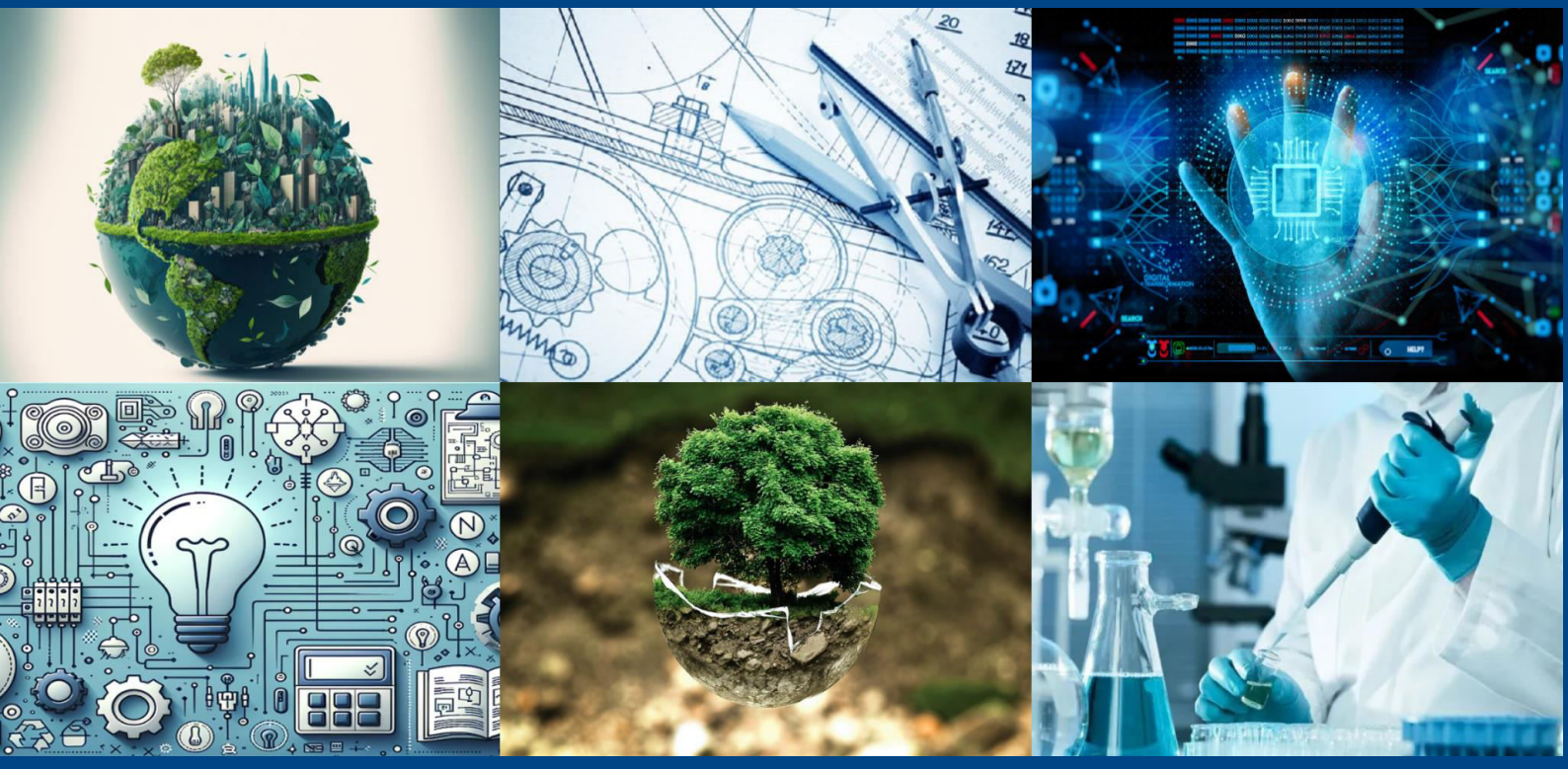
ISSN: 2582-7219

International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.864

Volume 9, Issue 7, July 2026

© 2026 IJMRSET | Volume 9, Issue 7, July 2026| DOI:10.15680/IJMRSET.2026.0907013

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Fully Automated CI/CD Pipeline using Jenkins, Maven, SonarQube, Nexus and Slack Integration

Aditya Pansare, Aditi Chavan, Peeyush Bagewadi, Swarali Darekar

School of Computer Science, Engineering and Applications (SCSEA), D Y Patil International University, Akurdi, Pune, India

ABSTRACT: Modern software teams need to build, verify, and release code changes rapidly and reliably. This paper presents a fully automated Continuous Integration and Continuous Delivery (CI/CD) pipeline built around Jenkins as the orchestration engine. The pipeline automatically pulls source code from a GitHub repository, compiles and packages it using Maven, executes automated unit tests, performs static code-style checks with Checkstyle, runs static analysis through SonarQube backed by a mandatory Quality Gate, uploads the versioned build artifact to a Nexus Repository, and finally notifies the team through Slack. The system was validated across ten pipeline runs on a Jenkins server hosted on an AWS EC2 instance, including successful builds, a deliberately observed Quality-Gate failure, and a real executor-scheduling fault caused by low disk space, demonstrating that the pipeline is functional, resilient, and diagnosable under real infrastructure constraints.

KEYWORDS: CI/CD, Jenkins, DevOps, SonarQube, Nexus Repository, Continuous Integration, Continuous Deployment, Quality Gate, Slack Notification, Automated Testing.

I. INTRODUCTION

In traditional software development, building, testing, and deploying an application manually is slow, repetitive, and error-prone — a single missed step can introduce bugs or inconsistent artifacts into production. DevOps practices solve this problem through automation, and a CI/CD pipeline is the centerpiece of that automation. This work implements an end-to-end pipeline that treats every code push as an event capable of triggering a complete, repeatable workflow: fetch, build, test, analyze, package, publish, and notify.

The pipeline is written as Jenkins Declarative Pipeline code (a Jenkinsfile), meaning the entire build process is version-controlled alongside the application code. Quality is enforced automatically: Checkstyle flags style violations, SonarQube performs static analysis for bugs, vulnerabilities and code smells, and a Quality Gate blocks the pipeline from proceeding if the code does not meet the configured threshold. Once code passes all checks, Jenkins uploads the compiled WAR artifact to Nexus and posts a build-status message to Slack, giving the team immediate visibility without checking the dashboard manually.

The primary objective of this work is to design and implement a Jenkins-based CI/CD pipeline that fully automates code checkout, build, testing, static analysis, quality gating, artifact publishing, and team notification, removing the need for manual intervention at any stage. Secondary objectives include enforcing coding standards, providing centralized versioned artifact management, delivering real-time build notifications, and ensuring the CI/CD infrastructure itself is monitored and diagnosable.

II. RELATED WORK

Jenkins is widely used as an open-source automation server for building CI/CD pipelines through its Pipeline-as-Code model [1]. Static analysis platforms such as SonarQube are commonly integrated into such pipelines to enforce code quality through configurable Quality Gates [2]. Artifact repository managers, such as Sonatype Nexus, are used to version and centrally store build outputs for traceability across releases [3]. Build automation for Java-based applications is typically handled through Apache Maven [4], while team-communication integrations such as Slack are used to close the feedback loop by notifying developers of build outcomes in real time [5]. This project combines all of these established

© 2026 IJMRSET | Volume 9, Issue 7, July 2026| DOI:10.15680/IJMRSET.2026.0907013

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

tools into a single automated, observable pipeline and additionally documents real infrastructure-level failure modes encountered during implementation.

III. PROPOSED METHODOLOGY

The proposed system automates eight sequential stages, each corresponding to a Jenkins pipeline stage: (1) Fetch Code, (2) Build, (3) Unit Test, (4) Checkstyle Analysis, (5) SonarQube Code Analysis, (6) Quality Gate, (7) Publish to Nexus, and (8) Slack Notification. Fig. 1 illustrates the complete control flow, including the conditional branch at the Quality Gate stage that determines whether the build proceeds to artifact publishing or is halted with a failure notification.

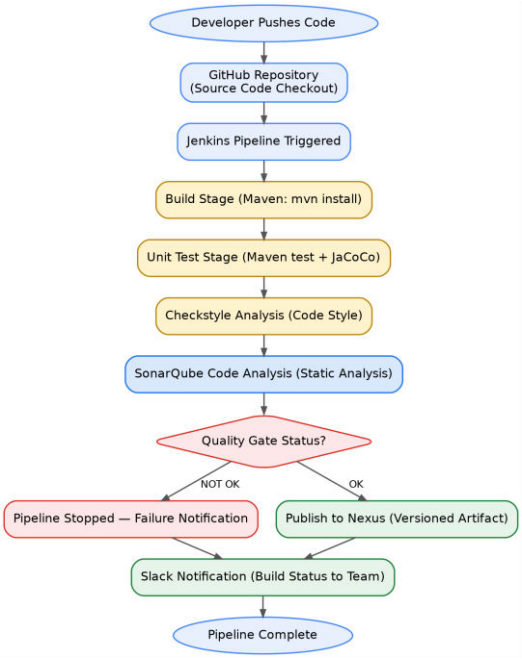


Fig. 1. Flowchart of the fully automated CI/CD pipeline.

The "Fetch Code" stage clones the latest commit from the GitHub repository. The "Build" stage runs "mvn install -DskipTests", packaging the application into a deployable WAR file while JaCoCo attaches as a Java agent to begin coverage collection. The "Unit Test" stage executes the full automated test suite via "mvn test", and results are captured in Surefire reports. The "Checkstyle Analysis" stage scans the codebase against the Sun coding-convention ruleset without failing the build, purely to track style health. The "SonarQube Code Analysis" stage runs the SonarScanner CLI, combining source scanning, the Checkstyle report, Surefire results and JaCoCo coverage into a single analysis report uploaded to the SonarQube server.

The "Quality Gate" stage pauses (with a one-hour timeout) and polls SonarQube for the analysis outcome; only an "OK" status allows the pipeline to continue. The "Publish to Nexus" stage uploads the WAR artifact to a dedicated repository, automatically versioned by build number and timestamp. Finally, the "Slack Notification" stage sends a color-coded build-status message to the team's Slack channel.

TABLE I. TECHNOLOGIES USED

Category	Tool / Technology
Cloud Platform	Amazon Web Services (EC2)
CI/CD Orchestration	Jenkins 2.568.1 (Declarative Pipeline)

© 2026 IJMRSET | Volume 9, Issue 7, July 2026| DOI:10.15680/IJMRSET.2026.0907013

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Source Control	Git & GitHub
Build Tool	Apache Maven
Language / Runtime	Java (OpenJDK 21), JSP
Code Coverage	JaCoCo 0.8.9
Style Analysis	Maven Checkstyle Plugin
Static Code Analysis	SonarQube (Community 26.4)
Artifact Repository	Sonatype Nexus (vprepo)
Notifications	Slack (Jenkins Slack Plugin)

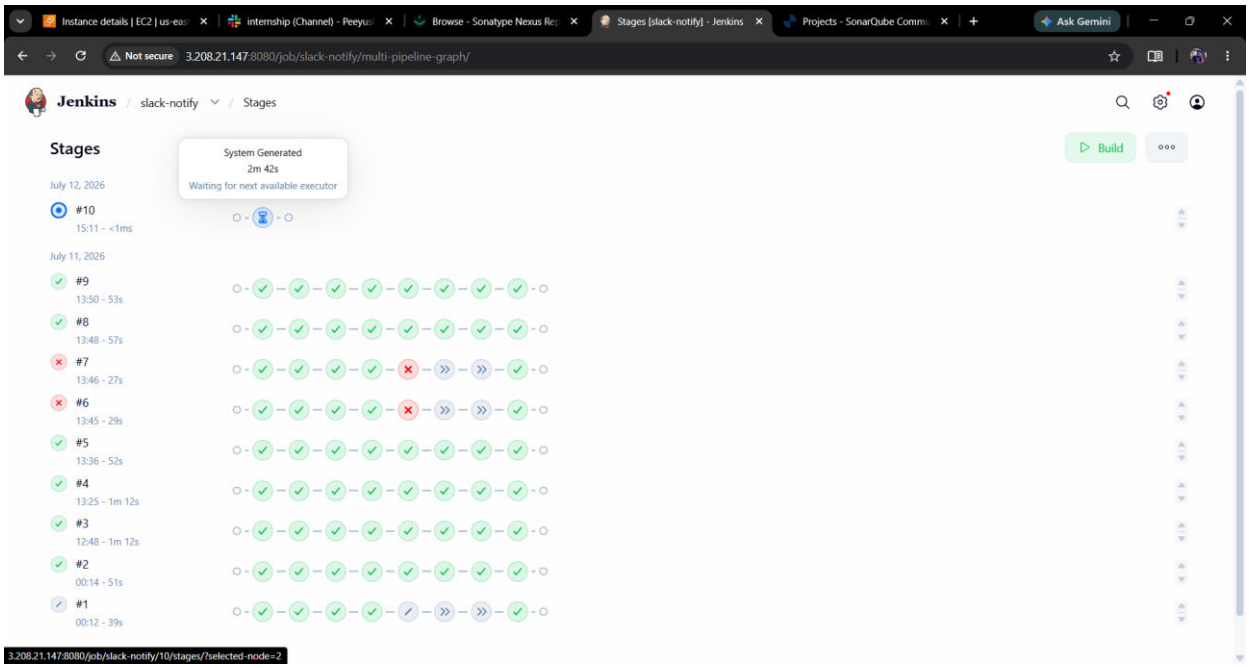


Fig. 2. Jenkins multi-pipeline stage view showing consecutive automated build runs (#1–#10), each progressing through Fetch Code, Build, Unit Test, Checkstyle, SonarQube, Quality Gate, Nexus Publish and Slack Notify.

IV. IMPLEMENTATION AND RESULTS

The pipeline was executed across ten build runs (Fig. 2). Builds #1–#5, #8 and #9 completed all stages successfully. Builds #6 and #7 show a deliberately observed failure at the SonarQube analysis stage, with subsequent stages automatically skipped, confirming that the pipeline correctly halts propagation when an earlier stage fails. Build #10 illustrates a scheduling delay, waiting on the "System Generated" stage for an available executor before starting.

Representative excerpts captured directly from the Jenkins console log of a completed run are shown below, confirming each stage executed and passed in sequence.

```
[Pipeline] { (Build)  +mvn install -DskipTests
[INFO] BUILD SUCCESS   Total time: 8.937 s
[Pipeline] { (UNIT TEST)  +mvn test
Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[Pipeline] { (Checkstyle Analysis)
641 errors reported by Checkstyle 9.3 (sun_checks.xml)
[Pipeline] { (Sonar Code analysis)
```

© 2026 IJMRSET | Volume 9, Issue 7, July 2026| DOI:10.15680/IJMRSET.2026.0907013

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

```
ANALYSIS SUCCESSFUL — EXECUTION SUCCESS (28.401 s)
[Pipeline] { (Quality Gate)
SonarQube task status: SUCCESS — Quality gate: OK
[Pipeline] { (Publish to Nexus)
Uploaded vprofile-v2.war (83 MB at 32 MB/s)
[Pipeline] { (Post Actions) — Slack: #internship, color: good
Finished: SUCCESS
```

These results confirm end-to-end automation: nine of nine unit tests passed, the Quality Gate returned an "OK" verdict, an 83 MB artifact was uploaded to Nexus at 32 MB/s, and a Slack notification was delivered without any manual step.

V. INFRASTRUCTURE MONITORING AND FAULT DIAGNOSIS

Because the entire pipeline depends on an available Jenkins executor, node health — free disk space, swap and temp space — is actively monitored. During testing, build #10 stalled in a "Waiting for next available executor" state. Investigating the Jenkins Nodes dashboard (Fig. 3) revealed that the Built-In Node had gone offline, flagged by Jenkins' automatic Free Disk Space and Free Swap Space monitors due to critically low available disk space on the underlying EC2 instance. Resolving it — by freeing disk space and bringing the node back online — restored normal scheduling, confirming this as a genuine, reproducible infrastructure failure mode rather than a pipeline logic defect.

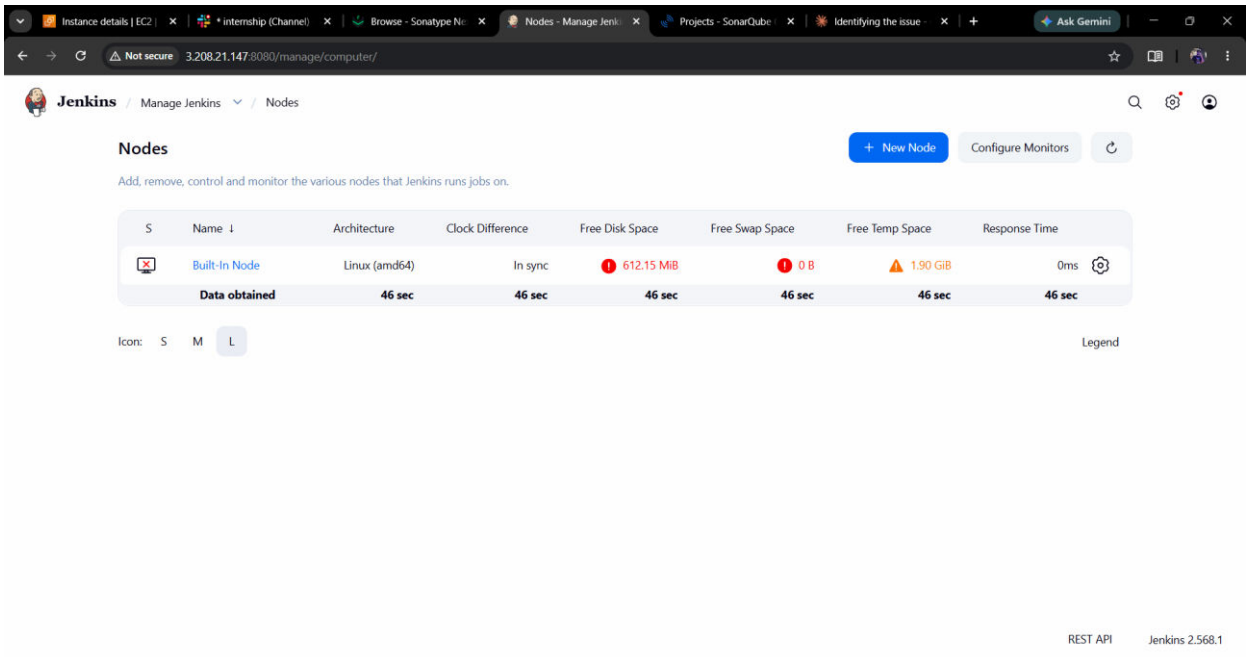


Fig. 3. Jenkins "Manage Nodes" dashboard showing the Built-In Node offline, with critical alerts on Free Disk Space (612.15 MiB) and Free Swap Space (0 B) — the root cause of the executor-scheduling delay.

VI. ADVANTAGES AND LIMITATIONS

The proposed pipeline offers faster feedback, since developers learn within minutes whether a commit builds, passes tests and meets quality standards. Because the pipeline is defined as code, every build follows an identical sequence, eliminating environment inconsistencies. The Quality Gate prevents substandard code from reaching the artifact repository, Nexus provides traceable versioned storage, and Slack integration ensures build failures are visible immediately rather than sitting unnoticed.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 9.864| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

The main limitation is infrastructure dependency: the pipeline depends on the health of the underlying Jenkins server, and resource exhaustion — as observed with low disk space — can stall every subsequent build until resolved. A single-executor configuration also creates a scheduling bottleneck under concurrent load, and initial setup of Jenkins, SonarQube, Nexus and Slack integration requires careful credential and plugin configuration.

VII. FUTURE SCOPE

Future extensions include a containerized deployment stage using Docker and Kubernetes for full continuous deployment; additional Jenkins agents or dynamic cloud agents to remove the single-executor bottleneck; automated rollback to the last known-good Nexus artifact on failed health checks; SAST/DAST and dependency-vulnerability scanning as additional gated stages; infrastructure-as-code management of the Jenkins server using Terraform or Ansible; and Slack-based manual approval steps for production promotion.

VIII. CONCLUSION

This work demonstrates a fully automated CI/CD pipeline that takes a code commit through build, automated testing, static code-quality analysis, a Quality Gate, artifact publishing and real-time team notification without manual intervention. The pipeline was validated across ten build runs, including successful executions, a deliberate Quality-Gate failure, and a real infrastructure fault, all of which were observed, diagnosed and resolved. By combining Jenkins, GitHub, Maven, Checkstyle, SonarQube, Nexus and Slack into a single automated workflow, the system reduces manual effort, enforces consistent quality standards, and gives the development team immediate visibility into build health, forming a solid foundation for a production-grade DevOps workflow.

REFERENCES

[1] Jenkins Project, "Pipeline," Jenkins Documentation. [Online]. Available: https://www.jenkins.io/doc/book/pipeline/
[2] SonarSource, "SonarQube Documentation." [Online]. Available: https://docs.sonarsource.com/sonarqube/
[3] Sonatype Inc., "Nexus Repository 3 Help." [Online]. Available: https://help.sonatype.com/repomanager3
[4] Apache Software Foundation, "Maven Getting Started Guide." [Online]. Available: https://maven.apache.org/guides/
[5] Jenkins Project, "Slack Notification Plugin." [Online]. Available: https://plugins.jenkins.io/slack/
[6] JaCoCo, "Java Code Coverage Library." [Online]. Available: https://www.jacoco.org/jacoco/

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com