Chapter 4. **Research Methodology**

4.1 Introduction

This chapter focuses on challenges related to design contributions and research methodology. The blockchain technology is utilised in the proposed system in an effort to achieve more transparency and reliability in software business. Instead of depending on tools like continuous integration servers and automated test suites to assess whether or not code is properly progressing through the pipeline, data is recorded on a distributed ledger known as a blockchain. Blockchain technology is normally very high in disruptive potential and effectively works without an intermediary. Through this system, we have tried to strengthen DevOps with blockchain for all software enterprises. Source code (coding repositories), artefacts (continuous integration and delivery, automation), documents (configuration, description ,definition), and containers (layer ,images) are frequently dispersed across several departments and are rarely accessible in a single location. The fundamental inefficiencies of DevOps can be minized by integrating all stakeholders for a piece of documentation onto a single peer-to-peer, decentralised network and instituting transparency so that every change is documented on this single platform.

The procedures of coding, building, testing, releasing, deploying, operating, monitoring, and planning are all part of the DevOps methodology. Continuous development, integration, testing, continuous monitoring, and continuous feedback are some of the processes that make up the DevOps lifecycle. A conceptual solution for the implementation of the DevOps auditable change control process using a private blockchain is shown in figure 4.1. The proposed framework uses blockchain to bring together several stages of DevOps into a single application. Automation is a huge part of DevOps culture. DevOps contains different tools in its lifecycle. DevOps tools allow enterprise developers to run fully automated DevOps pipelines from code to production. Blockchain technology will transform the way business is conducted without disturbing its ongoing automated processes. All DevOps stakeholders can access the blockchain network through DevOps tools which use decentralized databases for storing data files.

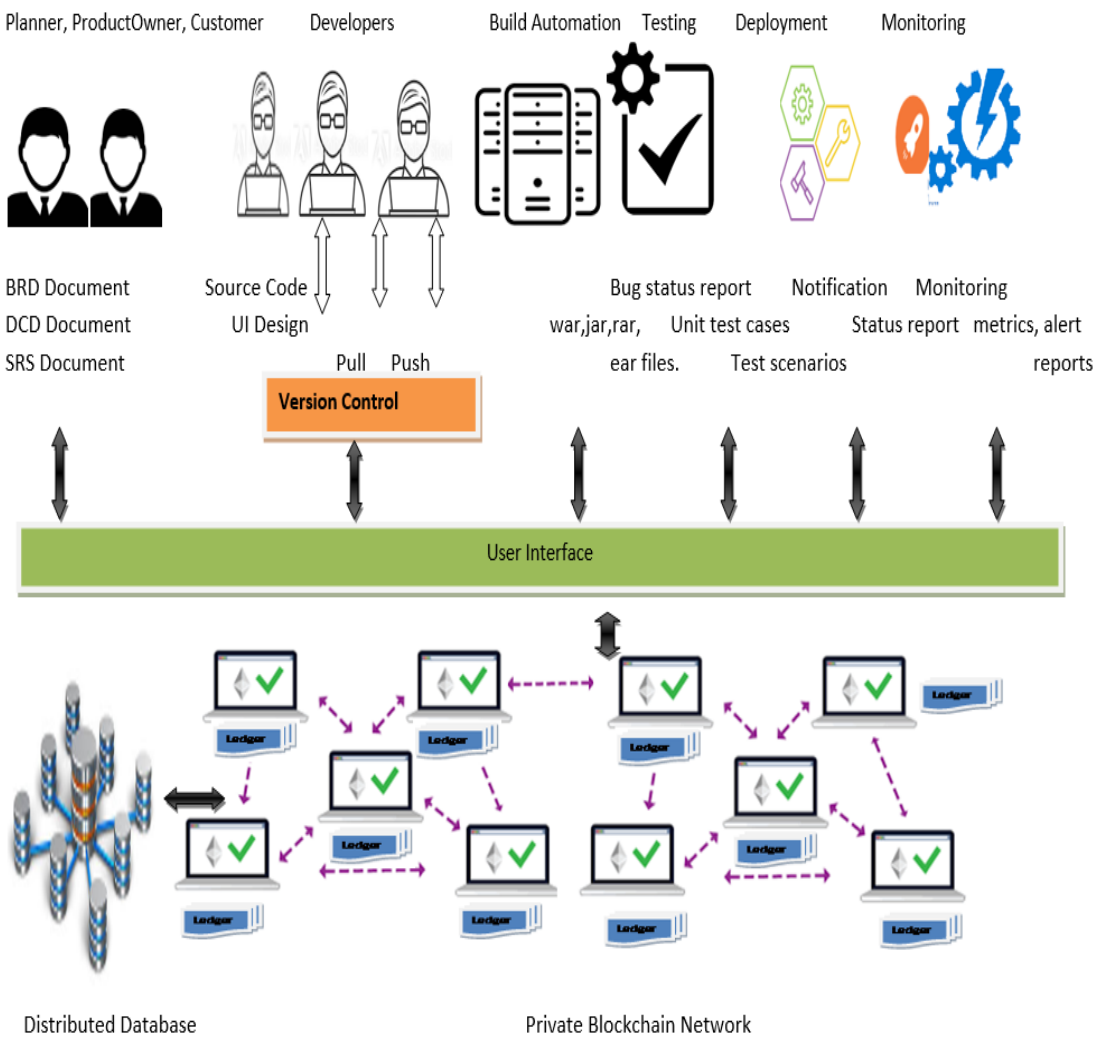


Figure 4.1: DevOps with blockchain

By incorporating blockchain, the proposed DevOpsChain system has become transparent, scalable, and secure for managing DevOps for geographically dispersed clients and development teams. The DevOpsChain framework is designed to adhere to the DevOps software development life cycle and incorporates a private blockchain for the purpose of managing the transactions associated with DevOps in an effective manner. These transactions include posts, customer quick feedback, and acceptance testing. Blockchain is an appropriate platform for the creation of software in a distributed environment due to its distributed ledger, distributed database, high availability, and high security qualities.

4.2 Architecture Overview

The first fold of our research is to create a blockchain-based framework named as DevOpsChain, which ensures the traceability of different operational modes. The framework helps to track and trace every mode of operations of artifacts. The architecture ought to have two major modules, such as development and deployment. We concentrated on the development of systems for the verification of DevOps transactions, the efficiency of information retrieval, and the fault-tolerant mechanism of queries. Because of the environment of DevOps, public blockchains may not be a good fit. For example, a DevOps may not be able to access all of the blockchain's material for privacy reasons. It's also feasible that users seek more control over the operation of the blockchain, which is impossible with open public blockchains. As a result, businesses may have to put up their own private blockchain. A private blockchain framework Hyperledger Fabric, which is part of The Linux Foundation's Hyperledger efforts.

The overall architecture of the private blockchain solution, includes the different components, Decentralized Database (IPFS), and the private blockchain network (Hyperledger Fabric). The Blockchain network connects all software stakeholders, including managers, administrators, testers, and clients, to their DevOps phase. It depicts a decentralized system with a new intermediary that would deliver software development documents with an increase in audibility and availability. The concept, operational processes, and user experience have no bearing on the underlying blockchain architecture. The purpose of this project is to support all DevOps operations without interfering with the automation in any way.

It is proposed to implement blockchain for transparency so that all transactions and alterations to various files are available to all stakeholders and their history is documented on immutable storage. Only transactions that meet the chaincode's set of restrictions can be accepted onto the blockchain ledger. By incorporating blockchain into the system, all modifications made in the DevOps environment are registered into the blockchain, allowing all stakeholders to check the changes that have been made to the record. In Hyperledger Fabric, users must build the consensus algorithm, chaincode in Go or Node.JS, to allow each transaction to be accepted into the blockchain.

Hyperledger Composer: It is an abstraction of Hyperledger Fabric that enables flexibility in business logic development and support for several connectors, such as REST-API, for the

back-end interface of a business network. This proposed system uses Hyperledger Composer with Hyperledger Fabric Blockchain which provides an application development framework that simplifies and expedites the creation of hyperledger fabric blockchain applications. It uses a generic chaincode that maps composer models to chain code tables and runs transaction processor functions in the javascript interpreter in a javascript engine. So that it runs in its own runtime environment and chaincode container.

Hyperledger Fabric significantly relies on the chaincode smart contract engine, whereby each network peer executes in Docker containers. Orderer and endorsers are important types of nodes that are responsible for transaction validation.

Orderer peers: Orderers receive endorsed transactions from the client SDK and package them into blocks as specified in your configuration file before sending them to all other peers so that they can verify the transaction and update their ledgers. Ordering Peers are an unique sort of node. They receive endorsed transactions from the client SDK and package them into blocks as specified in your configuration file before sending them to all other peers so that they can verify the transaction and update their ledgers.

Endorser peer: Endorser peers are nodes that validate the transaction, run the chain code, and simulate the transaction's conclusion. When they receive a transaction proposal for endorsement, they respond by granting or denying endorsement.

Hyperledger Fabric's first three consensus steps are endorsement, ordering, and validation. The policy (m out of n signatures) on which a transaction is based determines whether or not participants endorse it. After obtaining the endorsed transaction from the ordering phase, the orderer will be committed to the ledger. A block of ordered transactions is validated to ensure its accuracy through the process of validation. A public key infrastructure is used by the hyperledger fabric to generate cryptographic certificates. The Membership Service Provider (MSP) is responsible for developing digital identities for the organization's members and customers. A MSP evaluates whether Certification Authorities (CAs) may be trusted in order to identify the domain members (member, admin, and so on).

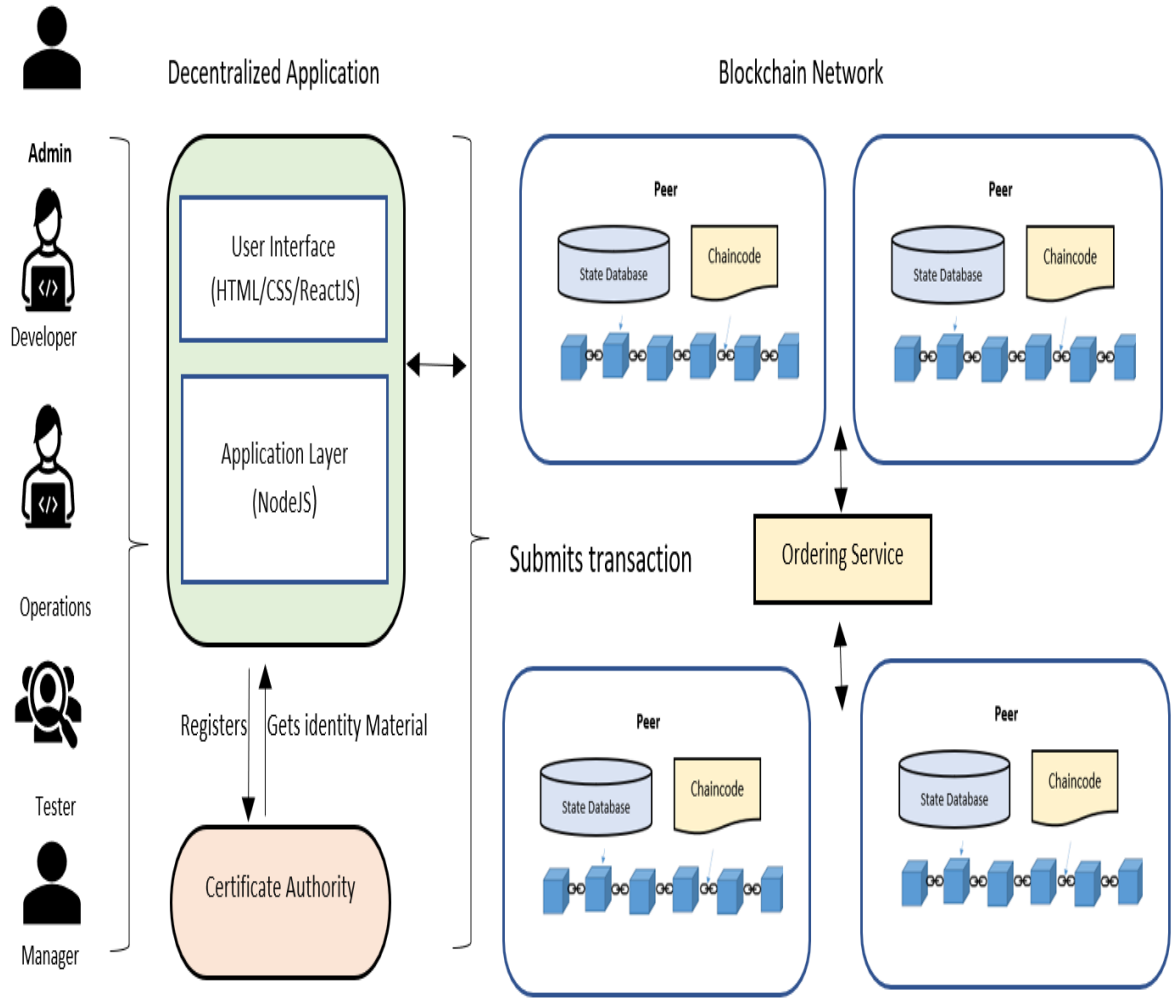


Figure 4.2: Technology stack

The framework shown in figure 4.2 gives a clear insight into a proposed DevOpsChain system using Blockchain technology which shows technologies used for implementation and different components of the proposed framework. DevOpsChain methodology is divided into two major tasks: Development and Deployment.

4.3 Continuous Development

This phase involves the tasks initiated from the project proposal. Client interaction with the project manager is initiated when clients come to the organization with their project proposal to get their work done. The client interface keeps records of all contact between the client and the project manager so that no one can back out of what they have agreed on. Because the client is a critical stakeholder, the interface also functions as a medium for informing them of project developments. The interface allows users to share files or other types of attachments with one

another. When the client invokes any service, the corresponding fabric function will be called. The sequence of fabric functions is shown in figure 4.3 which provides a history of functions called.

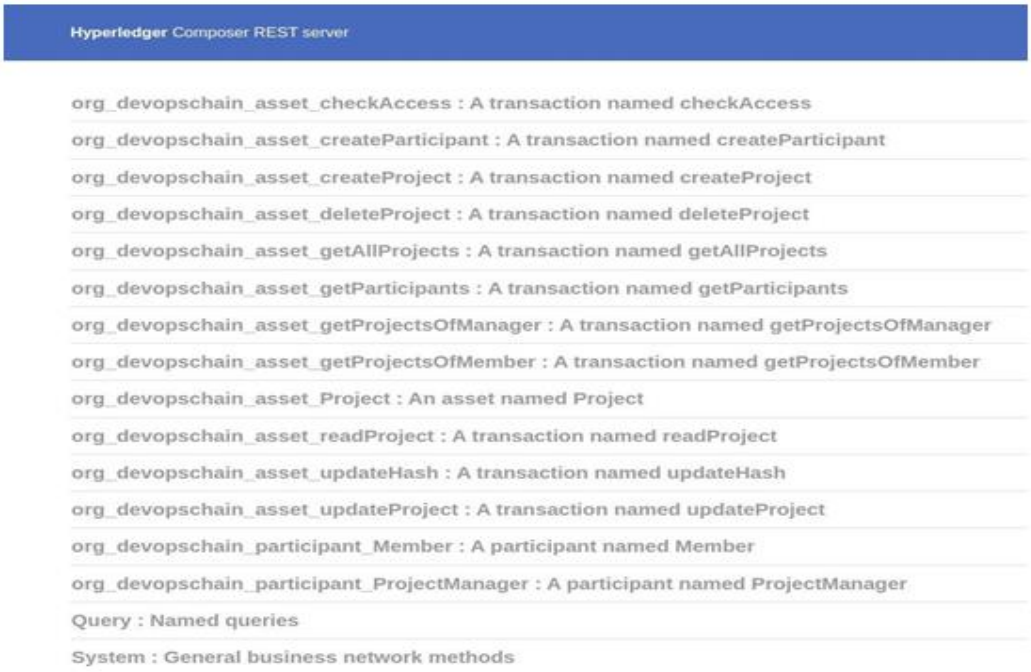
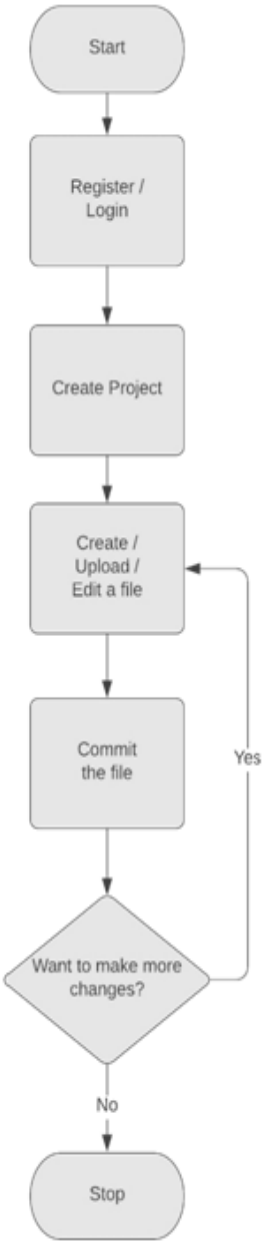


Figure 4.3: Fabric transactions

Project Initiation: Following a successful interaction, an organization employee will initiate a project with a distinct project id. The employee's credentials, i.e. their card, are checked for access at this point. If the card has been tampered with, the blockchain will provide an error and the user will be denied access. If the user is authentic, project info is stored in the blockchain's state database. The IPFS database is then used to store a project's git repository. After that, the project owner can invite contributors to the project repository. In this system, there are two repo access options, a project can be either kept public or private, depending on the project's needs. CRUD activities can only be performed by the project's collaborators. If the project's owner makes it private, project will not be visible to non-collaborators of the repo in the organisation. On the other hand, if the project is kept public then the non-collaborators can see, download, and clone the repo but they can’t perform the CRUD operations on the project conceptual solution. Following diagrams 4.4 shows the processes for versioning of files and merging of branches.



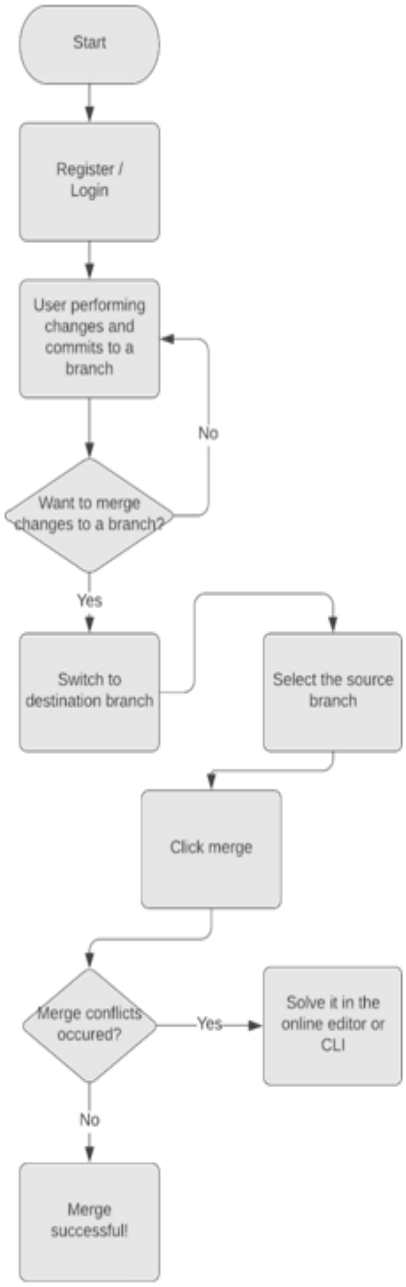


Figure 4.4(a): Versioning of files workflow Figure 4.4(b): Merging of branches workflow

Project Access Management: Every time a user requests any project then he is validated whether he is the collaborator of the project or not. All of the malicious activity that has occurred has been documented in the transaction history of the blockchain. Administrations have complete control over whether or not an employee is allowed to remain in the system, and they can remove anyone they choose.

Blockchain to IPFS Communication: The blockchain first authenticates and authorises the user, and if the user is valid, the request is sent to the IPFS server with an authorization token. For authentication at the IPFS server, the private key shared by the blockchain authentication server is used. If the token is valid then the intended operation on the project/git repository is executed otherwise the request is denied. After the IPFS transaction is completed, a new project hash is generated and sent to the authentication server, which updates the blockchain's state database with the new project hash.

Enabling all Git operations: Current version control systems are either centralized or distributed. In the case of distributed version control also, rights are centralized at one entity. As shown in figure 4.5, the proposed system presents blockchain-based version control, here every update or a new file is stored and updates are tracked on immutable storage.

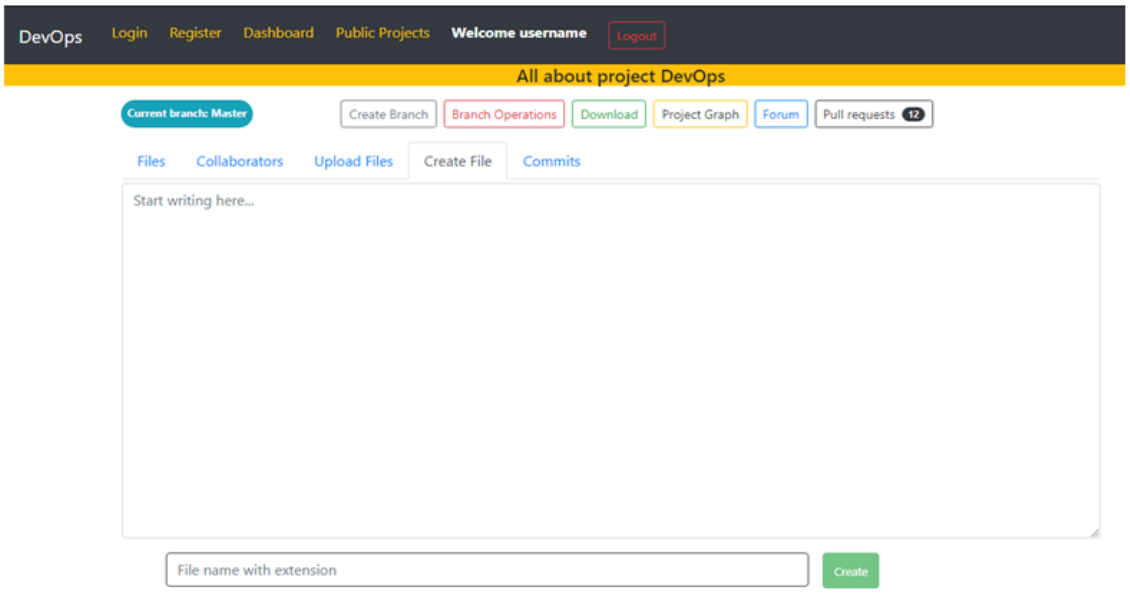


Figure 4.5: Specific project page to create a file

The IPFS server runs the git operations, and the repository is updated across all clusters as a result of each operation. As a result, multiple versions of the repository are kept up to date. Git features such as adding and committing are supported by this system, as well as the ability to merge and maintain a commit history. Moreover, merge conflicts can be resolved using the web UI or the CLI. The user has also indicated whether his branch is behind or ahead of the main branch so that he remains updated with the master. In case of any repository loss, while executing the operations occurs, the repository can be made available by the IPFS clusters.

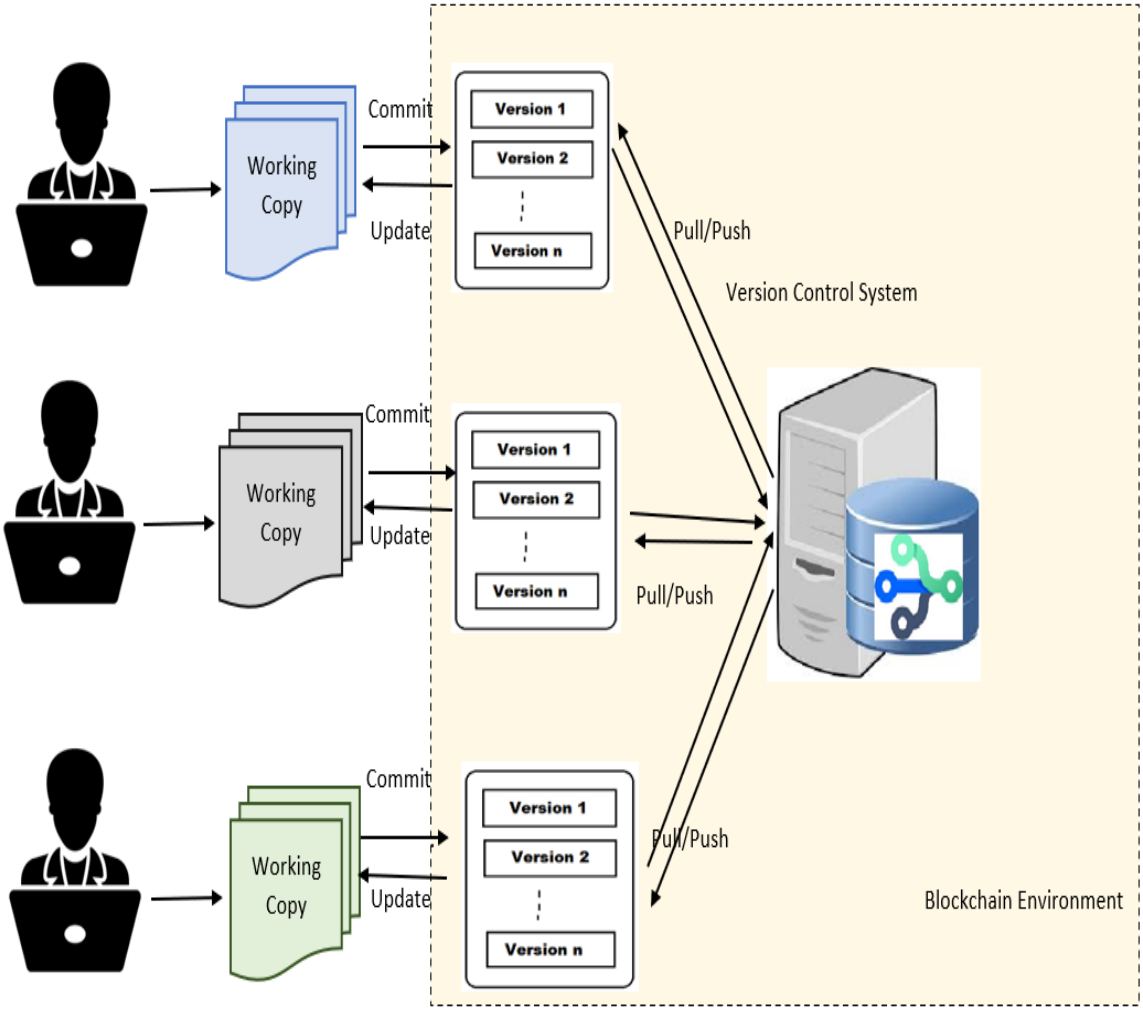


Figure 4.6: Blockchain-based version control

User is allowed to upload files via CLI interface or GUI interface. Once the file is created, the blockchain function is called and this gets stored in the form of blockchain transactions.

Developing Custom Cli: The git operations on the remote repository are performed using a custom CLI is developed. Above figure 4.6 shows the detailed architecture of blockchain based version control. It is not possible to use the official git CLI with the blockchain's authentication system in most cases. After the user authenticates with the blockchain's authentication server, they can then perform operations on the remote repository. The CLI manages its session using a keychain mechanism which is used by the windows or mac/ubuntu systems to manage the accounts and their credentials. In order to compile and generate an executable version of your project, you must use a process known as Continuous Integration (CI). CI is triggered when you send your production-ready code to a piece of software that has been given specific steps and procedures to follow when receiving the trigger. The build server is triggered to compile, run,

and inspect code changes when a git repository is used. An alarm is sent to the developers if the automatic check cases fail due to code modifications that were made. Alternatively, the developers can roll back their changes to the previous working version of the code, or they can go forward and include a fix for the failing code. The Jenkins integration server is used in this system. To make it easier for developers to incorporate changes into the project and to make it easier for users to obtain a fresh build, the proposed system uses Jenkins.

Continuous Integration and Continuous Deployment (CI / CD) is a better option for large enterprises than the traditional waterfall approach, which can take weeks or months to deliver a product following integration and testing.

Role of IPFS: Storage of files on IPFS nodes within a network or cluster of nodes within a network is made possible by the IPFS backend file store, which eliminates latency caused by accessing files hosted on distant servers. It is not necessary to go to the central server even if a node or peer located next to us has the file you seek. The distributed nature of the network allows it to do so. Each node can communicate with and store files on the other nodes. All of the nodes' data is stored on these servers. In IPFS, files can be accessed via the metadata hash. A "multi hash" is the technical term for this type of hash. As a result, a content-based addressing mechanism, rather than an IP address-based one, is used to locate a file or folder in the network and retrieve it more quickly than a server's response time would be. The integrity of the uploaded file can be preserved by relying on the hash of the file/folder. A new hash of the file is generated whenever a file is modified, which necessitates a re-upload of the file and the generation of a new reference to access the modified file.

IPFS will house all of the artefacts generated by a version control system, build, or testing tool. With the help of the hash of the file, we can fetch files from an IPFS node that is close to us. This eliminates the need for URL-based addressing mechanisms if cloud-based DevOps pipeline solutions are implemented by an organisation. IPFS will house all of the artefacts generated by a tool's operations, whether they are uploaded or not. IPFS makes use of content-based addressing and lets us retrieve files from a nearby IPFS node using a file hash. If a company decides to use cloud-based DevOps pipeline solutions, this eliminates the need to rely on URL-based addressing mechanisms.

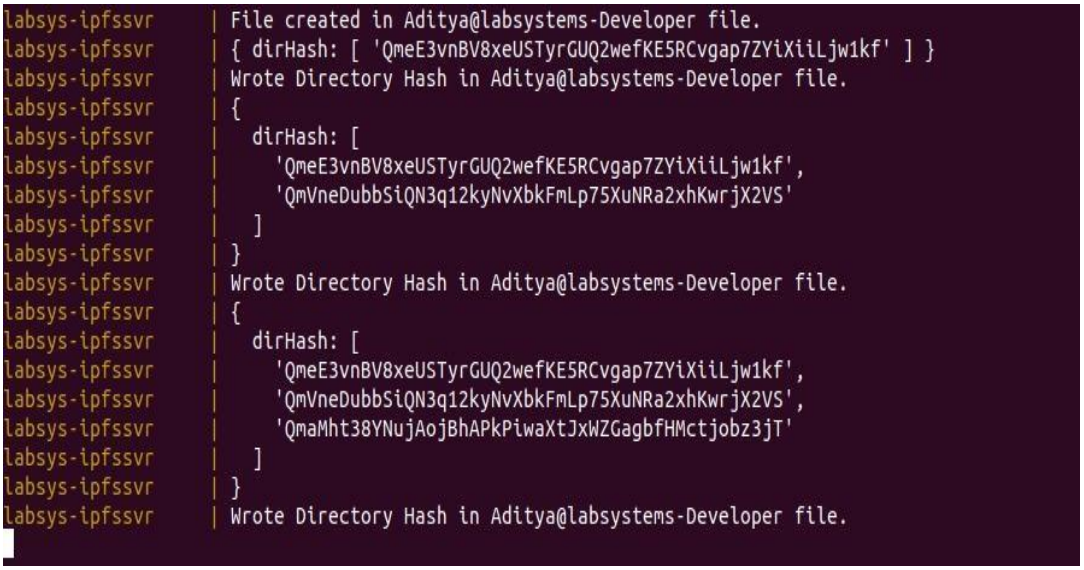


Figure 4.7: IPFS content-based addressing

Figure 4.7 depicts how this work keeps the hashes of several folders within a JSON file. Since each uploaded material is saved in a directory named 'DocumentsDir,'. It is possible to keep track of the many versions, regardless of whether it is a single file or a group of files. The 'dirHash' JSON array contains many iterations of the directory hash when files inside the directory are updated or new files are added.

The files or artefacts are saved as 256 kb-sized chunks. It then uses a Linked List-based reference method to link sections of a rather big file. This means that users do not need to worry about the privacy of their project documents.

4.4 Continuous Delivery and Deployment

In the Continuous Delivery / Deployment process (CD), stakeholders will be able to completely automate the deployment procedures to the point just before deploying to production or to the point where the production server is fully automated. A production server is a server where our project will be hosted, and clients can access the project by the provided URL. The whole idea of CD is to make sure that the project can sustain its workflow in different deployment environments and is not affected by any issues that disrupt its work methodology. CD can be performed with the help of various configuration scripts which are stored in Software Configuration Management (SCM) repositories. Software Configuration Management, where all the essential configuration about the number of nodes to host replicas, orchestration of nodes,

along with other reliability and scalability parameters are being stored in a version-controlled environment.

Today organizations are required to deliver greater levels of customer satisfaction through their online services. However, many are forced to support these initiatives with an interrupt-driven approach as they react to repair things after they break. However, for a more proactive approach and to manage expected high levels of Service Level Agreements (SLAs), organizations can prefer to reduce their amount of unscheduled downtime by implementing the Continuous Delivery (CD) model. In DevOps, an organization's pipeline is filled with a wide variety of tools and processes. The management of these tools relies heavily on the ability to monitor them. It is important for an organisation to be prepared for the various problems that can arise at each stage of the pipeline when it implements a Continuous Integration / Continuous Deployment model for its software engineering process. Companies cannot afford to spend time looking after the tools when they got their product deployment to worry about. A Continuous Monitoring solution is presented as a better solution to solve the issue of having to track numerous microservices from previous stages. As shown in the figure 4.8, the proposed methodology uses the Jenkins integration server.

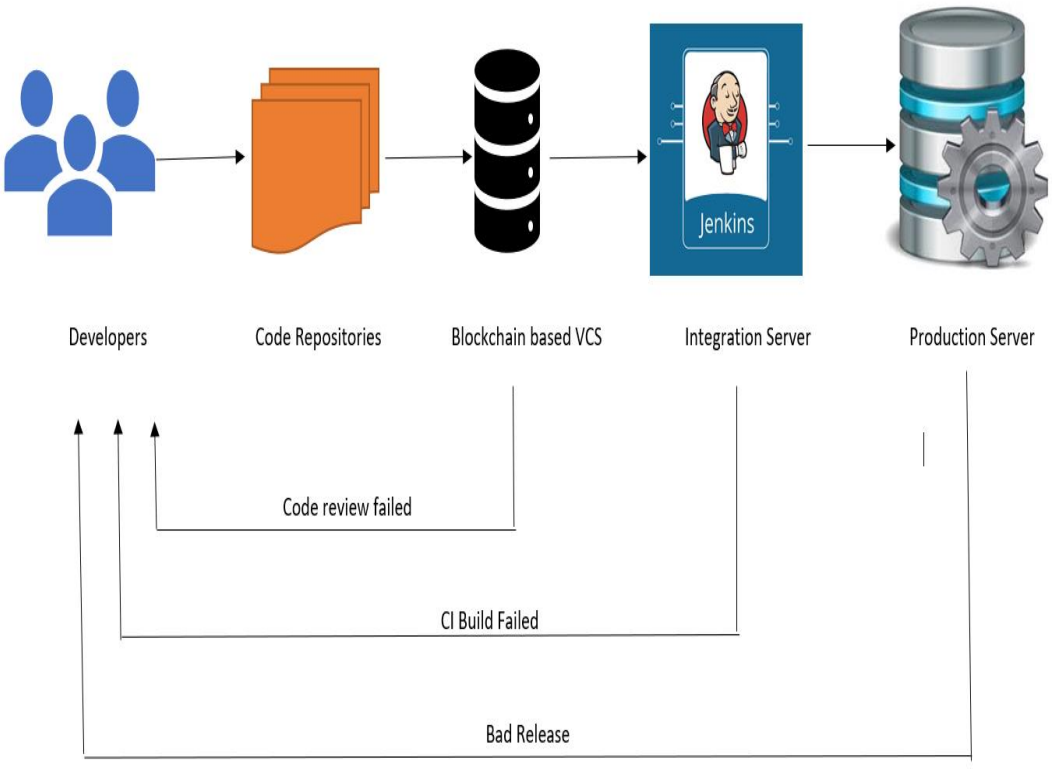


Figure 4.8: Architecture for continuous delivery/deployment

Jenkins: Jenkins is a continuous integration tool that can use preinstalled (or user-installed) plugins and/or a script file that defines all the commands and procedures it must execute to carry out the process of building a project and ensuring that testing and integration have been performed correctly each time.

Jenkins is configured by a DevOps engineer, by communicating with the developers of a project to make sure that it can build the same technology stack that has been developed. Jenkins is an open-source project itself and it has a very engaging community that is capable of building and shipping its programmed plugins to support more and more variety of technologies that are coming into the market. These plugins are developed using the Java programming language and Maven technology since Maven allows you to package your Java code into a .hpi file, after which Jenkins will allow your plugin to be uploaded. This process itself is also carried out in Jenkins as a Job.

Every programming language has its special build tool which allows users to automatically produce executables of their project. This is very useful when there is a need to integrate multiple modules and then collectively form an executable since it will be a waste of developers' precious time to spend integrating every module by themselves and creating an executable that works.

Every build task in Jenkins is termed as a 'Job'. Each Job is run on the Jenkins CI server with the help of either a script file (particularly in the case of pipeline jobs – the script is called Jenkins file) or with the help of commands defined in the Job settings which is shown in figure 4.9). Jobs are running on single or multiple agents which can be either a node identified with the help of a label that is preconfigured in Jenkins agents set up or it can even be a dockerized environment. The idea is to build the project and make sure that a proper executable of the same comes out from a successful job.

Figure 4.9: Jenkins file

Docker: Docker is a technology that can encapsulate a confined and configured environment in the form of containers and this ability allows operation specialists to deploy projects at any server regardless of the environment with the only requirement of having Docker CE (Community Edition) installed. There was a problem of having to shift their codebase (projects as a whole) from one server to another where the installed dependencies cannot be moved because of their huge sizes. So as projects and ideas grew in size, eventually many organizations required a solution to this problem as it was becoming more common in a lot of places. Since this was a problem essentially regarding portability, Virtual Machines and Virtual Images of the project installed on a Virtual Server could be a possible solution. But because of large projects requiring the large number of modules and dependencies installed, the overhead of having to manage storage for even the base operating system such as Linux distributions or Windows complete environment was unnecessary. The requirement was to manage portability for only the project and because of having portable ‘VMs’, there were huge storage management issues as undesired aspects of an environment were also brought along with the project. So overall, there was a need to solve this problem of portability but also avoid any storage overhead due to unnecessary elements of being packaged.

The containerization technology – Docker, gave the ability to package only the required elements of an environment and not every single thing available on a huge Operating System. Docker is open-source software and it has a community that is very engaged in developing tons of docker images and making them as slimmer as possible so that they can be similarly reused by other developers. Docker Hub is a place where developers host their custom Docker images, as well as the place, also consists of official Docker images. A Dockerfile is a text file that contains all of the instructions a user may use to create an image. Using docker build, users may generate an automated build that sequentially runs several command-line instructions. The proposed system uses docker files from which Docker images can be created.

Continuous monitoring of performance parameters to be done to decide what action should be taken to improve and promote it at various levels. Monitoring reports are generated and stored in DevOpsChain and are available to all stakeholders. Developers will write or updates code in the DevOps blockchain ledger, and code is merged into the blockchain ledger main branch repository. Once the change will be noticed by the integration server, which will then run the test suite. There will be two servers for each build instance: one for staging and one for production. Code is tested thoroughly on the staging server. If both the support team and the client are happy with the results, the support team lead will work with the client to set up a time when the production server can be updated. When that happens, customers will be able to use the new feature. The overall procedure for continuous delivery/deployment is shown in the figure 4.10.

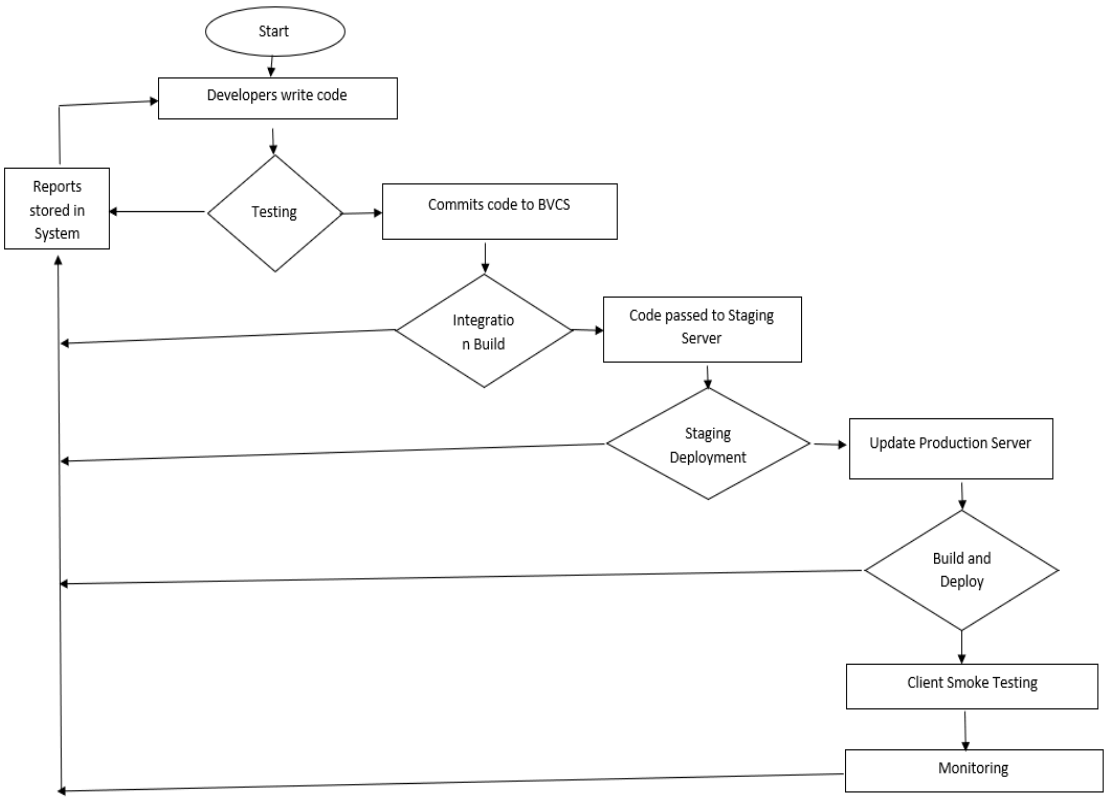


Figure 4.10: Working of Continuous Delivery/Deployment

In this phase, smart contracts are used for continuous delivery. A smart contract makes sure that code changes get built, tested, and ready for a production release as part of the Continuous Delivery process. Here, when developers make many commits throughout the day, our automated build scripts will notice changes in blockchain Source Code Management (SCM) systems like Git. Once the change is found, the source code would be sent to a dedicated build server to make sure the build isn't failing and that all test cases and integration tests are running well. The build application is then put on the test servers (pre-production servers) for the User Acceptance Test (UAT). Lastly, the application is put on the production servers by hand before it is released.

Following section discusses the algorithms algorithms used in the proposed system. Algorithm 1 describes the users Admin and Participant Enrolment while Algorithm 2 shows the process to create Report. Algorithm 3 shows procedure to create Project in the system and algorithm 4 shows the process to update document.

Algorithm 1: Admin and Participant Enrolment

Stakeholders S_T : Manager M_N, Developer D_N,Project Regitsry P_{RG}

Input: Enrollement Certificate (EC) requested from certificate Authority(CA)

Output: Access to all Stakeholder(S_T): M_N,D_N,T_N

Initialization: N_{ADMIN} should be valid nodes.

```
while true do
        if M_N is valid then
                add_node (M_N,S_T);
                grant_access(C_N);
        else
                Not_valid(M_N);
        end

        if D_N is valid then
                add_node (D_N,S_T);
                grant_access(C_N);
        else
                Not_valid(D_N);
        end

        if T_N is valid then
                add_node (T_N,S_T);
                grant_access(T_N);
        else
                Not_valid(T_N);
        end
end
```

Algorithm 2: Create Report

Input: ID and Key requested from N_{ADMIN}

Output: Access to transaction

Initilization: M_N,D_N,T_N should be valid nodes

```
While True do
        if M_N is  in ST_N
                if P_NID is in not in P_N then
```

Create_Project(P_{Nid});

else

Read_Project(P_{Nid});

update_Project(P_{Nid});

end

else

Not_valid(P_{Nid});

end

if D_N is in ST_N

if P_{NID} is in not in P_N then

Notify(Error);

else

Read_Project(P_{Nid});

update_Project(P_{Nid});

end

else

Not_valid(P_{Nid});

end

if T_N is in ST_N

if P_{NID} is in not in P_N then

Notify(Error);

else

Read_Project(P_{Nid});

update_Project(P_{Nid});

end

else

Not_valid(P_{Nid});

end

If DocType is TestingReport

P_{NHash} ← hashData(TestReport).projecthash;

projectRegistry ← getAssetRegistry(P_N);

P_{RG} ← P_N;

else

Error("not allowed to Update the Hash of Project...");

end

If DocType is DeploymentReport

P_{NHash} ←hashData(TestReport).projecthash;

projectRegistry ← getAssetRegistry(P_N);

P_{RG} ← P_N;

else

Error("not allowed to Update the Hash of Project...");

end

If DocType is MonitoringReport

P_{NHash} ← hashData(TestReport).projecthash;

projectRegistry ← getAssetRegistry(P_N);

P_{RG} ← P_N;

else

Error("not allowed to Update the Hash of Project...");

end

end

Algorithm 3: Create Project

Project Set:$P_{N,}$

Stakeholders S_T , Manager M_N ,Developer D_N Project Registry P_{RG}

Input: Project Data

Output: Project P_N

function createProject

{

While True do

If P_N is not in P_{RG}

factory ← getFactoryDetails;

project P_N ← newResource(NS,'Project',createData.projectid);

add description;

create P_N Collaboration Chat

relationship ←newRelationship(S_T,P_N);

projectRegistry ← getAssetRegistry(P_N);

P_{RG} ← P_N

else

Notify(“Error!”);

end

}

Algorithm 4: Update Document

Project Set:PN,

Stakeholders S_T :Manager : M_N Developer D_N $P_{RG,}$Client Asset CA_{RG}

Input: Project Document $PD_{I,}$ PD_{IHash}

Output: Project P_N

function addDocumentHash

{

While True do

If P_N is not in P_{RG}

Error("not allowed to create Document")

else

P_{RG} ← getAssetRegistry(C_D);

clientAsset ← await clientAssetRegistry.get(dochash.cpid);

S_T ← getCurrentParticipant().getType();

S_{TID} ← getCurrentParticipant().getIdentifier();

if(CA_{iRG} is not in CA_{RG}){

Error("not allowed to perform the operation")

else

CA_{RG}← PD_{IHash}

end

}

Following table 1.1 compares the current DevOps approach with DevOpsChain approach with its advantages.

Table 4.1: Comparision of DevOps approach with DevOpsChain approach

Phase	**DevOps Approach**	**DevOpsChain Approach**	**Advantages**
Plan	Stakeholders and customers' needs and opinions are used to make a product roadmap that will guide future development.	Requirements and feedback documents are stored in DevOpsChain. All Communications are also maintained.	Files can be available to easily available to each stakeholder with each modification so creates trust and changed new requirements get updated to all stakeholders
Code	Developers begin developing the code for the application and code repositories are stored in Version Control Systems	Code repositories are stored to Secure Blockchain-Based Version Control	We need to monitor and track each modification to a file or set of files over time.
Build	Developers commit their code to a shared code repository, and the codebase is built and runs a series of end-to-end, integration and unit tests to identify any regressions	Build Reports,War,Jar Files are stored to DevOpsChain	Any of the previous builds can be easily restored with immutable update history
Test	After putting the application into the test environment, a series of manual and automatic tests are run.	After testing, Test Reports are stored to DevOpsChain	Test incident report, Test cycle report, Test Summary Report, Test results,also QA plan, Revised bugs list, User Acceptance test files are available at each stakeholder
Release	build becomes ready for deployment into the production	Integration cases, Yaml fles are stored to DevOpsChain	YAML files. They need to be managed with time and authorization.

	environment,Automation scripts are manually passed.		To handle code break issue
Deploy	Build is released into production	Build released and Build release notification reports are stored to DevOpsChain	property files, logers .class files ,war jar . testresults gets generated
Operate	Release becomes live and being used by the customers, customers provide feedback on their service	Customers feedback are stored to DevOpsChain	standard output process (SOP), customer feedback reports
Monitor	Monitoring Reports, Customer feedback , analytics on customer behaviour, performance, errors given Product Manager and the development team to close the loop on the process	Monitoring Reports, Analytics and customer behaviour are stored to DevOpsChain	these documents need to be stored on blockchain, so it can create trust, province among stakeholders and it will product development efficiency.

4.5 Summary

In the first part of this section, a different generalized architecture is discussed at length. In the second part, the proposed architecture of the DevOps project development is discussed followed by a discussion of the parts of the framework. The fourth part deals with the algorithms used in the proposed work. Finally, a comparison of the DevOps approach with the proposed DevOpsChain approach is done in the last part of the section.