iJRASET
International Journal For Research in Applied Science and Engineering Technology



# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 13 **Issue:** VII **Month of publication:** July 2025

**DOI:** https://doi.org/10.22214/ijraset.2025.73434

www.ijraset.com

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

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue VII July 2025- Available at www.ijraset.com*

# Blockchain-Enabled DevSecOps Pipeline for Automated Compliance and Security Audits

Nallam Sri Venkata Kalyan
*Jawaharlal Nehru Technological University, Kakinada*

***Abstract:*** ***DevOps speeds up the delivery of software but is risky with centralized logging. Blockchain integration into DevSecOps provides decentralized, tamper-evident audit trails with smart contracts. This provides secure, traceable CI/CD pipelines using tools such as Jenkins, Docker, Truffle, and MetaMask for real-time, verifiable logging and compliance automation.***
***Keywords: Blockchain, DevOps, CI/CD, Smart Contracts, Jenkins, Secure Pipelines, Audit Trails***

## I. INTRODUCTION

Today's software delivery pipelines depend on automation for fast, repeatable, and reliable builds, tests, and releases. Yet, their automated processes are still at risk of security compromise, unauthorized access, and inadequate transparency in audit records. As software systems increase in sophistication and release frequency, immutable, verifiable logging and secure validation of the process become essential. The integration of blockchain technology with DevSecOps practices presents a hopeful path to address these issues by infusing transparency, traceability, and tamper-evident auditing into CI/CD pipelines.

### *A. Introduction*

These days in software engineering, CI/CD (which stands for “Continuous Integration” and “Continuous Deployment”) pipelines are cornerstone pieces of automation for building, testing, and deploying our software. However, these automated processes cause several concerns such as security risks, insider threats, and non-auditable logging. As a distributed ledger with smart contract features, blockchain technology has the potential to add new dimensions of security, transparency, and auditability to the DevOps process. In this paper, we propose a blockchain-augmented DevSecOps pipeline, which uses Ethereum smart contracts to log each of the important CI/CD actions with the guarantee of non-repudiation. Leveraging open-source technologies like Jenkins for automation, Docker for containerization, and Truffle w/Ganache for blockchain interaction, we provide a single layer to govern traditional software delivery against decentralized security models.

As a result, all builds, tests, and deployments performed can be traced and verified with cryptographic proofs that logs have not been tampered. Further, through rolling a MetaMask compatible frontend, developers and auditors can now interface with the blockchain on-the-fly to inspecting logs and how security compliance is being upheld.

### *B. Existing System*

In traditional DevOps systems, logging and security validation are typically centralized and rely heavily on backend tools or scripts. These systems do not provide cryptographic assurance of logs nor do they offer tamper-resistant records. Logs can be deleted or modified if the central server is compromised. Also, role-based access control is often loosely enforced in CI/CD tools, leaving room for privilege abuse. Without distributed verification, there is minimal trust in recorded software activities.

### *C. Proposed System*

This paper proposes a Blockchain-Enabled DevSecOps Pipeline where all critical pipeline events—like build triggers, test results, and deployment approvals—are logged onto the Ethereum blockchain. Smart contracts are designed to record and expose this information securely and transparently. Jenkins acts as the automation orchestrator, Docker is used to deploy the application in isolated containers, and the blockchain ensures that event logs are immutable and auditable. A web interface built with HTML and Web3.js allows users to fetch logs via MetaMask for decentralized access.

This system ensures tamper-proof audit trails, real-time visibility, and enhanced compliance. It supports both manual and automated logging via CLI (logEvent.mjs) and frontend UI interaction.

©IJRASET: All Rights are Reserved 

### D. Objectives of the Paper

This paper has one main goal, that is to inject blockchain technology into current DevOps pipelines to implement decentralized logging. This integration is expected to guarantee immutability, maintain transparency and transparency of the software delivery process. One of the main objectives is to prevent unauthorized tampering or deletion of CI/CD (Continuous Integration / Continuous Delivery) logs. To contribute to this a smart contract will be implemented to record the significant events, builds, tests and deployment steps. We will also provide a frontend using MetaMask so that users can securely check logs live. Finally, the system will be assessed on performance, auditability, and cost of integration with standard DevOps toolchains.

### E. Organization of Paper

Section 1: Introduction and proposed solution
Section 2: Related work and research gaps
Section 3: System design and architecture
Section 4: Implementation details and tools
Section 5: Testing and validation strategy
Section 6: Experimental results and screenshots
Section 7: Conclusion and future enhancements

## II. RELATED WORK

The increased complexity of CI/CD pipelines has amplified the demand for secure, auditable, and tamper-proof logging solutions. Recent studies emphasize how blockchain integrated in DevOps increases trust, transparency, and traceability of software release.

### A. Blockchain for CI/CD Security

Legacy CI/CD platforms use centralized logging, thereby exposing them to insider attacks and tampering with logs. Scholars such as Saleh et al. (2024) and Dhawde (2024) suggested Ethereum-based logging systems with smart contracts for securing and auditing pipeline events. Such methods provide immutability and provide decentralized audit trails.

### B. Logging and Metrics with Smart Contracts

Smart contracts enable safe logging of CI/CD events like build success or failure and test outcomes. Ethereum is utilized by Dhawde's model for keeping logs verifiably, preventing insider attacks and enhancing post-incident analysis in cloud-native DevOps environments.

### C. Supply Chain Security with Blockchain

Karanam (2024) focused on blockchain-enabled software attestation to mitigate threats such as dependency poisoning and privilege escalation. Their approach encourages Zero Trust Architecture and RBAC to ensure CI/CD integrity and traceability.

### D. Blockchain in Cyber-Physical Systems (CPS)

Khalil et al. (2021) investigated the use of blockchain for real-time security in CPS and IoT systems. Utilizing permissioned ledgers such as Hyperledger, they imposed secure authentication, access control, and rollback protocols—principles relevant to secure DevOps pipelines.

### E. Reviews on Blockchain for Cybersecurity

Literature reviews observe blockchain's advantages in attack resistance and identity management. Despite being effective to strengthen security, scalability, energy consumption, and regulatory issues are areas of concern.

### F. Threat Models and Defense

Literature reviews observe blockchain's advantages in attack resistance and identity management. Despite being effective to strengthen security, scalability, energy consumption, and regulatory issues are areas of concern.

### G. Software Supply Chain Auditing

Benedetti et al. (2022) presented Sunset, a blockchain-based software that rates third-party software dependencies according to their risk, enhancing accountability and vendor transparency in the supply chain.

*H. Blockchain Testing and Deployment Frameworks*

Tools such as NVAL smartly automate smart contract deployment and analysis. These tools are crucial to test blockchain systems incorporated in CI/CD pipelines so that trusted scaling and performance monitoring can be guaranteed.

*I. Developer Challenges*

Bosu et al. (2019) reported developer pain areas in blockchain adoption—like debugging tools and hard SDKs—requiring improved DevOps integration, particularly in security-critical workflows.

*J. IoT and Blockchain Security*

Salah and Khan (2017) suggested blockchain for IoT device authentication and the defense against real-time threats. Their research is in favor of employing smart contracts for verifiable identity and access control across distributed systems, pertinent to DevOps security too.

| Study (Year) | Area of Focus | Blockchain Use Case | Key Takeaway |
|---|---|---|---|
| Saleh et al. (2024) | CI/CD Pipeline Security | Smart contracts for code integrity | Traceable and secure deployments |
| Dhawde (2024) | DevOps Logs & Metrics | Ethereum-based log recording | Tamper-proof logs and better auditability |
| Karanam (2024) | DevSecOps Security Risks | Software attestation using blockchain | Secure supply chain and compliance |
| Khalil et al. (2021) | Cyber-Physical Systems (CPS) | Hyperledger for distributed security | Real-time monitoring and M2M auth |
| Benedetti et al. (2022) | Supply Chain Risk Evaluation | Risk scoring via blockchain | Dependency visibility and vendor trust |
| Tran et al. (2022) | Deployment Automation | Network auto-deploy & evaluation tools | Simplified setup of blockchain testbeds |
| Salah & Khan (2017) | IoT Security | Smart contract-based auth | Tamper-resistance in IoT comms |

TABLE 1: Comparative View of Selected Literature

## III. PROPOSED SYSTEM AND MODULES

With mounting the need for reliable and secure DevSecOps pipelines, our system design incorporates blockchain technology into CI/CD pipelines to verify immutable logging, transparent deployment, and decentralized validation of vital events. Taking cues from trust mechanisms employed in distributed systems, our design builds on smart contracts and Web3 connectivity to facilitate traceability and automation in software

*A. Modules Overview*

The system architecture is structured into separate functional modules:

Smart Contract Logger: Provides Solidity functions such as recordLog() and getLogsCount() to safely log CI/CD event logs onto the Ethereum blockchain.

CI/CD Automation: Utilizes Jenkins and Maven to automate build, test, and deployment processes of code.

Docker Deployment: Deploys applications into containers that maintain an environment of consistency and fast delivery.

Blockchain Interaction: A Node.js layer powered by Web3.js streams Jenkins logs onto the blockchain.

Blockchain Interaction: A Web3.js-powered Node.js layer streams logs from Jenkins to the blockchain.

Frontend dApp Interface: An interface based on HTML and JavaScript allows users to retrieve and authenticate on-chain logs using MetaMask.

Each module is structured to impose separation of concerns yet facilitate end-to-end trust in DevOps workflows.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

*B. Blockchain-Integrated CI/CD*

Traditional CI/CD environments typically do not have strong tamper-proofing, so they can be vulnerable to log tampering and insider attacks. Adding blockchain brings distributed consensus and immutability to the picture, making each build, test, and deployment operation provable. Previous research with Ethereum and Hyperledger has demonstrated the possibility of decentralized audit trails. Our solution builds on this with the addition of blockchain hooks integrated into Jenkins pipelines to have real-time, traceable logs for delivery phases.

*C. Secure Logging and Audit Trails*

Log integrity is essential for software security audits. In our system, logs generated during CI/CD executions are transmitted directly to a smart contract using Web3.js. Unlike traditional log servers, our design guarantees:

- Real-Time Logging: Logs are written to the blockchain during pipeline execution.
- Verifiability: All entries can be publicly audited using a frontend interface.
- Tamper Resistance: Logs are cryptographically secure and cannot be modified or deleted.

This mechanism offers a reliable audit trail for security reviews and post-deployment assessments.

*D. Smart Contract Design*

At the core is the DevOpsLogger.sol smart contract. It:

- Accepts log entries with metadata (timestamp, sender).
- Stores records immutably on-chain.
- Exposes APIs (getLog(), getLogsCount()) for retrieving data from CLI tools or web interfaces.

This replaces centralized logging APIs with decentralized alternatives, aligning with zero-trust and compliance-focused software lifecycles.

*E. Jenkins Pipeline with Blockchain Hooks*

We define a customized Jenkinsfile that:

- Clones source code from a Git repository.
- Copies blockchain-related files (e.g., ABI, event loggers).
- Builds and tests the application using Maven.
- Creates and deploys Docker containers.
- Executes a Node.js script (logEvent.mjs) to log pipeline data to the blockchain.

This tightly coupled integration facilitates continuous compliance and security visibility without altering the developer workflow.

*F. Visualization and Verification*

To enhance transparency, a frontend dApp was developed using Web3.js, HTML, and MetaMask. It enables:

- Retrieval of on-chain logs.
- Visualization of messages and timestamps.
- Manual log submission (e.g., for QA events or stakeholder updates).

Unlike traditional dashboards (e.g., ELK, Splunk), our frontend is serverless and directly queries the blockchain, eliminating backend dependencies and enhancing accessibility across distributed teams.

## IV. SYSTEM ARCHITECTURE AND DESIGN

*A. Overview of the Architecture*

The system integrates traditional DevOps tools (Jenkins, Maven, Docker) with Ethereum-based blockchain components (Ganache, Truffle, Smart Contracts) to create a secure and auditable CI/CD pipeline. This architecture ensures that every build, test, and deployment event is transparently logged and immutably recorded on the blockchain, offering traceability, accountability, and real-time verification. The system integrates traditional DevOps tools (Jenkins, Maven, Docker) with Ethereum-based blockchain components (Ganache, Truffle, Smart Contracts) to create a secure and auditable CI/CD pipeline. This architecture ensures that every build, test, and deployment event is transparently logged and immutably recorded on the blockchain, offering traceability, accountability, and real-time verification.

Key Components:

- Jenkins: Manages the CI/CD pipeline and triggers logging events.
- Docker: Builds and deploys the web application in a containerized environment.
- Truffle & Ganache: Handles compilation and local deployment of Ethereum smart contracts.
- Smart Contract (Solidity): Stores log entries immutably.
- Web3.js Frontend + MetaMask: User interface for viewing and verifying blockchain logs.
- logEvent.mjs: Node.js script used to send Jenkins log messages to the smart contract.
- verifyLogs.mjs: Node.js script used to query blockchain logs from the terminal.

### *B. High-Level System Workflow Diagram*

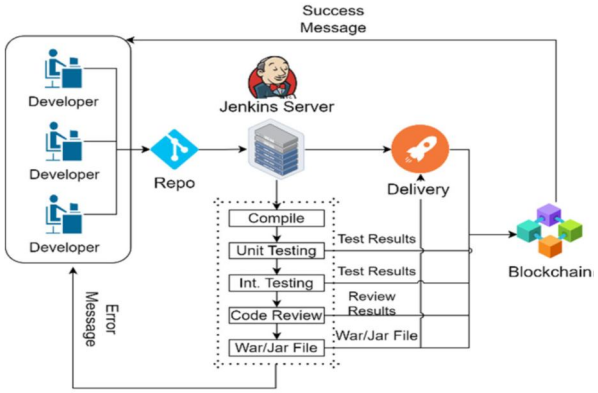



Fig 1: High-Level System Workflow Diagram

This flowchart visualizes the end-to-end path of a DevOps event—from source code commit to blockchain logging and real-time visualization.

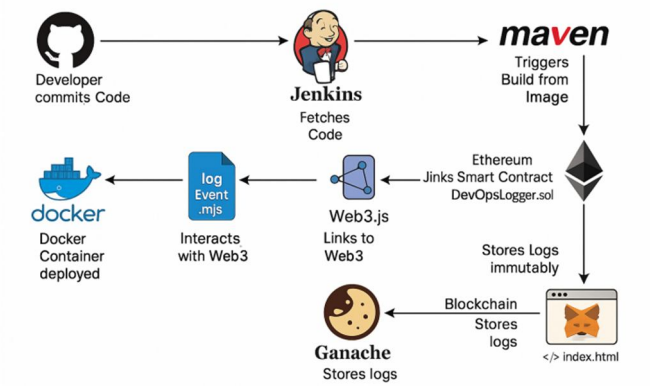



Fig 2: Blockchain-Enabled DevSecOps Pipeline Integration

### *C. Jenkins Pipeline Design*

The Jenkinsfile orchestrates all stages of the DevOps cycle:

| Stage | Purpose |
|---|---|
| Clone Code | Pull source code from GitHub |
| Copy Blockchain Files | Copy necessary .mjs, ABI, and contract JSON files into the workspace |
| Build with Maven | Compile the web application |
| Build Docker Image | Package the app into a Docker image |
| Run Docker Container | Deploy and expose the application |
| Log to Blockchain | Call the smart contract to store a log using logEvent.mjs |

TABLE 2: The final pipeline ensures both functional deployment and traceable logging on-chain.

*D. Smart Contract Design*

Our implementation uses Ganache to simulate a local Ethereum blockchain and Truffle for contract compilation and deployment. A Solidity contract, DevOpsLogger.sol, defines a struct to hold log messages and timestamps. Functions such as recordLog() and getLog() enable secure write/read access. Jenkins pipelines are defined using Jenkinsfile, specifying Maven build stages, Docker integration, and blockchain logging via logEvent.mjs. This script connects to the Ethereum node, fetches contract ABI and address, and logs build metadata. verifyLogs.mjs allows audit log retrieval and timestamp verification. Deployment artifacts are pushed to IPFS, with hashes stored on-chain. The frontend application provides log search and visualization using Web3.js. Access is controlled through MetaMask authentication. All interactions are cryptographically verifiable.

Algorithm 1: Smart Contract Logging
This algorithm is used to log CI/CD events into the blockchain using Solidity smart contracts.
Input: Log message from CI/CD pipeline
Output: Immutable on-chain log entry

Algorithm 2: Logging from Jenkins to Blockchain
This algorithm defines how the Jenkins pipeline triggers on-chain logging after a successful build.
Input: Build completion event
Output: Blockchain transaction recording build success

Algorithm 3: Verifying Blockchain Logs via Frontend
Used to retrieve and display logs from the blockchain using a Web3.js frontend interface.
Input: User request via UI
Output: Human-readable DevSecOps log list

Algorithm 4: RBAC with Smart Contracts
Defines access controls enforced on-chain.
Input: User role and action request
Output: Approved or denied access based on policy

Algorithm 5: Secure Rollback Trigger
Ensures failed builds or threats can be rolled back securely.
Input: Failed build or detected anomaly
Output: Rollback to last known good state

*E. Smart Contract Design*

The Solidity smart contract DevOpsLogger.sol.

*F. Blockchain Integration Layer*

Implemented via Node.js scripts:
logEvent.mjs – Blockchain Logger Script
This script is invoked by Jenkins at the end of the build and logs CI/CD events to the blockchain.
verifyLogs.mjs – Blockchain Log Query Script
Used from CLI for auditors to inspect logs stored on the blockchain.

*G. Frontend UI Design (MetaMask Integrated)*

- Built using HTML + JavaScript + Web3.js, the frontend allows:
- MetaMask wallet connection
- Fetching total log count
- Displaying each log message with timestamp
- Recording logs manually (for testing or validation)

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue VII July 2025- Available at www.ijraset.com*

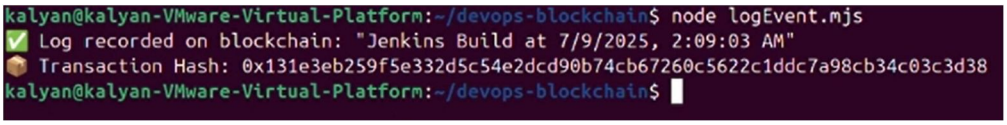



Fig3: CLI- Based Log Output

### *H. File Structure Overview*

The project is structured into modular directories to provide clarity and maintainability. The /contracts/ directory includes the DevOpsLogger.sol smart contract, and Truffle deployment files are found in /migrations/ and truffle-config.js. The /jenkins/ directory includes the Jenkinsfile for automating the CI/CD process. Blockchain interaction scripts such as logEvent.mjs and verifyLogs.mjs are found in /scripts/. The frontend interface, implemented using Web3.js and MetaMask, is located in /frontend/, and compiled artifacts such as the ABI are kept in the /build/ directory. The organization facilitates a clean delineation of blockchain logic, automation scripts, and user interface elements.

### *I. Implementation Tools*

| Layer | Technology | Function |
|---|---|---|
| CI/CD Automation | Jenkins, Maven | Builds and deploys the app |
| Containerization | Docker | Runs the app in a portable container |
| Smart Contract | Solidity (Truffle + Ganache) | Stores and emits logs |
| Backend Log Hook | Node.js | Interacts with blockchain |
| Frontend UI | Web3.js + MetaMask | Displays on-chain logs for transparency |
| Verification CLI | Node.js (verifyLogs.mjs) | Allows command-line log review |

TABLE 3: Tools & their functions

## V. IMPLEMENTATION

This part explains the end-to-end implementation of the Paper. The system was proposed to increase trust, traceability, and transparency in the software development process by combining blockchain technology with a conventional DevOps CI/CD pipeline.

### *A. Environment Setup and Toolchain Installation*

The environment was built on Linux with open-source tools. Jenkins handled CI/CD automation, Maven was employed for application building, and Docker was used for containerization. For the integration with blockchain, Ganache provided a local simulation of an Ethereum network, and Truffle handled smart contract compilation and deployment. Node.js with Web3.js supported blockchain interaction, with MetaMask offering wallet access in the browser. All tools were installed through default package managers and tested for compatibility.




Fig 4: Terminal showing all tool installations

©IJRASET: All Rights are Reserved 

### *B. Smart Contract Design and Deployment*

An immutable CI/CD logs recording Solidity smart contract, DevOpsLogger.sol, was created for the purpose. It contains a deployment of a struct to hold messages and timestamps, and functions to log and get logs. Truffle was used to deploy the contract on a local Ganache network. Successful deployment produced a contract address, allowing integration with the frontend and the CI pipeline.

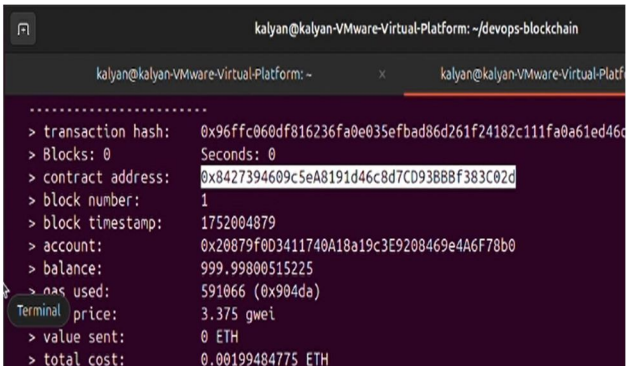



Fig 5: Truffle migration output with deployed contract address

### *C. Jenkins CI/CD Pipeline Configuration*

The Jenkinsfile orchestrates all CI/CD stages, starting from code cloning and Maven build, to Docker image creation and container deployment. At the final stage, the logEvent.mjs script logs the build success on the blockchain. The pipeline provides real-time feedback through console outputs and transaction confirmations, ensuring traceability of each build step.




Fig 6: Jenkins pipeline with all stages

### *D. Logging Scripts and Blockchain Interaction*

Two Node.js scripts handle blockchain operations:

- logEvent.mjs: Sends CI/CD log entries to the deployed smart contract.
- verifyLogs.mjs: Retrieves stored logs for audit and review purposes.

Both scripts utilize Web3.js and interact with Ganache using a preconfigured private key for local testing.

### *E. Web-based Log Viewer DApp*

A frontend DApp was built using HTML, JavaScript, and Web3.js. Users can connect via MetaMask to fetch and view logs directly from the blockchain. The UI also allows submitting custom logs, which are signed and stored on-chain. All interactions are verified in real-time, promoting transparency and user accessibility.

### *F. Integrated System Deployment*

The system runs across multiple localhost endpoints: Jenkins (:8080), Dockerized app (:8081), and Ganache (:8545). Logs generated from Jenkins were successfully recorded on-chain and retrieved via CLI and browser-based frontend. Each component worked seamlessly, contributing to a secure and traceable CI/CD workflow.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

| Component | Role |
|---|---|
| Jenkins | Automates CI/CD and triggers logging |
| Maven | Compiles and packages Java app |
| Docker | Hosts app in isolated container |
| Truffle | Compiles and deploys smart contracts |
| Ganache | Provides local Ethereum simulation |
| Node.js + Web3.js | Enables contract interaction |
| MetaMask | Wallet for authenticating transactions |
| HTML/JS Frontend | UI for real-time log visualization |

TABLE 4: Components and their roles

## VI. RESULTS AND SECURITY ANALYSIS

After implementation, the blockchain-integrated DevSecOps pipeline was thoroughly tested. Jenkins, Docker, Truffle, and the smart contracts all worked together to securely record CI/CD logs on an Ethereum-based blockchain. Each pipeline run—from code build to container deployment—triggered a transaction using the logEvent.mjs script, successfully storing logs on-chain. Transaction hashes confirmed immutability.

The verifyLogs.mjs script was used to fetch and verify logs from the smart contract. It retrieved logs in order, each with a timestamp and message. The MetaMask-enabled frontend also worked as expected, dynamically displaying blockchain-stored logs in the browser. This confirmed full-stack functionality and log integrity from pipeline to UI.

| Log Message | Timestamp (UTC) | Truncated Transaction Hash |
|---|---|---|
| Jenkins Build #1 | 2025-07-09 02:09:03 AM | 0x13e1b...3d38 |
| Jenkins Build #2 | 2025-07-09 02:09:10 AM | 0x2b04d...5fd5 |

TABLE 5: Sample Logged Events and Blockchain Transaction Metadata

Performance Observations:

System performance under local conditions was within acceptable limits. Blockchain write times averaged ~2.5s, log reads took <2s, and frontend fetches were near-instant.

| Operation | Avg. Execution Time | Outcome |
|---|---|---|
| Jenkins Build + Docker Run | ~25 seconds | Successful |
| logEvent.mjs (Write to Blockchain) | ~2.5 seconds per log | Successful |
| verifyLogs.mjs (Read from Chain) | < 2 seconds for 5 logs | Successful |
| Frontend Fetch (via MetaMask) | Instant (<1s) | Fully functional |

Table 6: System Performance Observations under Local Executi

Smart contracts proposed tamper-evident, cryptographically verifiable logging. A log was emitted as an event and history of the pipeline would be transparently reconstructed. Students were dismissed if their inputs were malformed, and duplicate entries were prevented. Secure wallet-based access was provided by MetaMask, where only authenticated users could write or query logs. This dispersed the insider threat and dispensed with reliance on changeable databases.

In general, the system showed high recoverability, transparency and security. The logs were written once, read once, and verified by both the scripts and the user-friendly frontend. Collaboration of DevOps tools with Block chain successfully provided solutions to tampering, traceability, auditability in CI/CD workflows.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

## VII. CONCLUSION AND FUTURE WORK

This research showed that it is possible to include blockchain in a DevSecOps pipeline to make it more secure, transparent and traceable. Injecting Ethereum smart contracts directly into the Jenkins CI/CD process, we constructed an immutable logging service to capture pivotal pipeline events in real time. The use of Web3. js and MetaMask provided a secure way for developers to interact with the blockchain and cryptographically trust build records.

We experimented with the Ganache for a local blockchain simulation, Truffle for interacting with their contracts, Docker for containerization and Jenkins for CI. Tests also demonstrated that both logs were properly written and serviced with low latency. A frontend that is viewable in a user friendly way with meta mask was created to watch these logs. Compared with the traditional methods, our system is resistant to tampering, and mitigates the insider threats by decentralizing control.

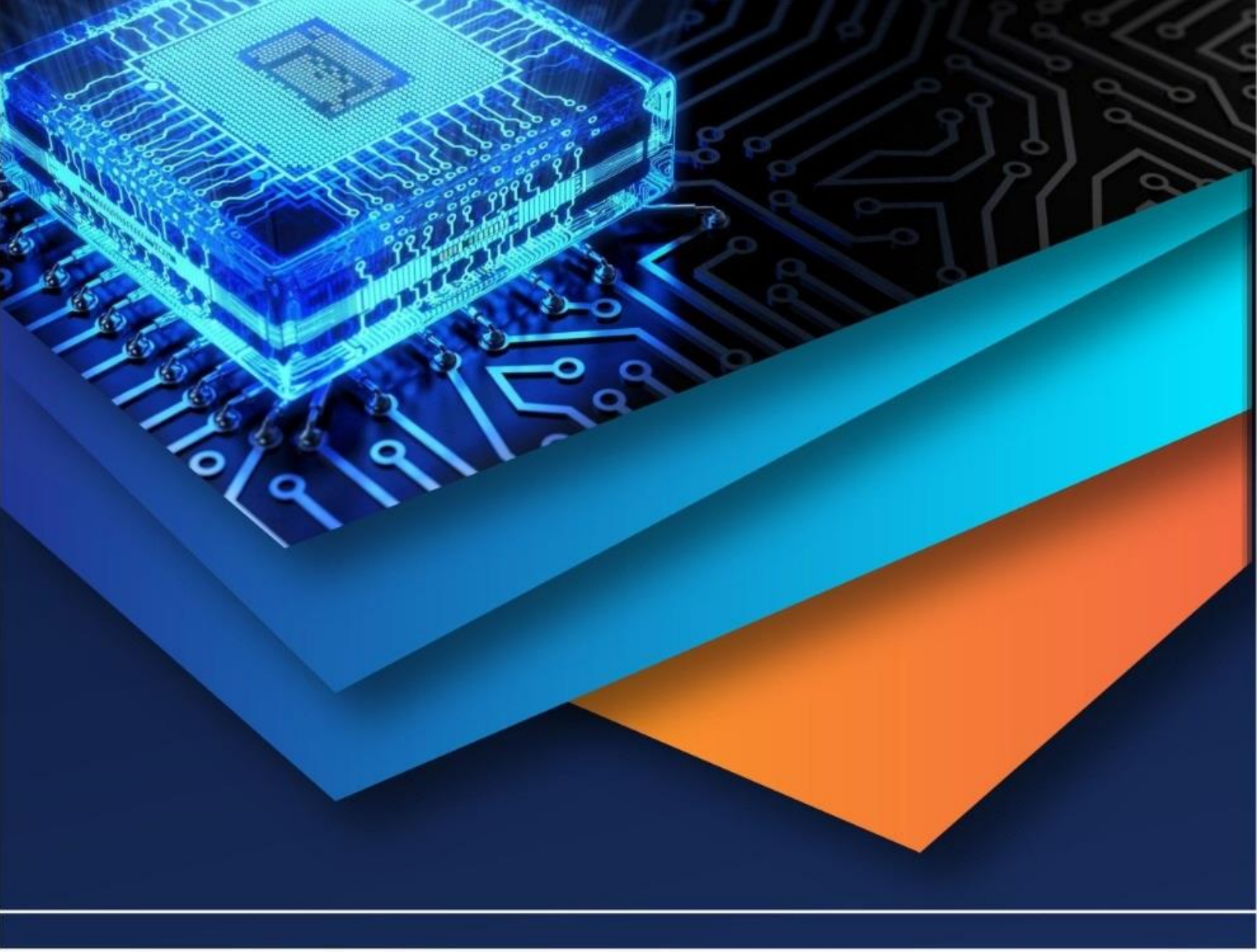
Though, it was only tested on a local network. This configuration does not simulate real-world performance concerns such as latency or gas fees on public blockchains. Moreover, the design accommodates only one CI/CD pipeline and a minimal UI despite supporting no features such as search or export.

Future enhancements involve deployment on public Ethereum networks, integrating more CI/CD tools such as GitLab, and incorporating decentralized identity (DID) for secure access. Layer-2 solutions such as Polygon can minimize costs, and UI advanced features can enhance usability. AI tools can further augment log analysis and automation of compliance.

## REFERENCES

[1] Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A comprehensive review of DevOps principles and associated challenges. ACM Computing Surveys, 52(6), 1–35. https://doi.org/10.1145/3359981
[2] Gall, M., & Pigni, F. (2022). Mainstreaming DevOps: Critical insights and a conceptual roadmap. European Journal of Information Systems, 31(5), 548–567. https://doi.org/10.1080/0960085x.2021.1997100
[3] Khan, A. A., & Shameem, M. (2020). A taxonomy of DevOps risk factors using AHP. Journal of Software: Evolution and Process, 32(10), e2263. https://doi.org/10.1002/smr.2263
[4] Akbar, M. A., Mahmood, S., & Siemon, D. (2022). Blockchain-driven DevOps: A scalable and efficient approach. In Proceedings of EASE '22 (pp. 421–427). ACM. https://doi.org/10.1145/3530019.3531344
[5] Bankar, S., & Shah, D. (2021). Integrating blockchain with DevOps for secure software pipelines. In Proc. ICNTE, 1–6. https://doi.org/10.1109/ICNTE51185.2021.9487760
[6] Faruk, M. J. H., Shahriar, H., Valero, M., & Rahman, A. (2022). Novel methods to mitigate software supply chain attacks. In IEEE ISSRE Workshops, 283–288. https://doi.org/10.1109/ISSREW55968.2022.00081
[7] Nayaka, P. S. K., Narayan, D. L., & Sutradhar, K. (2024). A review on secure DevOps metadata using blockchain. Security and Privacy, 7(2), e342. https://doi.org/10.1002/spy2.34
[8] Qureshi, J. N., & Farooq, M. S. (2024). ChainAgile: Enhancing agile DevOps using blockchain integration. PLoS ONE, 19(3), e0299324. https://doi.org/10.1371/journal.pone.0299324
[9] Farooq, M. S., Kalim, Z., Qureshi, J. N., Rasheed, S., & Abid, A. (2022). A distributed agile framework empowered by blockchain. IEEE Access, 10, 17977–17995. https://doi.org/10.1109/ACCESS.2022.3146953
[10] Lu, Y. (2019). Blockchain for industrial systems: Research gaps and challenges. Journal of Industrial Information Integration, 15, 80–90. https://doi.org/10.1016/j.jii.2019.04.002
[11] Gad, A. G., Mosa, D. T., Abualigah, L., & Abohany, A. A. (2022). Emerging trends in blockchain and its DevOps applications. Journal of King Saud University - Computer and Information Sciences, 34(9), 6719–6742.
[12] Khalil, I., Yau, K. L. A., & Naik, K. (2021). Blockchain-based cyber-physical system security: A review. Future Generation Computer Systems, 124, 91–118.
[13] Khan, A. W., Zaib, S., Tarimer, I., & Seo, J. T. (2022). Cybersecurity challenges in DevOps software environments. IEEE Access, 10, 65044–65054. https://doi.org/10.1109/ACCESS.2022.3179822
[14] Marandi, M., Bertia, A., & Silas, S. (2023). Automation of security scanning in a DevSecOps pipeline. In WCONF 2023, 1–6.
[15] Diel, E., Marczak, S., & Cruzes, D. S. (2016). Communication issues in global DevOps teams. In ICGSE 2016, 24–28. https://doi.org/10.1109/ICGSE.2016.28
[16] Shahin, M., Babar, M. A., & Zhu, L. (2017). CI/CD: A review of tools and challenges. IEEE Access, 5, 3909–3943.
[17] Prates, L., Faustino, J., Silva, M., & Pereira, R. (2019). A metrics-driven approach to DevSecOps. In IS 2019, 77–90.
[18] Tariq, F., & Colomo-Palacios, R. (2019). Smart contracts in secure DevOps workflows. In LNCS: ICCSA, 327–337. https://doi.org/10.1007/978-3-030-24308-1_27
[19] Salama, R., Al-Turjman, F., & Kumar, S. (2023). Blockchain-driven cybersecurity: An extensive survey. In CICTN, 774–777.
[20] Warmke, C. (2024). What is Bitcoin: Philosophical and technical implications. Inquiry, 67(1), 25–67.
[21] Ahmad, J., Zia, M. U., & Naqvi, I. H. (2024). Blockchain and machine learning for secure DevOps pipelines. WIREs Data Mining and Knowledge Discovery, 14(1), e1515.
[22] Sunyaev, A. (2020). Blockchain-based Web services and SaaS architecture. In Internet Computing, 155–194. https://doi.org/10.1007/978-3-030-34957-8_6
[23] Sharma, T., & Sharma, P. (2024). AI and cybersecurity convergence for threat detection in CI/CD. In IGI Global, 81–98.
[24] Letafati, M., & Otoum, S. (2023). Privacy models for secure blockchain-led e-health DevOps. Ad Hoc Networks, 150, 103262.
[25] Mezquita, Y., Podgorelec, B., & Corchado, J. M. (2023). Interoperability model for blockchain in distributed systems. Sensors, 23(4), 1962

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)