-of-the-box integration with Jira and Bitbucket for enterprise-grade traceability, though it is a paid commercial tool. Finally, Travis CI, while an early cloud CI pioneer known for its simplicity and close integration with GitHub, has seen its relevance somewhat diminished by the powerful, native capabilities of GitHub Actions.

3.2. Cloud-native CI/CD services

These services are crucial in automating the entire development, testing, and deployment cycle of Java applications in cloud infrastructures. Also compatible with Maven, Gradle, Docker, and Kubernetes, managed pipelines are provided by AWS CodePipeline, Azure DevOps, and Google Cloud Build. They guarantee scalability, protection and more rapid delivery in cloud setups, but can also result in vendor lock-in.

3.2.1. AWS CodePipeline

AWS CodePipeline provides deployed CI/CD pipelines that are fully operational in the AWS environment. It does not support any alternative projects other than Java through AWS Codebuild and Elastic Beanstalk, and can easily deploy to EC2, ECS, or Lambda. It offers high availability, security, and integrates with AWS services; however, it also introduces vendor lock-in.

3.2.2. Azure DevOps

Azure DevOps offers the integration of Java projects by offering pipelines as a service. It has support for Maven, Gradle, JUnit, and Docker deployments to Kubernetes and Azure App Services. It is strong in hybrid and enterprise systems that have Microsoft services and Java-based systems.

3.2.3. Google Cloud Build

Google Cloud Build is a serverless CI/CD service offering on-demand build and deployment capabilities. For Java applications, it supports containerized builds using Docker and Kubernetes (GKE). Its integration with Google Cloud services and scalability make it attractive for cloud-native Java workloads.

4. Best Practices and Challenges in Ci/Cd for Java Cloud Application

Successful adoption requires robust testing, branching, modular design, and effective release strategies, while challenges such as test gaps, architectural complexity, database dependencies, and diverse environments often slow down CI/CD in Java cloud projects.

4.1. Best Practices in Ci/Cd for Java Cloud Applications

This category presents four core practices to allow and encourage successful CI/CD adoption, it is necessary to improve testing activities, implement branching mechanisms that work, break down development into smaller units, and include consumers:

4.1.1. Improve Testing Activity

Enhancing the testing phase is critical for effective CI. Practices such as using daily builds and test-driven development (TDD) are important. Test planning, where collaboration between developers and QA engineers produces a comprehensive suite of automated tests, helps reduce the burden of regression testing. Cross-team testing, in which integration testing is performed by individuals not involved in implementation, helps detect more defects and ensures unbiased evaluations. In order to increase productivity, it has also been proposed to separate unit testing from functional and acceptability tests.

4.1.2. Branching Strategies

Branching strategies are crucial for supporting CI. Repository management and short-lived feature branching have proven effective, with the latter also facilitating CD. Shorter branches allow faster exposure of new features and quicker feedback. Practices such as committing changes locally and synchronizing with a central repository ensure stability, particularly when code is committed only after passing automated tests [21]. Conversely, maintaining multiple long-lived branches can hinder CI adoption, whereas a single stable branch strategy minimizes merging efforts.

4.1.3. Decompose Development into Smaller Units

Breaking development into smaller, independent units helps reduce build and test times, thereby accelerating the feedback loop. Large features should be decomposed into smaller increments for safer and faster testing [22]. Dependency management patterns such as interface modules, platform-independent modules, and native modules can simplify build scripts and reduce complexity in cross-platform applications. Dead code practices, where components are activated only when dependencies are ready, further reduce integration issues. Smaller units also enable cross-functional teams, which facilitate the adoption of CI and CD.

4.1.4. Partial Release

Releasing software continuously can be risky, as customers may encounter unstable features. To mitigate this, several practices are recommended: deploying to a small subset of users, hiding or disabling unfinished features, and enabling fast rollback. Incremental release strategies, such as canary deployment and dark deployment, are widely adopted, enabling businesses to test new features in safe settings prior to a full release. Additional practices include using microservices to minimize pipeline delays and adopting empty releases to help teams familiarize themselves with CD workflows.

4.1.5. Customer Involvement

Customer involvement is a key enabler of CD. Practices such as pilot or lead customer programs enable early adopters to test frequent updates and provide rapid feedback [14]. Continuous SCRUM models incorporate triage meetings to dynamically prioritize requirements. Involving customers in testing also helps organizations with limited QA resources, as customers can assist in detecting functional issues and validating features.

4.2. Challenges in CI/CD for Java Cloud Applications

Despite the maturity of CI/CD practices, Java cloud applications face several critical challenges that hinder seamless automation and deployment. These challenges arise due to the inherent complexity of Java ecosystems, cloud-native requirements, and organizational constraints. Key challenges include:

- Many Java applications struggle with testing due to incomplete automation, complex environments, and strong code test coupling. Poor test quality, including unreliable and slow tests with low coverage, further slows feedback and reduces confidence in deployments [23]. UI-level tests often require manual validation, making testing a major bottleneck in CI/CD adoption.
- Many Java applications face difficulties in implementing effective testing strategies, primarily due to incomplete automation caused by infrastructure limitations, complex test environments, and reliance on manual processes [24]. The strong coupling between code and tests further complicates integration, particularly in large-scale enterprise projects.
- Integration bottlenecks: Java applications often depend on multiple third-party libraries, frameworks, and APIs. Incompatibility among these dependencies, along with tightly coupled design, leads to frequent merge conflicts. Teams must spend significant effort resolving these conflicts, slowing down CI pipelines.
- Java enterprise systems often suffer from monolithic or tightly coupled architectures, where changes in one component ripple across teams and modules, creating coordination overhead. Also, the Java cloud applications with regular changes of schema in their database are bottlenecks during deployment, as even a small change in code can necessitate significant changes. Automation of environments in order to deal with such dependencies is still a major challenge to the adoption of CI/CD.
- Java cloud applications are implemented using various infrastructure, middleware and configurations; hence, it is difficult to test in production-like conditions. The compatibility with several versions and integration with legacy systems also make CD further complicated.

- The field of deployment is another area that has a major influence on CI/CD feasibility. Whereas Java web applications may follow a fast-release cycle, requirements in Java systems in other fields like telecommunications, embedded systems, and safety-critical industries are more rigid and thus restrict the frequency of releases [25]. These limitations frequently result in non-continuous deployment in the form of calendar-based or staged release, rather than complete continuous deployment.

Integrating security scanning tools like **Static Application Security Testing (SAST)** and **Dynamic Application Security Testing (DAST)** directly into the **Continuous Integration/Continuous Delivery (CI/CD) pipeline** is the foundation of **DevSecOps**, transforming security from a reactive gate to a continuous, automated process. **SAST** runs early in the CI phase, analyzing proprietary source code, bytecode, or binary code *without* executing the application, identifying vulnerabilities such as SQL injection or buffer overflows as soon as code is committed. This "shift-left" approach provides developers with immediate feedback, making flaws cheaper and easier to fix. Subsequently, **DAST** tools execute against the running application in a testing or staging environment (usually in the CD phase), simulating attacks from the outside to identify runtime issues like configuration errors, authentication flaws, and exposed APIs.

5. Literature Review

This literature review on CI/CD Practices for Java Cloud Applications outlines important developments and factual discoveries, and technological developments, offering insights to guide future research and real-world implementations.

Guseila, Bratu and Moraru (2019) demonstrate how DevOps is a software development methodology that helps businesses automate processes to deliver software product features, encourage teamwork, and boost productivity throughout the process of software lifecycle management, opening the door to cloud computing and supporting digital transformation. Additionally, by tracking and measuring KPIs for an operational system, DevOps depends on automation in operations. The study illustrates the importance of DevOps in the transformation of IT service delivery by utilising a DevOps assessment to determine software application maturity and DevOps adoption through automated testing and the CI/CD pipeline. For future research, the study provides a conceptual knowledge of the CI/CD pipeline based on agile tools [26].

Cheon (2019) provide a native method for creating a multiplatform application that runs on Java and Android, two comparable yet distinct systems. From tool configuration to program design and development, they tackle real-world software engineering issues related to native multiplatform application development. Method improves the quality of the program while allowing 37% to 40% of the application code to be shared between the two platforms. They think method can be modified to convert current Java apps to Android apps as well [27].

Singh et al. (2019) These microservices may be set up as serverless functions like AWS Lambda, virtual machines, or Docker containers. Nevertheless, managing and deploying microservices gets more difficult as their number rises. Thus, the problem is resolved by employing CI and CD systems to deploy the microservices with the least amount of downtime possible. In this study, a number of CI and CD systems were examined, accounting for server monitoring, pipeline integration, cloud compatibility, and performance monitoring following deployment [28].

Utomo et al. (2018) seek to create a system of information that may be utilised to sell goods made by MSMEs. Sales through online platforms are anticipated to boost SME furniture income and boost the worth and competitiveness of related sectors. Using the generic data required to hold the ordering information and the completed transactions, the information system is created. This study's development approach, Rapid Application Development (RAD) uses MySQL databases and the PHP programming language [29].

Xiao (2018) demonstrates the process of downloading and setting up the Java Development Kit (JDK). It presents well-known Java integrated development environments (IDEs) and text editors, demonstrating how they manage basic Java "Hello World" applications. As with any programming language, technology experts use an IDE to build Java programs, which may greatly simplify the process. In addition to text editors and IDEs, there are a number of online compilers, so IT workers do not need to download and install the Java JDK or an IDE on their machines. One programming language's source code for an application can be converted into another by using online code converters [30].

Bobrovskis and Jurenoks (2018) include confirming and summarizing the present situation of CI/CDE/CD by market research and a study of pertinent literature. It is the first stage in creating and implementing strategies for automated resource allocation in a programmable environment. An organization's ability to implement updates, enhancements, and repairs in real-time with agility and accuracy is a critical component of modern information technology (IT) operations. Using automated techniques becomes increasingly necessary as technology advances. Agile approaches to software development have supplanted waterfall software development techniques in recent decades. There is little question that this change brings software engineers, operations, and companies closer together in pursuit of the common goals [31].

Table 1 provides a comparative overview of recent studies, outlining their approaches, key findings, implementation challenges, and proposed future directions to increase the use and efficacy of CI/CD practices for cloud Java applications.

Table 1. Literature Summary on Java-Based Big Data Frameworks

Reference	Study on	Approaches	Findings/Insights	Challenges	Future Work
Guseila, Bratu et.al. (2019)	Role of DevOps in software development and IT service delivery	DevOps methodology, CI/CD pipeline, automated testing, KPI monitoring	DevOps enhances automation, team collaboration, efficiency, and digital transformation. Maturity assessment of DevOps adoption was demonstrated conceptually.	Assessing DevOps maturity in existing systems; integrating CI/CD pipelines effectively	Implementation of the conceptual CI/CD pipeline in practical environments
Cheon et.al. (2019)	Native approach for multiplatform application development (Java & Android)	Code-sharing techniques, tool configuration, software design improvements	37–40% of code can be shared across platforms; quality of applications improved; approach adaptable to migrating Java apps to Android	Managing differences between similar platforms; tool and design configuration complexity	Extend the approach to more platforms and real-world applications
Singh et al. (2019)	Deployment and management of microservices	Serverless functions, virtual machines, Docker, and tools for continuous integration (CI) and continuous delivery (CD)	CI/CD tools reduce downtime and streamline deployment of microservices; comparison of CI/CD tools based on monitoring, integration, cloud compatibility	Managing large numbers of microservices and ensuring efficient deployment	Evaluate CI/CD tool performance under varied cloud environments

Utomo et al. (2018)	Information system for online sales of MSME products	Rapid Application Development (RAD) methodology, PHP, MySQL	Online sales platforms help boost competitiveness and SME furniture income	Data management, system scalability, integration with online transactions	Extend system features and integrate with larger e-commerce platforms
Xiao et.al. (2018)	Java programming environment setup and development	Introduction of JDK, IDEs, online compilers	IDEs simplify programming; online compilers remove the need for local setup; source code conversion across languages is feasible	Beginners need guidance in choosing appropriate IDE or compiler; compatibility issues with different platforms	Expand to advanced IDE features and multi-language development tools
Bobrovskis et.al. (2018)	Automated resource distribution and CI/CD practices in IT	Market research, literature analysis, programmable environment concepts	Agile and CI/CD practices enable error-free live deployment; automation needs are increasing; shift from waterfall to agile improves collaboration	Implementing automation at scale; integrating with existing IT practices	Develop practical methods for automated resource distribution and CI/CD implementation

6. Conclusion and Future Work

In contemporary software development, Continuous delivery/deployment (CD) and continuous integration (CI) have emerged as essential practices, particularly for cloud Java applications. CI/CD pipelines greatly increase software quality, speed up release cycles, lower integration risks, and boost developer productivity through the automation of the testing, deployment, and code development procedures. AWS CodePipeline, Azure DevOps, Google Cloud Build, and other cloud-native services, together with well-known CI/CD solutions like Travis CI, CircleCI, GitHub Actions, Jenkins, GitLab CI/CD, and Bamboo, are used to show how scalable, adaptable, and efficient they are in a variety of cloud environments. Core principles, including frequent integration, automated testing, rapid feedback, and continuous improvement, ensure continuous and dependable software delivery. However, organizations still face challenges such as infrastructure complexity, test reliability issues, dependency management, and cultural or skill barriers. These challenges are particularly pronounced in large-scale enterprise Java applications, monolithic architectures, and multi-environment deployments.

Future research and practice in CI/CD for Java cloud applications can focus on integrating advanced automation, AI-driven testing, and predictive analytics to further reduce failures and optimize pipeline performance. Enhancing support for legacy systems, monolithic-to-microservices migrations, and multi-cloud deployments will improve adoption across varied organizational contexts.

References

[1] B. Fitzgerald and K.-J. Stol, “Continuous software engineering: A roadmap and agenda,” *J. Syst. Softw.*, vol. 123, pp. 176–189, 2017, doi: https://doi.org/10.1016/j.jss.2015.06.063.

[2] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.

[3] A. N. Ansari, S. Patil, A. Navada, A. Peshave, and V. Borole, “Online C/C++ compiler using cloud computing,” in *2011 International Conference on Multimedia Technology*, IEEE, Jul. 2011, pp. 3591–3594. doi: 10.1109/ICMT.2011.6002124.

[4] S. S. S. Neeli, “Serverless Databases: A Cost-Effective and Scalable Solution,” *Int. J. Innov. Res. Eng. Multidiscip. Phys. Sci.*, vol. 7, no. 6, p. 7, 2019.

[5] D. Bhuriya and A. Sharma, "Study on Pros , Cons and Application of Cloud Computing," *Study Pros, Cons Appl. Cloud Comput.*, vol. 6, no. 2, pp. 959–964, 2019.

[6] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. M. A. Hashem, "Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches," *Internet of Things*, vol. 7, Sep. 2019, doi: 10.1016/j.iot.2019.100059.

[7] P. Pathak, A. Shrivastava, and S. Gupta, "A Survey on Various Security Issues in Delay Tolerant Networks," *J. Adv. Shell Program.*, vol. 2, no. 2, pp. 12–18, 2015.

[8] S. Chandra.V, D. Charan.K, and S. Rani.P, "Online C, C++ & Java Compilers using Cloud Computing," *Int. J. Comput. Sci. Mob. Comput.*, vol. 4, no. 8, pp. 348–355, 2015.

[9] L. Cui, "Research on Educational Reform of Java Programming," 2016. doi: 10.2991/mcei-16.2016.147.

[10] J. B.A. and K. A. Bhosale, "Research Paper on Java Interactional Development Environment Programming Tool," *IARJSET*, vol. 4, no. 4, pp. 121–124, Jan. 2017, doi: 10.17148/IARJSET/NCIARCSE.2017.35.

[11] L. Chen, "Continuous Delivery: Overcoming adoption challenges," *J. Syst. Softw.*, vol. 128, pp. 72–86, Jun. 2017, doi: 10.1016/j.jss.2017.02.013.

[12] Y. SKA and J. P, "A Study And Analysis Of Continuous Delivery, Continuous Integration In Software Development Environment," *J. Emerg. Technol. Innov. Res.*, vol. 6, no. 9, 2019.

[13] A. Balasubramanian, "Ai-Enabled Demand Response: A Framework For Smarter Energy Management," *Int. J. Core Eng. Manag.*, vol. 5, no. 6, pp. 96–110, 2018.

[14] R. Nirek, "Challenges and Solutions for Implementing CI/CD Pipelines in Linux-Based Development Frameworks," *J. Sci. Eng. Res.*, vol. 6, no. 6, pp. 229–232, 2019, doi: 10.13140/RG.2.2.20819.80161.

[15] S. Gupta, N. Agrawal, and S. Gupta, "A Review on Search Engine Optimization: Basics," *Int. J. Hybrid Inf. Technol.*, vol. 9, no. 5, pp. 381–390, May 2016, doi: 10.14257/ijhit . 2016.9.5.32.

[16] S. M. S and S. H. M, "Setting Up CICD Pipeline For Web Development Project in," *Int. J. Res. Eng. Appl. Manag.*, vol. 05, no. 02, 2019, doi: 10.35291/2454-9150.2019.0091.

[17] E. Stellwagen and L. Tashman, "ARIMA: The Models of Box and Jenkins," *Foresight Int. J. Appl. Forecast.*, no. 30, pp. 28–33, 2013.

[18] P. K. Koppanati, "Building Custom CI/CD Pipelines for Java Applications in GitLab," *J. Sci. Eng. Res.*, vol. 6, no. 6, pp. 233–238, 2019.

[19] V. Cosentino, J. L. Canovas Izquierdo, and J. Cabot, "A Systematic Mapping Study of Software Development With GitHub," *IEEE Access*, vol. 5, pp. 7173–7192, 2017, doi: 10.1109/ACCESS.2017.2682323.

[20] C. Renggli *et al.*, "Continuous Integration of Machine Learning Models with ease.ml/ci: Towards a Rigorous Yet Practical Treatment," pp. 1–19, Mar. 2019.

[21] D. Taibi, V. Lenarduzzi, and C. Pahl, "Continuous Architecting with Microservices and DevOps: A Systematic Mapping Study," in *Communications in Computer and Information Science*, vol. 1073, 2019, pp. 126–151. doi: 10.1007/978-3-030-29193-8_7.

[22] S. A. I. B. S. Arachchi and I. Perera, "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management," in *2018 Moratuwa Engineering Research Conference (MERCon)*, IEEE, May 2018, pp. 156–161. doi: 10.1109/MERCon . 2018.8421965.

[23] J. Gao, X. Bai, and W. Tsai, "Cloud Testing - Issues, Challenges, Needs and Practice," *Softw. Eng. An Int. J.*, vol. 1, no. 1, pp. 9–23, 2011.

[24] Y. Zhang, B. Vasilescu, H. Wang, and V. Filkov, "One size does not fit all: an empirical study of containerized continuous deployment workflows," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 295–306. doi: 10.1145/3236024.3236033.

[25] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "The making of cloud applications: an empirical study on software development for the cloud," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 393–403. doi: 10.1145/2786805.2786826.

[26] L. G. Guseila, D. V. Bratu, and S. A. Moraru, "DevOps Transformation for Multi-Cloud IoT Applications," in *2019 International Conference on Sensing and Instrumentation in IoT Era, ISSI 2019*, 2019. doi: 10.1109/ISSI47111.2019.9043730.

[27] Y. Cheon, "Multiplatform application development for Android and Java," in *Proceedings - 2019 IEEE/ACIS 17th International Conference on Software Engineering Research, Management and Application, SERA 2019*, 2019. doi: 10.1109/SERA.2019.8886800.

[28] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of Different CI/CD Tools Integrated with Cloud Platform," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, IEEE, Jan. 2019, pp. 7–12. doi: 10.1109/CONFLUENCE.2019.8776985.

[29] R. B. Utomo, A. Akbar, M. Andriansyah, Lasminiasih, and S. S. Utami, "Development of E-Commerce Applications based on RAD Methods for MSMEs Furniture Business in Central Java," in *2018 2nd International Conference on Electrical Engineering and Informatics (ICon EEI)*, IEEE, Oct. 2018, pp. 75–80. doi: 10.1109/ICon-EEI.2018.8784333.

[30] P. Xiao, "Getting Started with Java Programming," in *Practical Java® Programming for IoT, AI, and Blockchain*, 2018. doi: 10.1002/9781119560050.ch2.

[31] S. Bobrovskis and A. Jurenoks, "A survey of continuous integration, continuous delivery and continuous deployment," *CEUR Workshop Proc.*, vol. 2218, pp. 314–322, 2018.