INTERNATIONAL JOURNAL OF LATEST TECHNOLOGY IN ENGINEERING, MANAGEMENT & APPLIED SCIENCE (IJLTEMAS)

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Special Issue | Volume XIV, Issue XIII, October 2025

Practicing Continuous Integration Continuous Delivery on AWS

Sonali P Lohar*, Padmaja D Kadam

Department of Computer Science, Dr. D. Y. Patil Arts Commerce Science College, Pimpri, Pune -18, Maharashtra, India

DOI: https://doi.org/10.51583/IJLTEMAS.2025.1413SP006

Received: 26 June 2025; Accepted: 30 June 2025; Published: 22 October 2025

Abstract: The rapid-fire elaboration of software development methodologies has needed the relinquishment of robotization-driven practices to enhance software quality and delivery speed. Continuous Integration (CI) and Continuous Delivery (CD) are foundation practices in ultramodern DevOps channels, enabling brigades to integrate law constantly, descry issues beforehand, and emplace operations fleetly with minimum homemade intervention. This exploration paper investigates the perpetration and optimization of CI/ CD channels using Amazon Web Services (AWS), a leading pall computing platform that offers scalable and intertwined DevOps tools. The study presents a methodical approach to constructing a CI/ CD workflow using AWS native services similar as AWS Code Commit for source control, AWS Code Build for automated testing and compendium, AWS Code Deploy for operation deployment, and AWS Code Pipeline for unity. The proposed channel is tested using a sample pall-native web operation and crucial criteria similar as figure time, deployment frequency, and failure recovery time are estimated. The exploration highlights the benefits of integrating CI/ CD practices with pall structure, including bettered software trustability, reduced deployment crimes, and enhanced development dexterity. Likewise, the paper discusses security considerations, cost-effectiveness, and scalability aspects associated with using AWS- managed CI/ CD services. The findings give practical perceptivity for software masterminds, DevOps interpreters, and experimenters seeking to apply flexible and effective software delivery channels in pall surroundings.

Keywords: Continuous Integration, Continuous Deployment, CI/CD pipeline, AWS CI/CD tools, AWS Code Pipeline, AWS Code Build, AWS Code Deploy, AWS Code Commit

I. Introduction

Software development is a landscape that is constantly developing. Companies need to be able to distribute quality applications quickly and, more importantly, confidently. Continuous integration and continuous deployment or CI/CD, this process plays an integral role. The CI/CD allows teams to automate and efficiently apply at speeds by automatic and efficient to the distribution of software. As long as the code is regularly integrated, developers can run automatic tests; deploy new codes with low manual testing, and quickly discovering and fixing problems quickly. Mistakes and problems can still arise. Developers get to save time on each release. Continuous Integration and Continuous Deployment reduced the number of cycles to deploy the software source code. The CI/CD allows teams to get into the flow more rapidly, which means providing value to users in weeks or days instead of months.

The cloud in AWS has a wide integrated platform of CI/CD tools and source code management. AWS Code Pipe Line, AWS Codebuild, AWS Codecommit, AWS codedeploy are examples of services that enable users to reuse the code with low overhead, management and control and also provide demonstrations with their software operates. AWS software releases provide a unique platform of devices for safe, flexible and scalable growth and signal devices to automate each phase of the release.

II. Literature Review

The field of software development has seen a major shift from traditional monolithic systems and infrequent release cycles to more dynamic, agile-driven approaches. With the rise of microservices and the demand for faster, more reliable software delivery, practices like Continuous Integration (CI) and Continuous Delivery (CD) have become essential components of modern development workflows. CI emphasizes the regular merging of code into a shared repository, paired with automated build and test processes to identify issues early. CD builds on this by ensuring that the software is always in a production-ready state and can be released to different environments through automated mechanisms.

By embracing CI/CD, organizations are able to accelerate release cycles, enhance software reliability, reduce the likelihood of deployment failures, and foster closer collaboration between development and operations teams (as highlighted by researchers such as Humble & Farley, and Forsgren et al.). These improvements are particularly important in agile and DevOps-driven environments where speed and stability are equally critical.

Cloud computing has played a pivotal role in advancing CI/CD capabilities. Among the leading cloud providers, **Amazon Web Services (AWS)** stands out with its extensive ecosystem of managed services tailored for automation, scalability, and continuous delivery. AWS enables organizations to construct end-to-end CI/CD pipelines using services like Code Commit, Code Build, Code Deploy, and Code Pipeline. These tools support automated testing, deployment, and infrastructure management, making AWS an attractive platform for implementing DevOps practices.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Special Issue | Volume XIV, Issue XIII, October 2025

III. Research Methodology

What is Continuous Integration and Continuous Delivery/Deployment?

Continuous Integration (CI) and Continuous Delivery/Deployment (CD) are software development practices that aim to ameliorate the process of delivering software by automating and streamlining the integration and deployment phases.

Continuous Integration (CI)

Continuous Integration is a development practice where inventors constantly integrate their law changes into a participated depository, generally several times a day. The crucial factors of CI include, Automated Testing Each integration is automatically tested to descry crimes snappily. This helps insure that new law changes don't break being functionality. Figure robotization the law is automatically erected and packaged, allowing for immediate feedback on the integration process. Version Control CI relies on interpretation control systems (like Git) to manage law changes and track variations over time. The main pretensions of CI are to reduce integration problems, ameliorate software quality, and dock the time it takes to deliver new features.

Continuous Delivery (CD)

Continuous Delivery is an extension of CI that ensures that the software can be reliably released at any time. It involves Automated Deployment The software is automatically prepared for release to product after passing all tests. This includes packaging the operation and preparing the terrain. Carrying surroundings, the operation is stationed to carrying surroundings that nearly act product, allowing for farther testing and confirmation. Release Readiness The software is always in a deployable state, meaning that new features can be released to druggies at any time with minimum trouble.

Continuous Deployment

Continuous Deployment is a farther step beyond Continuous Delivery. In this practice, every change that passes automated tests is automatically stationed to product without homemade intervention. This allows for rapid-fire delivery of features and fixes to druggies.

IV. Benefits of Continuous Delivery:

Faster Time to Market: CD allows teams to release new features and updates more frequently, reducing the time it takes to deliver value to customers. Improved Software Quality: Automated testing and deployment processes help catch bugs and issues early in the development cycle, leading to higher quality software.

Reduced Deployment Risk: By deploying smaller, incremental changes rather than large releases, the risk

Associated with employments is minimized

This allows for easier identification and resolution of problems when they occur.

Enhanced Customer Feedback: Frequent releases allow for quicker feedback from users, enabling teams to iterate on features and make improvements based on real user experiences.

Greater Flexibility and Adaptability: CD enables teams to respond quickly to changing market demands or customer needs, allowing for more agile development practices.

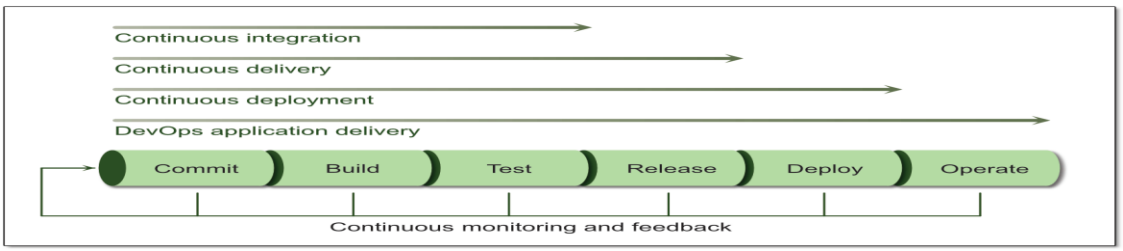
Consistent Deployment Process: Automation of the deployment process ensures that deployments are consistent and repeatable, reducing the chances of human error.

Improved Developer Productivity: With automated testing and deployment, developers can focus more on writing code and less on manual processes, leading to increased productivity.

Better Resource Utilization: Continuous Delivery can lead to more efficient use of resources, as teams can allocate time and effort to developing new features rather than fixing deployment issues.

Enhanced Visibility and Transparency: CD practices often include monitoring and reporting tools that provide insights into the deployment process, making it easier for teams to track progress and identify bottlenecks.

V. A Pathway to Continuous Integration/Continuous Delivery



ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Special Issue | Volume XIV, Issue XIII, October 2025

The image illustrates the concepts of DevOps and the application delivery pipeline, specifically highlighting the progression from Continuous Integration (CI) to Continuous Delivery (CD) and Continuous Deployment, culminating in the full DevOps application delivery process.

The Application Delivery Pipeline (Bottom Row):

This pipeline outlines the series of steps software undergoes from the point of initial development to its deployment and operation in a live environment:

- **Commit**: Developers make changes to the source code and submit (or "commit") them to a version control system such as Git. This action often initiates the automated processes in the pipeline.
- **Build**: The source code is compiled and packaged into an executable file or deployable component. This step ensures that all code syntax is valid and required dependencies are properly included.
- **Test**: The built application is subjected to various types of testing such as unit, integration, system, and sometimes performance or security tests. The objective here is to detect and resolve issues as early as possible in the lifecycle.
- **Release**: A stable version of the software is finalized and marked for deployment. At this point, the application is not necessarily live but is ready for deployment to production-like environments.
- **Deploy**: The application is moved to a specific environment such as staging, pre-production, or live production. Configuration settings are applied, and the software is made available within that environment.
- **Operate**: In this final stage, the software is live and serving end-users. Operational tasks include monitoring system health, managing performance, handling incidents, and ensuring continued reliability.

Continuous Monitoring and Feedback (Bottom Arrow)

A key feature illustrated is the ongoing cycle of monitoring and feedback that spans the entire application delivery pipeline. Throughout each stage from code commit to operation various aspects such as application performance, system issues, and user activity are constantly observed. The insights gained are continuously shared with both development and operations teams to help refine the application and enhance the overall delivery process.

The Evolution of Delivery Practices (Top Rows)

The visual representation also outlines how modern software delivery practices have evolved, each progressively covering more stages of the pipeline:

- **Continuous Integration (CI)**: Focused on the initial stages Commit and Build CI encourages developers to frequently merge their changes into a shared repository. Each integration triggers automated builds and tests, helping catch issues early. The CI phase often slightly overlaps with the beginning of the Test phase.
- **Continuous Delivery (CD)**: Building on CI, this approach ensures that the application is always in a deployable state once testing is complete. It spans the stages of Commit, Build, Test, and Release. Although deployment isn't automatic, the software is always ready to be deployed with manual approval.
- **Continuous Deployment**: This method takes CD further by automating the deployment step. If all tests and validations pass, the application is automatically released into the production environment. It covers nearly the entire pipeline from Commit through to Deploy and sometimes up to the Operate phase.
- **DevOps Application Delivery**: This represents a holistic, end-to-end approach that integrates development and operations across all stages from initial code commits to production operation. Continuous monitoring and feedback loops ensure that the process is continually refined and optimized.

Understanding the Difference: Release vs. Deployment

The diagram also emphasizes a critical distinction between two closely related concepts:

- **Release**: This refers to the point at which a version of the software is finalized and made ready for deployment. It involves packaging and tagging the build but does not necessarily mean the application is live.
- **Deployment**: This is the actual process of installing the released version into a specific environment such as staging or production making it accessible and operational for end-users.

VI. AWS Services for CI/CD: A Comprehensive Overview

AWS provides a comprehensive and tightly integrated suite of managed services that facilitate the creation of robust, scalable, and automated CI/CD pipelines. These services reduce operational overhead by abstracting away infrastructure management, allowing teams to focus on pipeline logic and application development.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Special Issue | Volume XIV, Issue XIII, October 2025

1. Source Control

AWS CodeCommit is a totally managed, secure, and highly scalable source control service that supports Git repositories. It provides a centralized platform to store application code, libraries, and related assets, with data encrypted both in transit and at rest. CodeCommit integrates directly with AWS Identity and Access Management (IAM), enabling fine-grained access controls to ensure that only authorized users can access or modify repositories. This makes it an ideal choice for building and managing CI/CD pipelines within the AWS ecosystem.

2. Build Automation

AWS CodeBuild: A fully managed continuous integration service, CodeBuild automates the process of compiling source code, running tests, and producing deployable software packages. Its serverless architecture means users don't need to provision, manage, or scale any build servers, paying only for the compute time consumed. CodeBuild supports various programming languages and runtimes and allows for custom build environments using Docker images. Build processes are defined via a buildspec.yml file, enabling detailed control over compilation, testing, and artifact generation (AWS, 2023b). CodeBuild is central to the "Build" and initial "Test" stages of a CI/CD pipeline, transforming raw code into validated artifacts.

3. Deployment Automation

AWS CodeDeploy: This is a fully managed deployment service that automates software deployments to a wide array of compute services, aiming to minimize downtime during updates. CodeDeploy supports diverse deployment targets including Amazon EC2 instances, AWS Lambda functions, AWS Fargate/ECS containers, and even on-premises servers. It offers flexible deployment strategies such as in-place updates, and more importantly, zero-downtime strategies like Blue/Green and Canary deployments. These advanced strategies allow for gradual traffic shifting and easy rollbacks, significantly mitigating deployment risks. Deployment actions are defined through an appspec.yml file, specifying lifecycle hooks and ensuring precise control over the deployment process (AWS, 2023c). CodeDeploy is the primary mechanism for the "Deploy" stage, translating validated artifacts into running applications.

4. Pipeline Orchestration

At the heart of the AWS continuous delivery ecosystem lays AWS CodePipeline, a fully managed service engineered to automate release workflows from end to end. This service empowers users to construct multi-stage pipelines, each typically encompassing distinct phase such as source retrieval, code building, testing, deployment, and optional approval gates. Within these stages, one or more specific actions can be defined. CodePipeline intelligently monitors for changes in the source repository (e.g., a new commit) and, upon detection, automatically initiates the predefined sequence of actions, providing a transparent and visual depiction of the pipeline's progression. Its profound integration with other core AWS development services like AWS CodeCommit (for source control), AWS CodeBuild (for building and testing), and AWS CodeDeploy (for deployment) ensures a remarkably fluid and integrated experience. Effectively, CodePipeline serves as the central nervous system, meticulously connecting and coordinating individual CI/CD components into a unified, automated, and highly efficient software delivery workflow.

5. Infrastructure as Code (IaC)

AWS CloudFormation: CloudFormation is a pivotal service for practicing IaC on AWS, enabling users to model and provision their entire AWS infrastructure using declarative templates (JSON or YAML). This ensures that environments (development, testing, production) are consistently provisioned and reproducible, eliminating configuration drift and "works on my machine" issues. CloudFormation manages the provisioning and updating of resources in a controlled and predictable manner, supporting change sets for previewing impacts before execution (AWS, 2023e). Its integration into CI/CD pipelines means infrastructure can be versioned and deployed alongside application code, treating infrastructure changes with the same rigor as application code changes.

AWS Cloud Development Kit (CDK): The AWS CDK is an open-source framework that allows developers to define cloud infrastructure using familiar programming languages (e.g., Python, TypeScript, Java). It compiles these definitions into CloudFormation templates, offering a higher level of abstraction and leveraging object-oriented programming benefits like reusability and modularity. CDK enhances developer productivity and streamlines the definition of complex AWS architectures within a CI/CD context (AWS, 2023f).

6. Container and Serverless Technologies (Integrated Targets)

Amazon Elastic Container Registry (ECR): ECR is a fully managed Docker container registry for securely storing, managing, and deploying container images. In a CI/CD pipeline, CodeBuild often pushes newly built Docker images to ECR, which then serves as the artifact repository for container orchestration services (AWS, 2023g).

Amazon Elastic Container Service (ECS) / Amazon Elastic Kubernetes Service (EKS): These services provide robust platforms for running containerized applications. ECS is AWS's native container orchestration service, EKS offers a managed Kubernetes experience, and Fargate provides serverless compute for both ECS and EKS. CI/CD pipelines frequently target these

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Special Issue | Volume XIV, Issue XIII, October 2025

services for deploying microservices and other containerized workloads, leveraging CodeDeploy or custom pipeline actions for orchestrating container updates (AWS, 2023h; AWS, 2023i; AWS, 2023j).

AWS Lambda: As a serverless compute service, Lambda allows running code without provisioning or managing servers, making it ideal for event-driven architectures. CI/CD pipelines can directly deploy Lambda functions and associated serverless applications (often via AWS Serverless Application Model - SAM) (AWS, 2023k). Lambda also serves as a powerful tool for custom actions within CodePipeline, enabling unique integrations or advanced pipeline logic.

7. Monitoring, Logging, and Observability

Amazon CloudWatch: CloudWatch is critical for monitoring pipeline execution (e.g., build times, deployment failures) and, more importantly, for monitoring the health, performance, and operational metrics of deployed applications. CloudWatch alarms can be configured to trigger automated rollbacks in CodeDeploy or notify teams of issues, ensuring rapid response to production incidents (AWS, 2023l).

AWS CloudTrail: CloudTrail provides an audit trail of all API calls made to your AWS account, offering insights into who did what, when, and from where. This is crucial for security, compliance, and troubleshooting pipeline failures or unauthorized changes within the CI/CD environment (AWS, 2023m).

AWS X-Ray: For distributed applications, X-Ray helps analyze and debug by providing an end-to-end view of requests as they flow through various services. While not a direct pipeline component, its insights into application behavior are invaluable for refining and improving the deployed software, feeding back into the continuous improvement loop of DevOps (AWS, 2023n).

VII. Conclusion

Continuous Integration and Continuous Delivery are cornerstones of modern software development, enabling organizations to deliver high-quality software rapidly and reliably. The advent of cloud computing, particularly the comprehensive and managed services offered by Amazon Web Services, has significantly democratized and scaled the implementation of these practices. AWS provides a powerful ecosystem of specialized tools like CodeCommit, CodeBuild, CodeDeploy, and CodePipeline, complemented by IaC services (CloudFormation, CDK) and a robust set of compute, storage, and monitoring solutions.

While the benefits including accelerated release cycles, improved reliability, cost efficiency, and enhanced developer productivity are well-documented, organizations must also navigate challenges such as initial setup complexity, legacy system migration, and the critical need for a strong organizational culture shift. As the cloud landscape continues to evolve, future research should delve into advanced optimization techniques, deeper security integrations, AI/ML-driven automation, and the long-term impacts of sophisticated CI/CD pipelines on business outcomes, ensuring that the promise of rapid and reliable software delivery continues to be met.

References

1. https://docs.aws.amazon.com/whitepapers/latest/practicing-continuous-integration-continuous-delivery/testing-stages-in-continuous-integration-and-continuous-delivery.html
2. https://www.researchgate.net/publication/330988585_Practicing_Continuous_Integration_and_Continuous_Delivery_on_AWS_Accelerating_Software_Delivery_with_DevOps
3. AWS CodePipeline Documentation. Retrieved from https://docs.aws.amazon.com/codepipeline
4. https://www.sumologic.com/glossary/aws-codepipeline
5. https://cloudvisor.co/aws-guides/aws-codepipeline/
6. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.
7. Kim, G., Humble, J., Debois, P., & Willis, J. (2014). The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution Press.
8. Nalawade, C., Mahajani, D., & Shinde, C. (2023). Implementing Continuous Integration and Deployment (CI/CD) for Machine Learning Models on AWS. International Journal of Global Innovations and Solutions (IJGIS), 2(2), 52-60.