A Hybrid Framework to Implement DevOps Practices on Blockchain Applications (DevChainOps)

Ramadan Nasr, Mohamed I. Marie, Ahmed El Sayed
Department of Information Systems-Faculty of Computers and Artificial Intelligence, Helwan University, Cairo, Egypt

***Abstract*—As the adoption and utilization of blockchain technology continue to expand in enterprise software development, integrating the established DevOps approach emerges as a rational decision. This integration has the potential to accelerate software development and delivery, enhance software quality, and improve overall productivity. However, there is currently a lack of guidance on a structured DevOps approach, specifically within the realm of blockchain-based software development. This paper emphasizes the importance of implementing an effective DevOps process and investigates its utilization in the development of blockchain smart contracts. Specifically, this study introduces a framework that aims to seamlessly integrate DevOps into the process of smart contract development. Specifically, this research paper presents a framework that has been developed to seamlessly incorporate DevOps principles into the process of smart contract development. The primary focus of this framework is to streamline the continuous delivery and deployment of blockchain smart contracts packaged in containers. It comprises two fundamental components: delivery and deployment, which communicate through Git-distributed version control. Smart contract applications and node-specific deployment configurations are stored in dedicated GitHub repositories. The delivery component guarantees the synchronization of the deployment package with the most recent version of the smart contract application and the node deployment configuration files. The deployment component, meanwhile, is responsible for executing blockchain-decentralized applications in containers across all blockchain nodes. It leverages GitHub, Jenkins, and Docker for this purpose. To validate its effectiveness, multiple tests have been conducted on Quorum's simple storage, Sawtooth's XO Integerkey, and Corda's token decentralized applications (dapps) dappsto evaluate the effectiveness of the proposed method.**

Keywords*—Blockchain; decentralized applications (dapps); DevOps; smart contracts; continuous integration (CI); continuous deployment (CD); model-driven development (MDD)

I. Introduction

The Agile Manifesto's key concept emphasizes the significance of early and consistent software delivery to meet customer requirements. In the domain of software development projects, organizations strive to adopt diverse development practices, with a predominant inclination towards agile methodologies [1].

An integral practice within this framework is DevOps, a software development methodology that particularly aims to enhance collaboration among disparate teams engaged in a project. Noteworthy is its substantial emphasis on fostering efficient cooperation between the development and operations teams, hence the derivation of its name [2, 3]. As a result, all teams involved experience increased productivity through efficient time utilization, leading to shorter software development cycles and enhanced product quality [4].

The rise in popularity of smart contracts in recent years can be attributed to the growing fascination with cryptocurrencies. Initially conceived as a way to facilitate transactions within digital currencies like Bitcoin and Ethereum [5], smart contracts have since evolved to encompass a wide range of applications beyond their original purpose. Researchers have identified numerous potential uses for smart contracts, including in areas such as supply chain management, healthcare, finance, and legal agreements. This broadening scope highlights the versatility and transformative potential of smart contracts in various industries.

Smart contracts serve as protocols that enable and enforce agreements between multiple parties on a blockchain. These contracts are self-executing, with the terms of the agreement directly written into code, which runs on a decentralized blockchain network. The decentralized nature of blockchain technology necessitates specific considerations during the development and deployment of smart contracts [6, 7]. Unlike traditional software, where updates and patches can be easily applied, smart contracts often cannot be modified once they are deployed on the blockchain. This immutability, while enhancing security and trust, also means that ensuring high software quality and reliability during development is crucial.

Smart contracts typically do not undergo the same software life cycle as regular applications, where new code versions can introduce enhancements or address issues. As a result, ensuring high software quality and reliability is crucial during development.

DevOps plays a key role in offering support through test automation and the creation of stable operating environments, among other aspects. As demonstrated in [8], frequent changes to the DevOps process may lead to unanticipated delays, posing a significant challenge for smart contracts due to the rapid evolution of programming languages [9] and the subsequent demand for new and more comprehensive tools and development strategies.

However, Wohrer and Zdun [10] have effectively shown the viability of implementing specific facets of DevOps on an Ethereum blockchain. Fully implementing all DevOps steps is essential for guaranteeing the success of a project. In addition, several alternative processes built on agile development principles have been proposed in [11-13].

These studies affirmed the feasibility of establishing a comprehensive framework for working with blockchains and smart contracts.

Finally, the transition to DevOps comes with obstacles, especially in the context of smart contracts, for the reasons outlined earlier. This paper presents a novel framework for directing the adoption of DevOps practices in the development and deployment of blockchain smart contracts. The major contribution is DevChainOps, which simplifies the development and deployment process by integrating tools like Visual Paradigm, GitHub, and Jenkins. It covers the entire CI/CD pipeline, from creating UML deployment diagrams to deploying smart contract dapps within the blockchain network.

The subsequent sections of this document follow the following structure: Section II provides a comprehensive overview of both DevOps and blockchain, providing pertinent contextual information. Section III explores the existing body of research on the subject.While, Section IV provides a detailed explanation of the design of the proposed framework (DevChainOps) that applies the DevOps practices to smart contract applications.

Three use cases are employed in this section to illustrate the essential steps for implementing CI/CD practices on Quorum, Sawtooth, and Corda blockchains.. The experimental results, including tests conducted on the distributed applications are discussed in Section V Finally, in Section VI, conclusions are drawn and future work is outlined.

II. Background

This section provides an overview of DevOps components and Blockchain smart contracts.

A. Components of DevOps

Two key components of DevOps are Continuous Integration (CI) and Continuous Delivery/Deployment (CD), which support the DevOps principle of merging the two main disciplines through automation. Fig. 1 shows the sequence of steps in CI/CD practices.

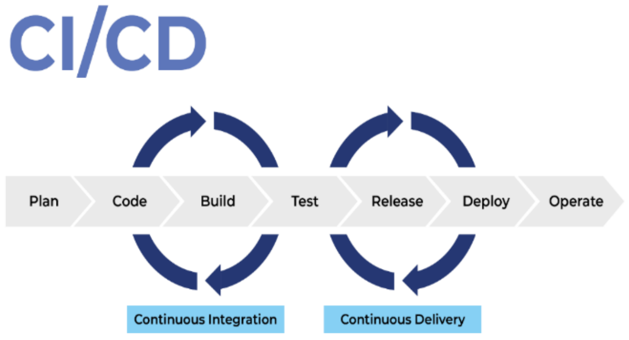


Fig. 1. Sequence of steps in CI/CD practices.

The CI phase involves continuously integrating software throughout its development cycle. This includes automating software builds and testing processes, often using a version control system like GitHub [14]. Developers routinely merge their code with primary branches [15, 16], after which the newly introduced or modified code is integrated into a build and subjected to verification through automated tests.

CI enhances team productivity with frequent releases and improves software quality through iterative testing. Overall, CI generates various outputs, such as compiled executables or libraries, suitable for use in other projects.

The CD involving the deployment of the artifact across different product environments. While CD activities do not handle security, confidentiality, or infrastructure concerns [17], they enable system component updates and facilitate more frequent feedback from diverse teams to developers due to process automation [18].

CD can be classified into two forms: continuous deployment and continuous delivery. Continuous delivery is a non-automated process that results in an artifact ready for deployment, while continuous deployment pushes the final artifact to a system without decision-making.

B. Blockchains Smart Contract

Blockchains serve as secure platforms for transactions, integrating computational and economic principles. Smart contracts, immutable code on the blockchain, automate business processes and ensure execution according to predefined terms [19].

They are fundamental to blockchain services, with Ethereum being a prominent platform, while other blockchains also support them [20, 21]. These contracts enable decentralized transactions and rely on consensus algorithms to maintain data integrity [22, 23]. Table I presents a comprehensive comparison between the most commonly used blockchain platforms [24].

The proposed framework presented in this paper encompasses Quorum, Hyperledger Sawtooth, and R3 Corda blockchain platforms to validate its effectiveness. In the following, brief descriptions and some details related to those platforms are presented.

- Quorum [25], developed by JP Morgan, is a permissioned ledger platform that uses a modified version of the Ethereum Virtual Machine (EVM) and Solidity language for private transactions. It focuses on enterprise and business needs, ensuring increased security by limiting transactions to authorized users within an organization and protecting confidential information.

- Fig. 2 depicts the architecture diagram of a Quorum node. The source code is derived from geth (the Ethereum Go client) and has been adapted to function within a permissioned environment, as previously discussed. This involves overseeing communication with clients and other nodes via the general-purpose Constellation peer-to-peer system for secure messaging [27].

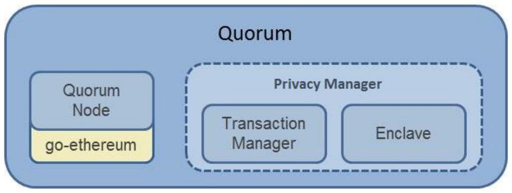


Fig. 2. Quorum architecture [26].

TABLE I. ILLUSTRATES A COMPARISON BETWEEN THE MOST COMMONLY USED BLOCKCHAIN PLATFORMS

Platform	Type	Smart Contract Language	Consensus Mechanism	Main Application Contexts	Related Projects
Ethereum	Public	Solidity, Vyper	Proof of Work	Financial, Asset trading	DAI Gitcoin Cryptokitties
Quorum	Private	Solidity, Vyper	Proof of Authority	Financial, Supply Chain and Logistics	Liink Komgo Project Ubin
Hyperledger Fabric	Public, Private	Java, Go, JavaScript	Proof of Work	Supply chain, Trade finance, Stock trading	IBM Food Trust Everledger diamond blockchain
Hyperledger Sawtooth	Private	Rust, Go Python, Java	PoET, PBFT, RAFT	Supply chain, Provenance Tracking	Sawtooth Private UTXO Sawtooth Marketplace
NEM	Private	Java	Proof of Importance	Augmented reality, Advertising and marketing, Banking	DigitCoin Bankera Pantos
Stellar	Public	Solidity, JavaScript, Java, Go	Stellar Consensus Protocol	Remittance	StellarX Tempo TillBilly
Corda	Private	DAML, Kotlin, Java	Validity and Uniqueness	Energy trading, Insurance, Retail markets	Energy Block Exchange TradeCloud MonetaGo

Within the Quorum node, the Constellation module consists of two sub-modules: Transaction Manager and Enclave. The Transaction Manager handles private transactions by facilitating access to them, transmitting encrypted data payloads to other Transaction Managers on different Quorum nodes, and utilizing the Enclave for cryptographic operations. Meanwhile, the Enclave functions independently by securely housing all private keys associated with transactions and conducting all encryption and decryption procedures internally.

- Hyperledger Sawtooth is an enterprise-grade blockchain platform specifically developed for distributed ledger applications and networks [28]. The design philosophy of this platform is oriented towards preserving ledger distribution and guaranteeing the safety of smart contracts. Sawtooth streamlines the development of blockchain applications by delineating the core system from the specific application domain, allowing developers to define specific rules using their preferred programming languages. Its high modularity allows enterprises and consortiums to make policy decisions based on their expertise. Fig. 3 shows a high-level view of the Sawtooth architecture.

Sawtooth is specifically designed for managing business supply chains rather than for cryptocurrency applications. The transaction process starts with the client organizing all transactions into a block, followed by signing the batch and sending it to a validator. The validator then employs its transaction processor to verify the integrity of the batch before committing it.

Sawtooth parallelly executes transactions through a REST API to enhance performance. Its modular nature includes various features such as consensus algorithms, rule sets, programming languages, and smart contracts, enabling efficient adaptation based on specific business requirements. Programmers have the flexibility to utilize Python, JavaScript, Go, C++, Java, or Rust for building and interacting with the Sawtooth blockchain. Sawtooth currently supports four different consensus algorithms: Dev_mode, PoET, Sawtooth PBFT, and RAFT [28].

- Corda is a blockchain project available for public use, created specifically for business purposes, allowing participants to engage in direct transactions using smart contracts. [29]. It optimizes operations by reducing transaction costs and simplifying record-keeping. Corda is scalable and adaptable to diverse business needs, with applications like CorDapps designed to revolutionize industries like insurance, healthcare, finance, and energy. Fig. 4 illustrates the various node types within the R3 Corda Architecture network.

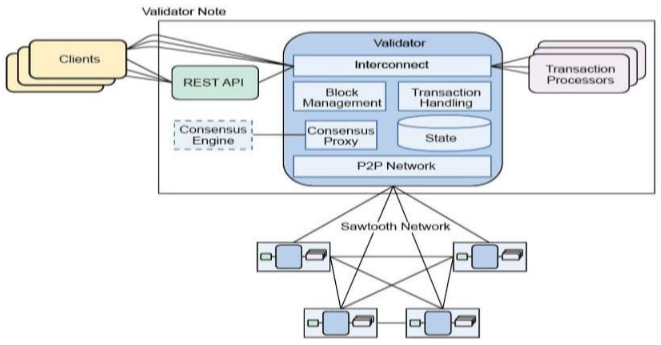


Fig. 3. Sawtooth architecture [28].

The consensus mechanism employed by R3 Corda exclusively involves nodes actively participating in a transaction, significantly impacting scalability. DLT nodes are the foundation for distributed applications and services, forming a fully connected graph. Transactions are verified by a notary node, and the ledger includes network map and Oracle nodes.

Communication between DLT nodes and the notary node uses AMQP/TLS, while secure communication between DLT nodes and Oracle nodes and the network map uses HTTPS.

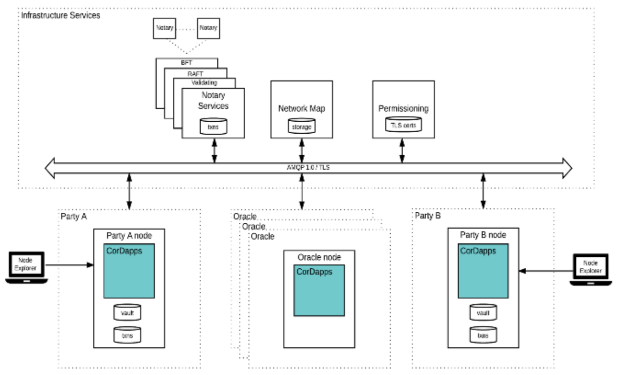


Fig. 4. Corda architecture.

III. Related Work

This paper gives prominence to the pivotal DevOps practices, specifically CI/CD, in addition to investigating blockchain technology and UML. Accordingly, the literature review encompasses the work related to these three areas. The first section delves into papers discussing recent advancements in continuous practice. The subsequent section examines articles that exemplify the most recent advancements in blockchain technology. Lastly, another section encompasses studies that showcase the latest applications of UML.

Considerable focus has been directed towards identifying such applications within blockchain solutions. Although various approaches and methodologies for CI/CD have been proposed in the literature [30], translating these theories into practice can pose significant challenges.

Laukkanen et al. [31], the objective of this study was to address the complexities encountered during the adoption of Continuous Delivery (CD) in software development, the authors conducted a comprehensive review by searching five major bibliographic databases, identifying 293 articles related to CD. Of these, 30 articles were selected for detailed qualitative analysis based on their empirical evidence and focus on practical implementation rather than just tooling. The review identified a total of 40 problems, 28 causal relationships, and 29 solutions associated with CD adoption. Testing and integration issues were the most frequently reported, with system design and testing problems having significant causal relationships with other issues, finally, their study provides valuable insights for both practitioners and researchers by synthesizing the existing knowledge on CD adoption challenges and solutions, offering a foundation for future research and practical guidance for organizations attempting to implement CD.

Abdalkareem et al. [32] have streamlined the execution time of continuous integration processes by identifying redundant commits and employing a prototype tool compatible with Git repositories.

Gallaba et al. [33] have introduced a tool designed to scrutinize feature misuse within Travis CI. Similarly, Saidani et al. [34] utilize the Travis CI platform to examine refactoring techniques within continuous integration. Numerous research has highlighted the importance of the widely used Jenkins automation server, which is popular among both academic and professional communities [35, 36].

Yu et al. [37] examined the application of continuous integration platforms in the evaluation of non-functional specifications. Recently, Leite et al. [38] conducted a comprehensive investigation into the practices of continuous delivery, with a specific focus on the organizational structure of DevOps teams and their patterns of communication. Their research highlights the significance and timeliness of their study in the field.

Blockchain is acknowledged as one of the most impactful technologies. Chowdhury et al. [39] have conducted a comparative study of permissioned and permission less blockchain frameworks. The study discussed the design principles, consensus mechanisms, and security considerations in frameworks like Corda, Hyperledger Fabric, and Quorum.

Notably, blockchain finds widespread utility within the energy sector, especially in the regulation of electrical flow within decentralized energy systems for consumers. Jamil et al. [40] propose a predictive energy trading system to optimize energy generation scheduling from renewable sources. Additionally, Saxena et al. [41] proposed a blockchain-based system for facilitating residential energy trading to align with consumer’s preferences for reducing energy demand using the permissioned Hyperledger Fabric.

In the realm of blockchain frameworks, researchers and practitioners utilize various frameworks, such as those scrutinized by Monrat et al. [42] and Al-Jaroodi et al. [43]. Both scrutinize the advantages and challenges associated with employing this technology in business applications, but their focus is limited. Healthcare data management extensively leverages blockchain [44], as demonstrated in Shahnaz et al.'s [45] presentation of an electronic health record system utilizing blockchain.

Furthermore, the use of UML is still the focus of this study. Practitioners employ the Unified Modeling Language for the purpose of software architecture modeling, representing architecture from diverse perspectives. For instance, Chavez et al. [46] have focused on achieving cohesion between Java source code and UML class diagrams. UML models can also incorporate Object Constraint Language (OCL) to provide clear semantics.

Lu et al. [47] illustrate the utilization of OCL restrictions for medical regulations in cancer registries, utilizing UML class diagrams as well. Recent studies demonstrate an increasing range of applications for UML in conjunction with model-driven development (MDD). Arora et al. [48] employed a bio-inspired approach to analyze the concurrent portion of a UML activity diagram, leading to the identification of several viable test scenarios. Meanwhile, study [49] utilized UML class diagrams to produce variants of product line architecture. Additionally, Arcaini et al. [50] introduced a method that combines tests created for subsystems to generate tests for the entire system model.

Moradi et al. [51] demonstrated a technique for converting the model of services into executable web services. Other studies have investigated the effectiveness of this transformation process. For example, Panach et al. [52] observe that software quality resulting from MDD surpasses that of manually written code, particularly for complex issues, while Basciani et al. [53] illustrate the reusability potential of combining existing transformations to formulate novel design approaches. The emerging domain for employing MDD encompasses blockchain technology. Within blockchain, a pivotal component is the smart contract.

Zou et al. [30] conducted an investigation to identify the genuine hurdles faced by developers during smart contract development. The results revealed security vulnerabilities within the source code of smart contracts. Furthermore, the existing frameworks were found to be rudimentary, with several constraints in the programming language. Górski et al. [54] present a methodology designed to produce node deployment packages specifically for the Corda blockchain platform.

Meanwhile, Xu et al. [55] delineate two distinct transformations integrated into blockchain. The first transformation utilizes collaborative business processes to generate smart contracts, while the second implements blockchain registries for commodities, ownership titles, and digital assets using the Ethereum blockchain platform. Moreover, Gao et al. [56] created a tool prototype to automatically detect bugs and validate smart contracts. Laukkanen et al. [57] emphasized the scarcity of documented solutions pertaining to system design and build design aspects within the Continuous Delivery field, as highlighted in their survey.

Zou et al. [30] assert that existing tools for blockchain application development exhibit notable deficiencies in providing comprehensive support. The authors provide a comprehensive analysis of the current state of smart contract development, focusing on both the challenges faced and the potential opportunities for advancement. The authors identify key issues such as security vulnerabilities, lack of robust development tools, and the need for formal verification methods. The study highlights the limitations of existing programming languages and virtual machines, emphasizing the infancy of Solidity and the Ethereum Virtual Machine (EVM). Additionally, the paper discusses the importance of community support and the necessity for best practices in coding, testing, and debugging smart contracts. It concludes by suggesting future research directions, including improvements in security tools, development frameworks, and language features to enhance the reliability and efficiency of smart contract development.

To sum up, previous studies may have lacked a comprehensive CI/CD pipeline designed for blockchain systems. This could have negatively impacted the efficiency and reliability of deploying blockchain applications. Additionally, there may have been challenges in automating deployment packages for distributed ledger technologies, resulting in manual processes prone to errors and delays. The absence of integration with automation tools like Jenkins could have also hindered the streamlining of deployment processes. In response, this paper proposes a systematic approach for implementing DevOps, customized specifically for smart contracts.

The proposed framework in this paper offers UML modeling support specifically for the deployment aspect of smart contract dapps, filling a gap in current blockchain research, which mainly emphasizes smart contracts as integral components of such dapps. Furthermore, appropriately configured blockchain nodes function as the deployment environment for these smart contract dapps.

In summary, the framework incorporates the Visual Paradigm modeling tool and leverages GitHub and Jenkins for automated build releases, as well as Docker containers for smart contract testing and deployment, thereby integrating the entire CI/CD pipeline from the UML deployment diagram to the deployment of smart contracts within the blockchain distributed ledger network.

IV. The Proposed Framework

The proposed framework presents a structured approach to integrating DevOps practices tailored for blockchain applications. This integration aims to improve the development and deployment processes for smart contracts. The framework emphasizes streamlining the continuous delivery and deployment of blockchain smart contracts. By utilizing tools like GitHub, Jenkins, and Docker, it ensures effective synchronization and execution of decentralized applications across blockchain nodes.

This section presents the proposed framework, known as DevChainOps. DevChainOps utilizes model-to-code transformation to generate scripts that streamline the deployment of blockchain smart contracts dapps. DevChainOps is designed to automate the delivery and deployment process of smart contract dapps.

Fig. 5 illustrates the architecture of the proposed framework. The framework comprises three primary layers: UML transformation, version control, and the CI/CD automation server.

In the UML Transformation layer, the UML deployment diagrams of the blockchain network have been built along with the UML profile for distributed ledger deployment [58]. These form the basis of the transformation process. With the modularity architectural principle in consideration, the transformation design has been divided into two primary components: a node config generator application and a transformation plug-in that integrates the transformation with the Visual Paradigm modeling tool.

The UML deployment diagrams may be stored in various formats, depending on the modeling tool. So, we have used the Application Programming Interface (API) of the Visual Paradigm modeling tool to get the complete set of nodes with specified tagged values.

Then the Java Node config generator application reads the proper configuration file templates and generates deployment configuration files tailored for the chosen blockchain platform.

GitHub repositories serve as the version control layer for storing the source code of blockchain smart contracts, as well as nodes deployment configuration files created using the transformation plugin, JenkinsFile, and docker-compose files to run and test the smart contracts.

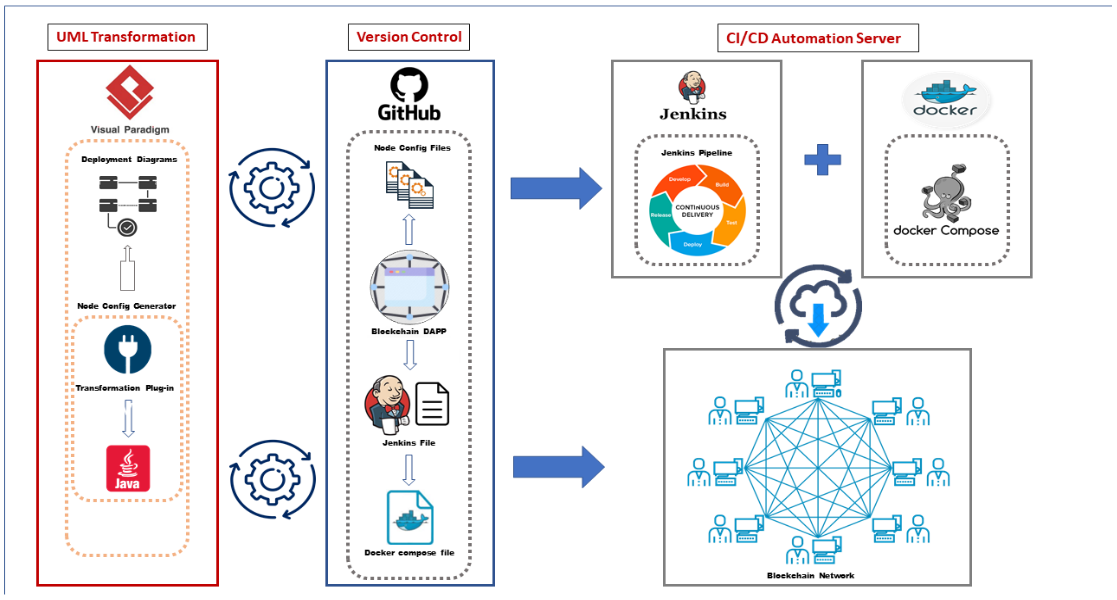


Fig. 5. The proposed framework architecture.

The Jenkins automation server is utilized in the CI/CD automation server layer to streamline the continuous integration and continuous delivery (CI/CD) process through the use of pipelines. This pipeline streamlines the process of developing, testing, and deploying smart contract.

The following subsections outline the three layers of the DevChainOps framework, comprising UML transformation, Version control, and the CI/CD automation server.

A. UML Transformation Layer

In the UML Transformation layer, the UML deployment diagrams are divided into component and deployment types, illustrating the physical components of software used by a system and the deployment environment of the designed system.

These diagrams provide a comprehensive representation of the blockchain dapp and its deployment environment, allowing for a better understanding of the elements involved in deploying a blockchain smart contract dapp.

The deployment diagrams, incorporating the UML Profile for blockchain dapp deployment, serve as the source of transformation and consist of services, nodes, and communication links. A single UML deployment diagram or multiple diagrams can be employed to represent the UML deployment model of the blockchain dapp.

The UML deployment diagrams for Quorum simple storage, Sawtooth xo Integerkey, and Corda tokens dapps are shown in Fig. 6, 7, and 8, respectively. While Fig. 9 shows the flowchart of the Node config generator component.

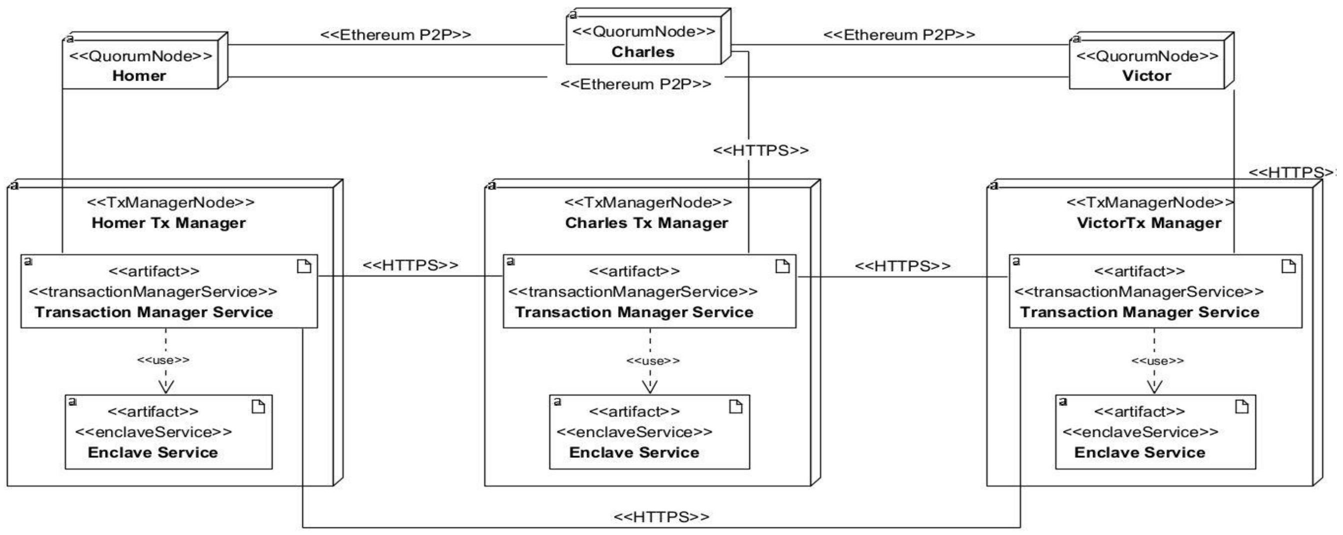


Fig. 6. Deployment diagram of Quorum Simple Storage Dapp.

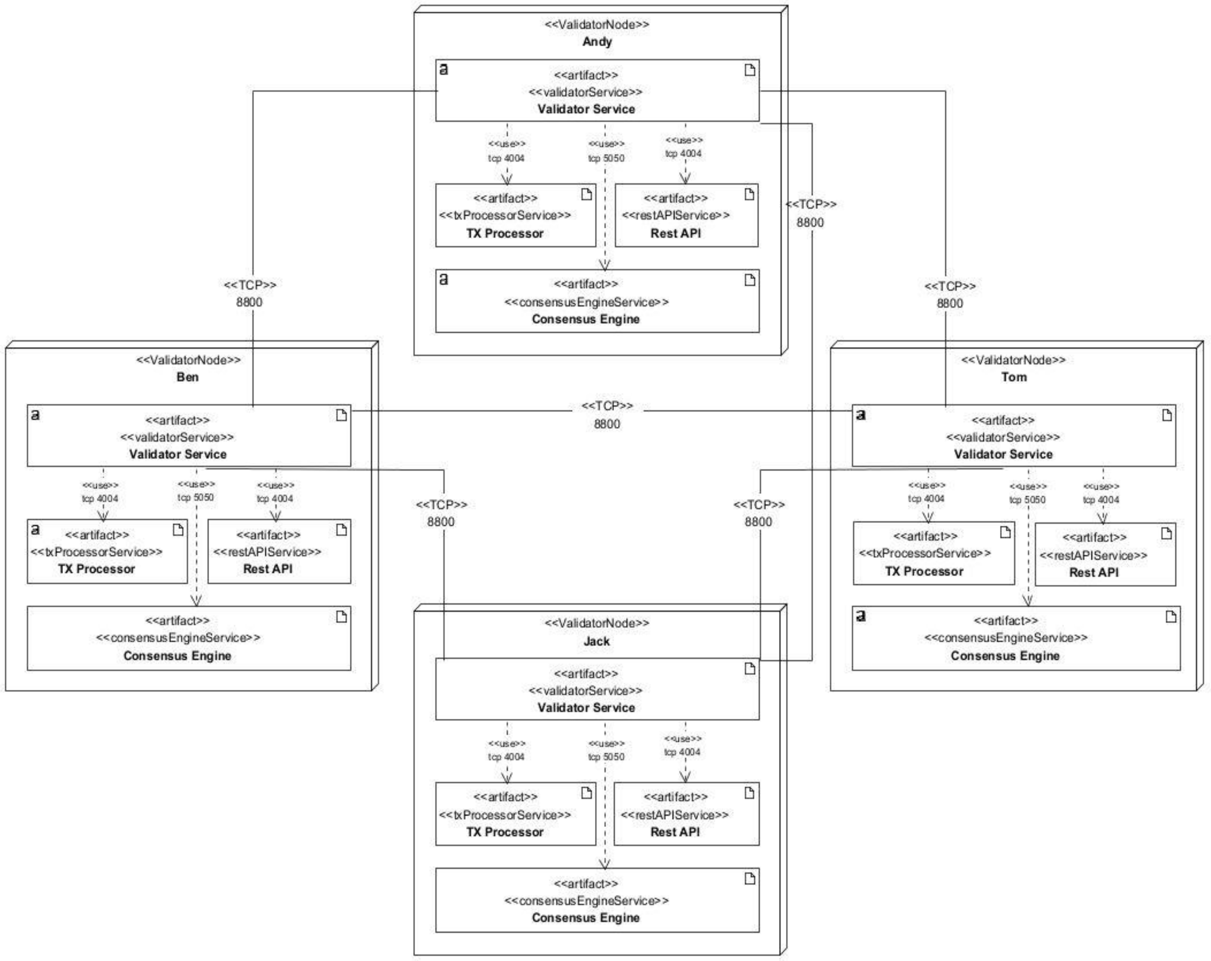


Fig. 7. Deployment diagram of Sawtooth xo intkey Dapp.

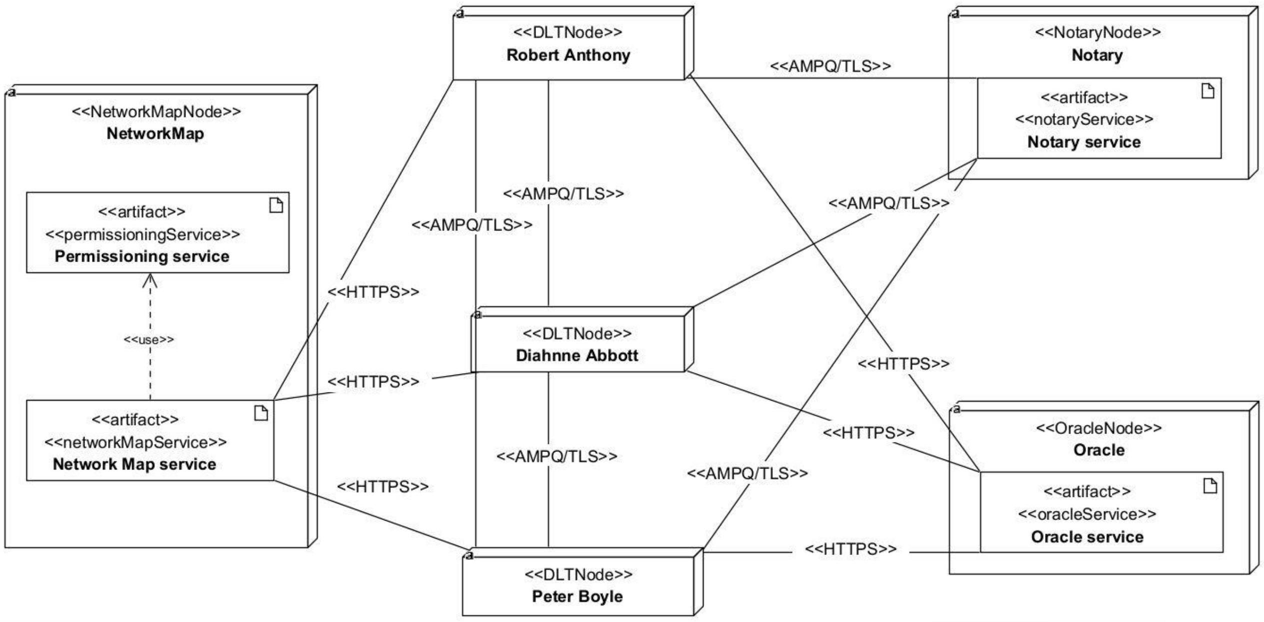


Fig. 8. Deployment diagram of Corda Tokens Dapp.

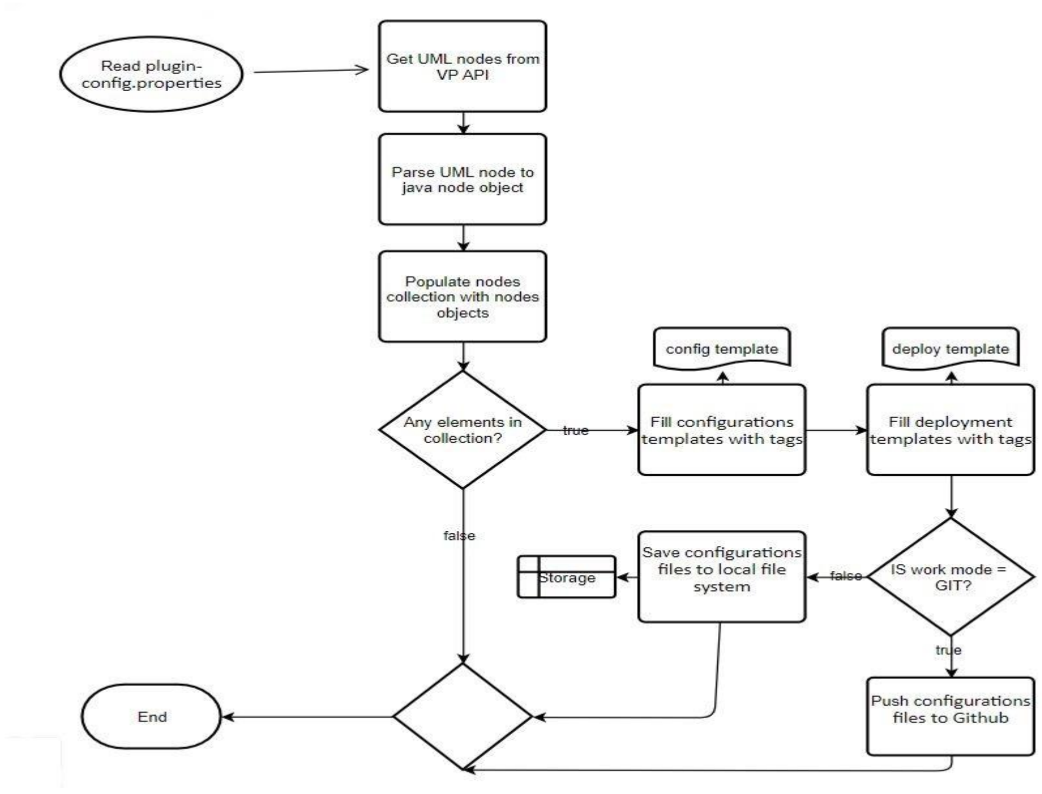

Read plugin-config.properties
Get UML nodes from VP API
Parse UML node to java node object
Populate nodes collection with nodes objects
Any elements in collection?
true
false
config template
Fill configurations templates with tags
deploy template
Fill deployment templates with tags
IS work mode = GIT?
false
true
Save configurations files to local file system
Storage
Push configurations files to Github
End

Fig. 9. The flowchart of Node config generator.

The Node config generator is a Java application that consists of classes and interfaces that correspond to Visual Paradigm API classes and UML nodes in which configuration file templates are utilized, filled with data from the UML Deployment Diagram, to generate specific configuration files needed for deploying the blockchain dapp. Fig. 10 presents interfaces and classes in the inheritance tree of the Node config generator.

By leveraging the Visual Paradigm API, we have established a flexible connection between the node configuration generator and the UML deployment diagram. The mapping between the Visual Paradigm API and the classes of the Node configuration generator is illustrated in Table II.

The API enables the manipulation of the UML deployment model by employing an object-oriented method, representing UML elements within the source code as interfaces, classes, and objects. Specifically, in Java source code, the creation of appropriate objects is achieved using the StereotypesEnum data type.

The Node configuration generator employs the transformation plugin, which functions in two different modes: LOCAL and GIT.

The LOCAL mode is retained for backward compatibility. In the LOCAL mode, users are prompted to designate a local path for storing generated files.

Enabling GIT mode causes the created deployment configuration files to be automatically committed and pushed to the selected repository. The plugin's work mode selection is governed via a specific property file named plugin-config.properties.

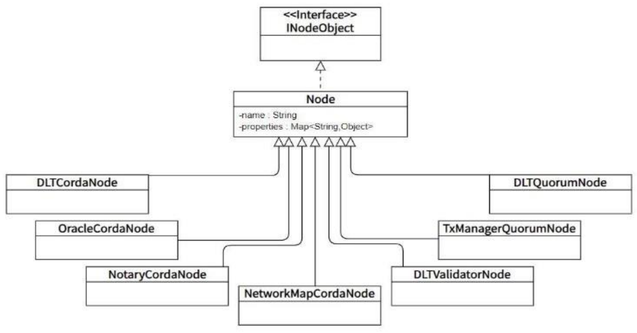


Fig. 10. The Node config generator class diagram.

TABLE II. THE MAPPING BETWEEN THE TOOL API AND THE CORRESPONDING JAVA CLASS

API element	Node config generator class
INode with < QuorumNode >	DLTQuorumNode
INode with < TxManagerNode >	TxManagerQuorumNode
INode with < ValidatorNode >	DLTValidatorNode
INode with < DLTNode >	DLTCordaNode
INode with < NetworkMapNode >	NetworkMapCordaNode
INode with < NotaryNode >	NotaryCordaNode
INode with < OracleNode >	OracleCordaNode

The transformation plugin examines the plugin-config file and specifies the location for saving the generated files. Utilizing the constructor of the PluginConfiguration class, the plugin defines the storage location (e.g., ramadannasr/xo-intkey-sawtooth-pbft/main/nodeConfigFiles).

The NodeConfigGenerator class of the transformation plugin calls upon the generateConfigFiles() method within Diagram2CodeTransformer's selected platform NodeManager class and provides it with generation destination information. The transformation process generates files and stores them in the designated location using the store() function.

In GIT mode, once the transformation for the designated environment is executed, The generated files are automatically pushed to the designated repository.

Samples of these newly generated deployment setup files for the three blockchain platforms are accessible in the ramadannasr/DeploymentConfigurationFiles repository in the main branch.

The GitHub repositories [59] contain source code for the Node config generator and transformation plugin. This transformation ensures the alignment of the UML deployment diagram with the generated configuration files for smart contract deployment on nodes.

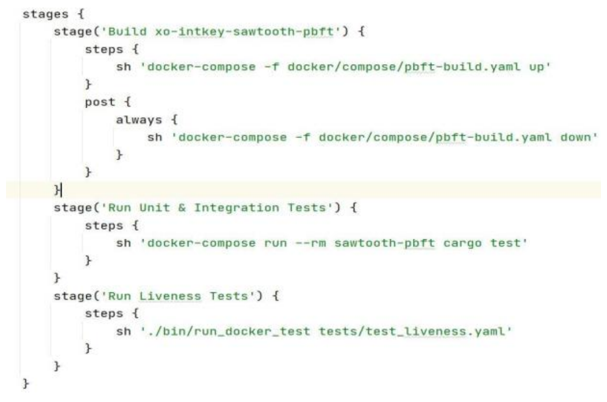
B. Version Control Layer

The version control layer includes Git repositories containing the smart contract application source code, configuration files for nodes created by the transformation plug-in, a JenkinsFile, and a docker-compose file for running and testing the smart contract application.

The JenkinsFile is used to define a Jenkins pipeline, providing a more concise and structured way to implement a basic three-stage continuous delivery pipeline (build, test, deploy).

The JenkinsFile is stored alongside the blockchain smart contract application code being built, so changes to the pipeline can be tracked along with changes to the code.

Fig. 11 shows the Jenkins file for Sawtooth XO integerkey Dapp. Three GitHub repositories were established for each smart contract application: one for Sawtooth xo integerkey, another for Quorum simple storage, and the third for Corda tokens [60, 61, 62].



```
stages {
    stage('Build xo-intkey-sawtooth-pbft') {
        steps {
            sh 'docker-compose -f docker/compose/pbft-build.yaml up'
        }
        post {
            always {
                sh 'docker-compose -f docker/compose/pbft-build.yaml down'
            }
        }
    }
    stage('Run Unit & Integration Tests') {
        steps {
            sh 'docker-compose run --rm sawtooth-pbft cargo test'
        }
    }
    stage('Run Liveness Tests') {
        steps {
            sh './bin/run_docker_test tests/test_liveness.yaml'
        }
    }
}
```

Fig. 11. The fragment of the JenkinsFile of Sawtooth xo intkey Dapp.

C. CI/CD Automation Server Layer

In the CI/CD Automation Server layer, The Jenkins automation server has been employed to support the continuous integration and delivery procedure by utilizing distributed version control systems like Github. A pipeline is a fundamental concept in Jenkins. It is an automated process that generates a release-ready package from software stored in a Git version control system It includes stages, with each stage block representing a distinct subset of tasks within the build process. Significantly, a proper node deployment package includes both smart contract business application and deployment configuration scripts. Therefore, one pipeline has been built to address this requirement, as shown in Fig. 10. The Jenkins pipeline automates the build, test, and deployment of the smart contract application. It triggers whenever new business logic source code is committed to the repository or a new deployment configuration is produced and pushed by the Node config generator.

Fig. 12 shows the UML activity diagram of the DevChainOps framework Jenkins pipeline.

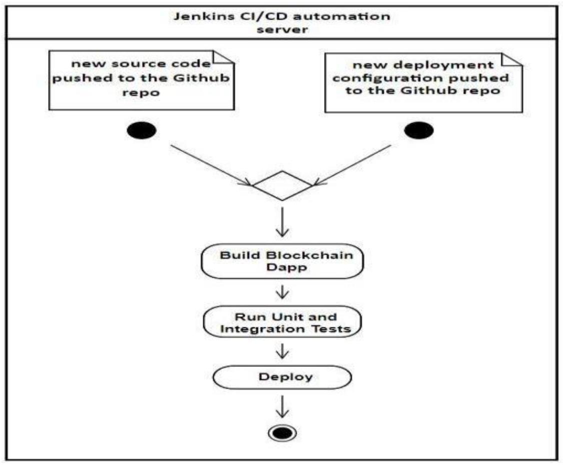

Jenkins CI/CD automation server
new source code pushed to the Github repo
new deployment configuration pushed to the Github repo
Build Blockchain Dapp
Run Unit and Integration Tests
Deploy

Fig. 12. The DevChainOps framework Jenkins pipeline.

Docker is widely used by organizations for standardizing build and test environments and deploying applications. The DevChainOps framework uses Docker containers for testing and deployment.

The Docker-compose file, located on GitHub, is crucial for the Jenkins server's pipeline execution process. It simplifies configurations of application service dependencies, allowing for specifying containers, networks, and persistent data volumes. The automated test suite is a key aspect of continuous deployment or integration, requiring an environment for testing.

V. Experimental Results

An integral element of this research involves verifying the accurate operation of the transformation process. As per the guidelines outlined in the IEEE 610.12-1990 Standard, validation refers to the assessment of a system or component either throughout or at the conclusion of its development phase, aimed at ascertaining its adherence to specified requirements and alignment with the intended application use.

Numerous test scenarios have been devised to verify the proper functionality of the smart contract application. The validation process consists of two types of tests: unit testing and integration testing. Unit testing focuses on discrete methods inside the smart contract application, whereas integration testing assesses overall operation across blockchain nodes, broadening its reach to check end-to-end scenarios.

These tests involve the configuration and operation of two or more blockchain nodes. Subsequently, transactions are executed and committed as part of the validation process. The test ensures that the ledgers of blockchain nodes accurately store the designated values. We have conducted both unit and integration tests for our three blockchain applications, namely Sawtooth xo integerkey, Quorum simple storage, and Corda tokens. As a result of the testing, the DevChainOps framework has proven to function effectively. Both the UML transformation and the Jenkins pipelines are working as intended.

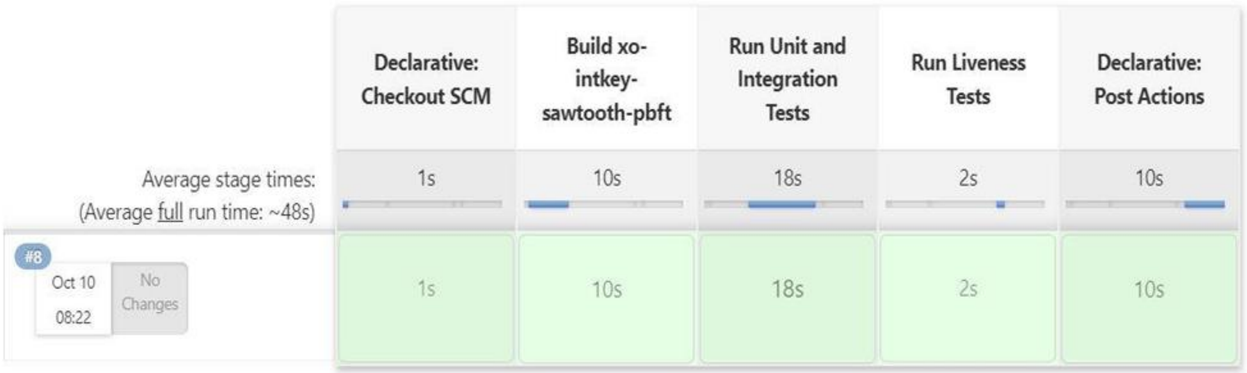
The Jenkins server displays a visual representation of every execution of the Groovy script in our configured pipelines. This visualization helps monitor each stage of the process and gather metrics, such as the average stage time. Both pipelines end with an additional stage called Declarative: Post Actions, which is closely linked with the chosen automation server on a technical level. After completing each pipeline, it is essential to perform a cleanup process within the workspace, including the removal of all temporary files and directories generated during execution Fig. 13 depicts metrics for the Sawtooth xo integerkey pipeline build execution on the Jenkins automation server. The pipeline's execution time is 41 seconds.

Fig. 14 depicts metrics for the Quorum simple storage pipeline build execution.

And Fig. 15 presents metrics for the Corda token pipeline. The pipeline's execution time is 42 seconds.

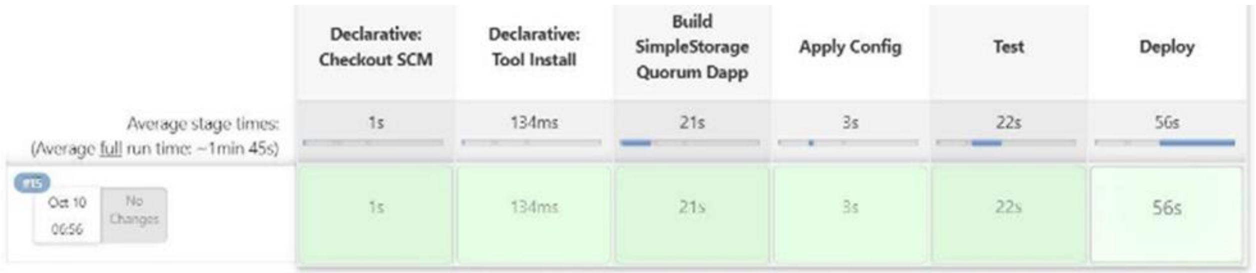
Comparison with the previous study:

In research [54] the authors introduced the BinCD framework, which aims to generate node deployment packages customized for the Corda blockchain platform.



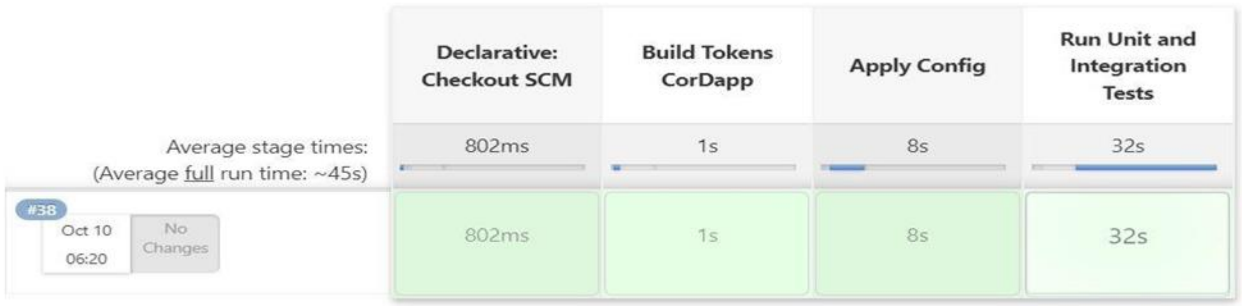
	Declarative: Checkout SCM	Build xo-intkey-sawtooth-pbft	Run Unit and Integration Tests	Run Liveness Tests	Declarative: Post Actions
Average stage times: (Average full run time: ~48s)	1s	10s	18s	2s	10s
#8 Oct 10 08:22 No Changes	1s	10s	18s	2s	10s

Fig. 13. Metrics of Sawtooth xo-intkey pipeline execution.



	Declarative: Checkout SCM	Declarative: Tool Install	Build SimpleStorage Quorum Dapp	Apply Config	Test	Deploy
Average stage times: (Average full run time: ~1min 45s)	1s	134ms	21s	3s	22s	56s
#15 Oct 10 06:56 No Changes	1s	134ms	21s	3s	22s	56s

Fig. 14. Metrics of Quorum simple storage dapp pipeline execution



	Declarative: Checkout SCM	Build Tokens CorDapp	Apply Config	Run Unit and Integration Tests
Average stage times: (Average full run time: ~45s)	802ms	1s	8s	32s
#38 Oct 10 06:20 No Changes	802ms	1s	8s	32s

Fig. 15. Metrics of Corda tokens dapp pipeline execution.

The BinCD framework is a continuous delivery method for producing packages for smart contract deployment on the Corda blockchain nodes. MDD is employed in generating the Platform Specific configuations tailored for the R3 Corda in version 4.6.

In this paper, we presented DevChainOps, an extensible framework designed to manage the entire CI/CD pipeline, focusing specifically on smart contracts. Notably, our framework includes a UML transformation component designed for adaptability, enabling smooth updates across various versions of blockchain platforms. Table III illustrates a comparison between the BinCD and DevChainOps frameworks.

TABLE III. PROVIDES A COMPARATIVE ANALYSIS BETWEEN THE BINCD AND THE DEVCHAINOPS FRAMEWORKS

Criteria	BinCD	DevChainOps
Blockchain platforms examined	R3 Corde 4.6	Hyperledger Sawtooth Quorum R3 Corda 4.8
DevOps phases implemented	Continuous Integration	DevOps phases implemented
Time complexity	O(n)	O(n)

DevChainOps encompasses three blockchain platforms: Quorum, Sawtooth, and R3 Corda 4.8.

We created a node config generator in Java that integrates with the Visual Paradigm Enterprise modeling framework using a dedicated Java plug-in. This method ensures that the config generator is independent of the modeling tool, enabling potential replacements with tools supporting UML extension mechanisms and providing APIs for accessing model content.

Our solution allows for managing models across various deployment environments within the UML deployment model, simplifying the generation of deployment configurations for blockchain networks in different settings. Additionally, by leveraging Java, our solution is compatible with specific Continuous Delivery Automation servers, enabling smooth integration of UML modeling support into CI/CD workflows.

The utilization of the simplicity architectural principle strives to attain a linear order-of-growth. In the running time of the node config generator. This hinges on two key factors: the execution time of individual statements and the frequency of their invocation. While the execution time is dictated by the environment, the invocation frequency is determined by the algorithm.

To minimize invocation frequency, the algorithm is developed using a straightforward and direct set of statements. One-dimensional dynamic collections, such as ArrayLists, are used to limit the number of repeating control structures to a single for loop. This is done by eliminating the usage of multidimensional data structures, the algorithm avoids nested repetition, thereby sidestepping quadratic, cubic, or exponential order-of-growth in running time.

The node config generator continues to operate efficiently, with a running time remaining under one second. Scaling up to larger networks warrants further performance analysis, though preliminary estimates suggest that generating configurations for 1000 deployment nodes could take around three minutes based on testing with a four-node Hyperledger Sawtooth xo-intkey blockchain network.

Memory usage is optimized through the utilization of local reference variables in Java, facilitating efficient garbage collection due to the restricted visibility of these objects.

The estimated size of the Java collection required for generating the deployment configuration for each node has been computed. This estimation factors in the memory usage of String objects, comprising 40 bytes for overhead, reference, hash, and padding, plus (2n + 24) bytes for a char array, where 'n' represents the character count in the string.

Assuming each tagged value contains 10 characters, an additional 8 bytes are used for the reference to the String object, resulting in a total size of 92 bytes per tagged value. For a Sawtooth node with 27 tagged values, the total size is calculated as 2484 bytes per node, with an additional 24 bytes for the collection object itself.

For a Quorum node with 87 tagged values, the size amounts to 8028 bytes per node, while for a Corda node with 107 tagged values, the size reaches 9868 bytes per node.

VI. CONCLUSION AND FUTURE WORK

DevOps, which has demonstrated success in traditional software development, holds significant promise for improving the process of developing blockchain smart contracts. This paper proposes a DevChainOps framework that adapts these principles to accommodate the unique features of smart contracts and integrates additional precautions. DevChainOps incorporates DevOps tools to facilitate the continuous integration and delivery of blockchain smart contracts.

The proposed framework generates comprehensive deployment packages for Sawtooth, Quorum, and R3 Corda blockchain platforms. The management of smart contract applications and deployment configuration files is conducted through GitHub repositories, employing version control. Integration with the Jenkins automation server has been achieved by making use of deployment configuration files and smart contract applications that are housed in GitHub repositories. In future works, there are plans to extend support to other blockchain platforms like HyperLedger Fabric. Additionally, efforts will be directed towards automating monitoring processes for smart contracts to minimize the necessity for manual intervention.

REFERENCES

[1] The Agile Manifesto. Principles behind the Agile Manifesto, 2001, [Online].Available : agilemanifesto.org/principles.html (accessed on 16 June 2024).

[2] Mikael Krief, Learning DevOps: A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins , Packt Publishing, 2022.

[3] De Kort, W. DevOps on the Microsoft Stack, 1st ed.; Apress Berkley: Berkeley, CA, USA, Volume 1;2016.

[4] Christopher Cowell; Nicholas Lotz; Chris Timberlake, Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples , Packt Publishing, 2023.

[5] J. Abou Jaoude and R. George Saade, "Blockchain Applications – Usage in Different Domains," in IEEE Access, vol. 7, pp. 45360-45381, 2019, doi: 10.1109/ACCESS.2019.2902501.

[6] Komalavalli, C., Saxena, D., & Laroiya, C. Overview of Blockchain Technology Concepts. Handbook of Research on Blockchain Technology, 349–371. 2020. doi:10.1016/b978-0-12-819816-2.00014-9

[7] V. Capocasale and G. Perboli, "Standardizing Smart Contracts," in IEEE Access, vol. 10, pp. 91203-91212, 2022, doi: 10.1109/ACCESS.2022.3202550.

[8] Zampetti, F.; Geremia, S.; Bavota, G.; Di Penta, M. CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution, Luxembourg, 27 September–1 October 2021.

[9] Chu, H., Zhang, P., Dong, H., Xiao, Y., Ji, S., & Li, W. A survey on smart contract vulnerabilities: Data sources, detection and repair. Information & Software Technology. 2023. https://doi.org/10.1016/j.infsof.2023.107221.

[10] Wöhrer, M.; Zdun, U. DevOps for Ethereum Blockchain Smart Contracts. In Proceedings of the 2021 IEEE International Conference on Blockchain (Blockchain), Melbourne, Australia, 3–8 December 2021; pp. 244–251.

[11] Al-Mazrouai, G.; Sudevan, S. Managing Blockchain Projects with Agile Methodology. In Proceedings of the 6th International Conference on Big Data and Cloud Computing Challenges, Kansas City, MO, USA, 9–10 September 2019; Vijayakumar, V., Neelanarayanan, V., Rao, P., Light, J., Eds.; Springer: Singapore, 2019; pp. 179–187.

[12] Marchesi, L.; Marchesi, M.; Tonelli, R. ABCDE—Agile block chain DApp engineering. Blockchain Res. Appl. 2020, 1, 100002.

[13] Lenarduzzi, V.; Lunesu, M.I.; Marchesi, M.; Tonelli, R. Blockchain Applications for Agile Methodologies. In Proceedings of the 19th International Conference on Agile Software Development, Companion, Association for Computing Machinery, Porto, Portugal, 21–25 May 2018.

[14] Abildskov, J. Collaboration in Git. In: Practical Git. Apress, Berkeley, CA. 2020. https://doi.org/10.1007/978-1-4842-6270-2

[15] Powell, R.; Stahnke, M. The 2020 State of Software Delivery, 2022, [Online].Available:circleci.com/resources/2020-state-of-software-delivery/

[16] A. S. Yaraghi, M. Bagherzadeh, N. Kahani and L. C. Briand, "Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts," in IEEE Transactions on Software Engineering, vol. 49, no. 4, pp. 1615-1639, 1 April 2023, doi: 10.1109/TSE.2022.3184842.

[17] Mahboob, J.; Coffman, J. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. In Proceedings of the IEEE 11th Annual Computing and Communication Workshop and Conference, Virtual, 27–30 January 2021.

[18] Dakkak, A., Bosch, J., Olsson, H. H., & Mattos, D. I. Continuous deployment in software-intensive system-of-systems. Information & Software Technology. 2023. https://doi.org/10.1016/j.infsof.2023.107200.

[19] E. Zaghloul, T. Li, M. W. Mutka and J. Ren, "Bitcoin and Blockchain: Security and Privacy," in IEEE Internet of Things Journal, vol. 7, no. 10, pp. 10288-10313, Oct. 2020, doi: 10.1109/JIOT.2020.3004273.

[20] Brandstatter, T.; Schulte, S.; Cito, J.; Borkowski, M. Characterizing Efficiency Optimizations in Solidity Smart Contracts. In Proceedings of the IEEE International Conference on Blockchain, Toronto, ON, Canada, 3–6 May 2020.

[21] Murugan, S.; Kris, S. A Survey on Smart Contract Platforms and Features. In Proceedings of the 7th International Conference on Advanced Computing and Communication Systems, Coimbatore, India, 19–20 May 2021.

[22] Oyinloye, D.P.; Damilare, P.; Teh, J.S.; Jamil, N.; Moatsum, A. Blockchain Consensus: An Overview of Alternative Protocols. Symmetry 2021, 13, 1363.

[23] Ma, J.; Jo, Y.; Park, C. PeerBFT: Making Hyperledger Fabric's Ordering Service Withstand Byzantine Faults. IEEE Access 2020, 8, 217255–217267.

[24] T. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," *Journal of Network and Computer Applications*, vol. 177, p. 102857, Mar. 2021, doi: 10.1016/j.jnca.2020.102857.

[25] M. Mazzoni, A. Corradi, and V. Di Nicola, "Performance evaluation of permissioned blockchains for financial applications: The ConsenSys Quorum case study," *Blockchain. Research and Applications*, vol. 3, no. 1, p. 100026, Mar. 2022, doi: 10.1016/j.bcra.2021.100026.

[26] A. Baliga, I. Subhod, P. Kamat, and S. Chatterjee, "Performance Evaluation of the Quorum Blockchain Platform," *arXiv.org*, Jul. 19, 2018. https://arxiv.org/abs/1809.03421

[27] Y. Sharma, "Blockchain and Distributed Ledger System," in *CRC Press eBooks*, 2020, pp. 177–206. doi: 10.1201/9780429352546-8.

[28] P. Moriggl, P. M. Asprion, and B. Schneider, "Blockchain Technologies Towards Data Privacy—Hyperledger Sawtooth as Unit of Analysis," in *Studies in systems, decision and control*, 2020, pp. 299–313. doi: 10.1007/978-3-030-48332-6_20.

[29] A. Castro Jiménez, "Development of a distributed application over R3 Corda," Treball Final de Grau, UPC, Escola TècnOUica Superior d'Enginyeria de Telecomunicació de Barcelona, Departament d'Enginyeria Telemàtica, 2021.

[30] Zou, W.; Lo, D.; Kochhar, P.S.; Le, X.D.; Xia, X.; Feng, Y.; Chen, Z.; Xu, B. Smart Contract Development: Challenges and Opportunities. IEEE Trans. Softw. Eng. 2021, 47, 2084–2106.

[31] Laukkanen, E.; Itkonen, J.; Lassenius, C. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. Inf. Softw. Technol. 2017, 82, 55–79.

[32] Abdalkareem, R.; Mujahid, S.; Shihab, E.; Rilling, J. Which Commits Can Be CI Skipped? IEEE Trans. Softw. Eng. 2021, 47, 448–463.

[33] Gallaba, K.; McIntosh, S. Use and Misuse of Continuous Integration Features: An Empirical Study of Projects That (Mis)Use Travis CI. IEEE Trans. Softw. Eng. 2020, 46, 33–50.

[34] Saidani, I.; Ouni, A.; Mkaouer, M.W.; Palomba, F. On the impact of Continuous Integration on refactoring practice: An exploratory study on TravisTorrent. Inf. Softw. Technol. 2021, 138, 106618.

[35] Couto, L.D., Tran-Jørgensen, P.W.V., Nilsson, R.S. et al. Enabling continuous integration in a formal methods setting. Int J Softw Tools Technol Transfer 22, 667–683 2020. https://doi.org/10.1007/s10009-019-00546-y.

[36] Mysari, S.; Bejgam, V. Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible. In Proceedings of the 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 24–25 February 2020; pp. 1–4.

[37] Yu, L.; Alégroth, E.; Chatzipetrou, P.; Gorschek, T. Utilising CI environment for efficient and effective testing of NFRs. Inf. Softw. Technol. 2020, 117, 106199.

[38] Leite, L.; Pinto, G.; Kon, F.; Meirelles, P. The organization of software teams in the quest for continuous delivery: A grounded theory approach. Inf. Softw. Technol. 2021, 139, 106672.

[39] Chowdhury, M.J.M.; Ferdous, M.S.; Biswas, K.; Chowdhury, N.; Kayes, A.S.M.; Alazab, M.; Watters, P. A Comparative Analysis of Distributed Ledger Technology Platforms. IEEE Access 2019, 7, 167930–167943.

[40] Jamil, F.; Iqbal, N.; Imran; Ahmad, S.; Kim, D. Peer-to-Peer Energy Trading Mechanism Based on Blockchain and Machine Learning for Sustainable Electrical Power Supply in Smart Grid. IEEE Access 2021, 9, 39193–39217.

[41] Saxena, S.; Farag, H.E.Z.; Brookson, A.; Turesson, H.; Kim, H. A Permissioned Blockchain System to Reduce Peak Demand in Residential Communities via Energy Trading: A Real-World Case Study. IEEE Access 2021, 9, 5517–5530.

[42] Monrat, A.A.; Schelén, O.; Andersson, K. A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities. IEEE Access 2019, 7, 117134–117151.

[43] Al-Jaroodi, J.; Mohamed, N. Blockchain in Industries: A Survey. IEEE Access 2019, 7, 36500–36515.

[44] Ismail, L.; Materwala, H.; Zeadally, S. Lightweight Blockchain for Healthcare. IEEE Access 2019, 7, 149935–149951.

[45] Shahnaz, A.; Qamar, U.; Khalid, A. Using Blockchain for Electronic Health Records. IEEE Access 2019, 7, 147782–147795.

[46] Chavez, H.M.; Shen, W.; France, R.B.; Mechling, B.A.; Li, G. An Approach to Checking Consistency between UML Class Model and Its Java Implementation. IEEE Trans. Softw. Eng. 2016, 42, 322–344.

[47] Lu, H.; Wang, S.; Yue, T.; Ali, S.; Nygård, J.F. Automated Refactoring of OCL Constraints with Search. IEEE Trans. Softw. Eng. 2019, 45, 148–170.

[48] Arora, V.; Singh, M.; Bhatia, R. Orientation-based Ant colony algorithm for synthesizing the test scenarios in UML activity diagram. Inf. Softw. Technol. 2020, 123, 106292.

[49] Assunção, W.K.G.; Vergilio, S.R.; Lopez-Herrejon, R.E. Automatic extraction of product line architecture and feature models from UML class diagram variants. Inf. Softw. Technol. 2020, 117, 106198.

[50] Arcaini, P.; Gargantini, A.; Riccobene, E. Decomposition-Based Approach for Model-Based Test Generation. IEEE Trans. Softw. Eng. 2019, 45, 507–520.

[51] Moradi, H.; Zamani, B.; Zamanifar, K. CaaSSET: A Framework for Model-Driven Development of Context as a Service. Future Gener. Comput. Syst. 2020, 105, 61–95.

[52] Panach, J.I.; Dieste, Ó.; Marín, B.; España, S.; Vegas, S.; Pastor, Ó.; Juristo, N. Evaluating Model-Driven Development Claims with Respect to Quality: A Family of Experiments. IEEE Trans. Softw. Eng. 2021, 47, 130–145.

[53] Basciani, F.; D'Emidio, M.; Ruscio, D.D.; Frigioni, D.; Iovino, L.; Pierantonio, A. Automated Selection of Optimal Model Transformation Chains via Shortest-Path Algorithms. IEEE Trans. Softw. Eng. 2020, 46, 251–279.

[54] Górski, T. Continuous Delivery of Blockchain Distributed Applications. Sensors 2022, 22, 128.

[55] Xu, X.; Weber, I.; Staples, M. Architecture for Blockchain Applications; Springer: Cham, Switzerland, 2019; pp. 5–7.

[56] Gao, Z.; Jiang, L.; Xia, X.; Lo, D.; Grundy, J. Checking Smart Contracts with Structural Code Embedding. IEEE Trans. Softw. Eng. 2020, 47, 2874–2891.

[57] Laukkanen, E.; Itkonen, J.; Lassenius, C. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. Inf. Softw. Technol. 2017, 82, 55–79.

[58] UML Profile for Distributed Ledger, 2023, [Online].Available at https://github.com/ramadannasr/UML-PROFILE-FOR-DLT.git

[59] Node Config Generator, 2023, [Online].Available at https://github.com/ramadannasr/Node-config-generator.git

[60] Sawtooth xo intkey dapp , 2023, [Online].Available athttps://github.com/ramadannasr/xo-intkey-sawtooth-pbft.git

[61] Quorum simple\ storage dapp , 2023, [Online].Available at https://github.com/ramadannasr/SimpleStorageQuorumDapp.git

[62] Corda tokens dapp , 2023, [Online].Available at https://github.com/ramadannasr/TokensCorDapp.git.