





CI/CD Process Development for Blockchain Environment Based on Hyperledger Infrastructure

Maciej Kopa¹, Michał Pawlak¹, Aneta Poniszewska-Marańda¹ ^a, Tomasz Krym^{1,2},
Łukasz Chomatek¹ ^b, Joanna Ochelska-Mierzejewska¹ ^c, Bożena Borowska¹ ^d,
Adam Czyżewski¹ and Krzysztof Stepień¹

¹*Institute of Information Technology, Lodz University of Technology, Łódź, Poland*

²*kodegenix, Łódź, Poland*

Keywords: Blockchain, DevOps, Continuous Integration and Continuous Delivery, Hyperledger Fabric, Chaincode.

Abstract: Blockchain is a promising and quickly developing technology with the potential to disrupt numerous industries. Hyperledger Fabric, an open-source blockchain platform can be used in conjunction with continuous integration and continuous delivery (CI/CD) pipelines. The paper presents the well-organized blockchain development environment for software engineers, its configuration and use in the form of CI/CD pipeline and its integration with Hyperledger Fabric. A case study illustrates the implementation of these technologies in real-world scenario to provide a comprehensive understanding of how Hyperledger Fabric and CI/CD pipelines can be leveraged to improve the efficiency of private blockchain network development process.


1 INTRODUCTION


Blockchain is a promising and quickly developing technology with the potential to disrupt numerous industries. Due to the blockchain's decentralized character, it is difficult for a single person to edit or tamper with the data kept on the ledger without the agreement of the other members. The potential of blockchain technology to facilitate trust and transparency between parties without the need for a central authority or intermediary is one of its primary advantages. As a result, it is a great technology for use cases where confidence is crucial. Despite the fact that blockchain technology delivers tremendous trust and security solutions, its high learning curve makes it challenging for developers to leverage its potential fully.


Blockchain technology is a complex and broad topic with various platforms and solutions, which can make it difficult for developers to initiate and maintain effective development. To build complicated networks and ecosystems of smart contracts, decentral-


ized storage, and consensus protocols, blockchain applications require developers with extensive knowledge and experience. In addition, developers typically lack a comprehensive understanding of the blockchain's architecture and configuration, necessitating the inclusion of blockchain administrators and operators in every project. As a consequence, it is imperative that a solution must be implemented in order to overcome the challenges that blockchain developers face during the early stages of the development process.

The paper examines various aspects of blockchain technology, DevOps, and Hyperledger Fabric to present how to design and build an efficient, collaborative, and robust Continuous Integration/Continuous Delivery (CI/CD) blockchain development environment based on Hyperledger Fabric infrastructure for experimenting, testing, and delivering networks, as well as interacting with the network and implementing business logic. It discusses the software engineering and DevOps problems related to the Hyperledger Fabric project concerning the CI/CD environment. This comprises the study and design of private Blockchain with Hyperledger Fabric and other tools, its setup and maintenance, chaincode creation and testing, and the building of CI/CD pipelines.

^a  <https://orcid.org/0000-0001-7596-0813>

^b  <https://orcid.org/0000-0002-7651-6877>

^c  <https://orcid.org/0000-0002-9295-3962>

^d  <https://orcid.org/0000-0001-7640-5782>

2 BLOCKCHAIN TECHNOLOGY AND HYPERLEDGER FABRIC

Blockchain is a technology that incorporates a shared ledger concept for a number of application domains. It can be used in almost any scenario that requires a decentralized, reliable, trustworthy, and automated system for data storage and business logic (IBM, 2021). Blockchain can be defined as a shared, immutable registry that accelerates the recording of transactions and tracking of material assets across an enterprise network. Assets can be material (e.g. car, land) or immaterial (e.g. patent, opinion) (Bitcoin, 2021; Dogecoin, 2021).

The core concept behind blockchain is *Distributed Ledger Technology (DLT)* – decentralized registry of data that each participant collaborates to maintain (Li and Kassem, 2021). Thanks to data being replicated across the network participants, each of them holds the same view of truth. DLT can occur in three different types: permissionless (public), permissioned (private) and hybrid. The hybrid solution strives to benefit from both public and private blockchains. It denotes controlled access while maintaining the condition of being open – some processes are kept private while others are public.

Immutable records are one of the characteristics of blockchain which builds trust and transparency in the network. Blockchain network can be viewed as peer-to-peer communication between its participants, but everything that resulted from the communication is recorded in a public ledger and accessible to all. The information recorded in the blockchain is guaranteed to be immutable thanks to many cryptographic techniques (Zhang et al., 2020), in such a system it is easy to track and prove the ownership of an asset. It also protects from changing and tampering with data.

Smart contracts are a set of rules that enables a bunch of ledger functions, e.g. transactions, querying, etc. They define business logic that generates new facts that are added to the ledger. They are written by administrators, agreed upon by participants, and saved on the ledger for automatic execution by clients. Each transaction in the blockchain network must be validated. *Consensus mechanisms* accumulate rules and govern how the validation is done. It should be understood as a full-circle verification that needs a number of steps to pass the check and approve the correctness of a block. There exist numerous types of these mechanisms, e.g. Proof of work in Bitcoin, Proof of stake in Ethereum, or highly customizable consensus policy from Hyperledger Fabric.

Hyperledger is an open-source platform for building and deploying blockchain solutions. It provides

distributed ledger software, libraries, and tools to help build almost any case-specific blockchain (Hyperledger, 2020a). Hyperledger Fabric is the most popular and followed GitHub project hosted by the Linux Foundation (GitHub, 2020). Fabric is intended to serve as a foundation for the development of private blockchain solutions (Gaur et al., 2018). It is the first blockchain architecture that is completely flexible for hosting any distributed applications (IBM, 2018). It offers a unique approach to consensus mechanism that provides the desired scalability and privacy with channels and private data (Team, 2020). Fabric supports smart contracts in, popular modern programming languages, Go, Java, and JavaScript, its governance, and versioning (Team, 2020). Fabric, in contrast to other blockchains, is cryptocurrency agnostic (IBM, 2018). Because of a flexible choice of pluggable databases – LevelDB or CouchDB – the data stored on the ledger is immutable and easily quarriable by peers. The assets can represent anything of value, making them versatile and adaptable to any use case. Identities in Fabric are managed by Membership Service Providers (MSP) and Certificate Authorities (CA). This makes Fabric a reliable system with a security architecture based on X.509 certificates and Public Key Infrastructure (PKI) (Gaur et al., 2018) and certificate attributes (Cooper et al., 2008). The current version of X.509 is RFC 5280 (Cooper et al., 2008) and is distributed by Certificate Authorities (CA), for example, Symantec (DigiCert).

3 HYPERLEDGER CHAINCODE DEVELOPMENT PIPELINES

The Hyperledger network is reliable, secure and provides a single source of truth, therefore, it can serve as a core of many systems. The goal of the project was to design and develop a system that can collect, store with blockchain and evaluate with Artificial Intelligence user opinions about products purchased. It is split into three separate modules that must swiftly interact to achieve desired outcomes (Fig. 1).

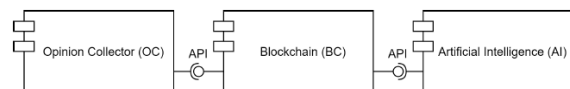


Figure 1: Modules of developed system.

The first module focuses on collecting and organizing the data from various sources. The source of the opinions depends on the actual needs, therefore the system should be flexible and easily adaptable to allow such multiple choices. The first and major

source of the data is going to be Opinion Collector (OC) application which is a custom-made web service with functions such as listing, collecting, and sending opinions further to the core of the system which is the blockchain module. The goal is to have a standalone service that will always and independently collect more and more data from users. The OC will also manage its own relational database, which will be additionally powered with the data previously stored by OC and other network participants inside the blockchain.

The next module, namely Artificial Intelligence (AI) aims to retrieve the data from the blockchain, analyse it and reflect the decision inside the blockchain. This module is responsible also for automatically training the newer and newer versions of the model and persisting it on its own. The model will never be exposed to the blockchain, but only be used for evaluations of new data on demand. The results, however, will be posted back in the ledger inside a different channel than the one with data providers. This will provide versatility and allow for the further expenditure of the network by adding new, interested cooperation companies, consequently providing more and more benefits from the system.

The core and fundament of the system is blockchain module (BC) built with Hyperledger Fabric. It merges every module into one reliable and fully functional system. The development process is driven by the needs of other modules as well as any security and reliability concerns. The network provides a place where invited organizations can interact in order to produce the desired value. The initial network consists of 3 organizations (other 2 modules and ordering service) and 1 channel, but with considerable space for expansion shortly. The network utilizes designed smart contracts for any operational logic involved and provides APIs for participants for safe interactions from outside the network.

Overall, this project aims to provide a comprehensive solution for collecting, analysing and utilizing data to allow more efficient and reliable decision-making. The opinions will originate from different sources and be analysed by an artificial intelligence model trained with substituted data. Everything of value will be placed inside the ledgers of the blockchain module and accessible for invited organizations that will participate in the maintenance of the network with an exchange of the new data or outcomes of their actions. This paper focuses on the BC module configuration and prototype development. It assumes that the OC and AI modules are independent, thus are out of the scope of this paper. Nevertheless, some decisions during the BC module development

are taken with explicit consideration for these modules, thus effectively influencing the shape of the network and development environment.

Hyperledger Fabric is highly modular and configurable, allowing developers to choose the components that are needed for a specific use case, and to easily add or remove functionality as required. This flexibility makes it easy to create solutions that meet the specific needs of different organizations. Privacy, performance, and scalability are key advantages of using Hyperledger Fabric, allowing organizations to reliably manage large volumes of transactions without any additional costs involved. That is why Hyperledger Fabric is a better choice for this case study than other blockchain platforms.

3.1 Architecture of Chaincode

It must be determined whether the network will be a test or production network. This decision is crucial because it indicates what actions must be taken and what the benefits of the system will be. The purpose of presented works was to prepare a development environment so that the network is built around the test network requirements. The test network provides advantages such as increased performance during chaincode development and allows the developer to focus on the solution and test cases rather than the network's technical aspects. The test network replaces some of its critical components with elements that operate on the same principle but have a simpler architecture. For example, the test network would not use any valid Certificate Authorities, but all cryptographic materials would be generated and stored locally using the cryptogen utility tool. Cryptogen tool is provided as a means of preconfiguring a network for testing purposes (Hyperledger, 2020b). As a result, all endorsements are automated, and proposals are generated faster than in a production network. Furthermore, it is likely to use a deprecated solo ordering service or raft ordering service with only one node, which boosts efficiency and saves resources but leaves it open to failures and brownouts. This example clearly demonstrates why a test network is insecure and should not be used in a production environment, but works flawlessly for testing both locally and during pipeline execution.

Reliable ordering service is a top priority for the production network. However, this component of the test network can be simplified. This can be done by using an algorithm for an ordering service that requires a minimum number of working peers to perform its duty. There are two options that can be considered good choices for a development environment:

solo or raft. Solo, as the name suggests, is a peer-to-peer, standalone ordering service. It is intended to function as a fully functional ordering service for Hyperledger Fabric, but it is regarded as unreliable and vulnerable to any potential data-tempering actions. Raft ordering service, on the other hand, is one of the ordering service implementations in Hyperledger Fabric that provides the most benefits. It uses the Raft consensus algorithm to ensure that all the ordering nodes in the network agree on the order of transactions. This ordering service provides a more fault-tolerant and decentralized way to order transactions compared to other ordering services such as Solo and Kafka. This guarantees that all peers in the network have the same version of the ledger and can quickly verify any transactions that are made. On top of that, Raft is the only recommended algorithm, while Solo and Kafka are both deprecated in Fabric version 2.5, although they can still be used. In this project, the ordering nodes (Raft) will be supplied by an administrative organization without the right to transact.

Since AI and OC modules perform different duties in the system, they also demand different transactions from the network. Developers must therefore take this into account when designing transactions and assets. This can be done in three different programming languages: JavaScript, Java, and Golang. As shown in figures 2 and 3, there are four different assets with four smart contracts. Each smart contract is responsible for the management of its assets stored in the ledger. All smart contracts are designed to be installed and invoked simultaneously on the same channel and accessible to both organizations. This decision is due to the fact that some of the contracts rely on each other, and storing them on different channels would unnecessarily slow down the transactions.

The assets are designed to optimize their usefulness in terms of business logic, as well as query capabilities. Two assets, namely Product and Review, are the core assets of the system. The other two assets are inextricably linked to the main assets. Specifically, products have critical relationships with product categories and reviews have critical relationships with review grades. While features like product categories and review ratings could be built into the system's foundational components, their design is ultimately informed by the need to accommodate users' requirements while also preserving the system's high level of maintainability and performance. A dynamic, rather than static, the addition of product categories is required. Review grades, on the other hand, will change often, and to keep these changes small, in terms of additional bytes stored on the ledger, it was decided to keep them separated from reviews that are very un-

likely to change ever.

The designed blockchain network is presented in figure 4. It contains one channel and ordering service, with one node dedicated to this channel (Raft consensus algorithm). Two organizations are connected to the channel via peers. Each of them has a peer, its own ledger and has installed smart contracts on it. Organizations invoke transactions via a dedicated API for each organization and communicate with each other on the channel via peers.

3.2 CI/CD Pipelines Structure

CI/CD pipelines can greatly increase the efficiency of developers during Hyperledger Fabric network development by providing a streamlined and automated workflow for building, testing, and deploying code (Fig. 5). The pipelines are constructed with main and child pipelines. Parent pipeline triggers given child pipeline only on code changes regarding this chaincode. The child pipeline contains jobs such as building and various testing. If all chaincodes succeed, the parent pipeline triggers the next stage where jobs build and turn on the network, install all chaincodes, and fire integration tests against this network. Lastly, the pipeline delivers tested containers to the docker repository. This can help reduce errors and improve the overall quality of the codebase. In the end, this will speed up the development process and make it possible to add new features and functions more quickly. Moreover, the workflow allows for efficient deployment of smart contracts on a Hyperledger Fabric network.

Continuous Integration (CI) pipelines can be combined with automated testing tools, which can run a suite of tests on the codebase each time a change is made. That is why every change to the codebase will trigger a pipeline dedicated to the given chaincode where the change was detected. Additionally, this process of separation will also optimize the pipeline and shorten its duration. The pipeline will cover code linting and unit testing with the Gradle plugin, scenario testing with the Cucumber tool from Smartbear, and finally, integration testing against a real network with a simple bash script that utilizes curl and grep commands.

With CI/CD in place, developers can easily deploy new versions of smart contracts to a test network, test them, and ensure they are working as expected, before deploying them to a live network. It will also ensure that the entire work is stable and free of errors. The test network will be created with the use of configuration files and the Docker compose tool every time a pipeline is triggered. As a result, the pipeline job will



Figure 2: Smart Contract classes for Chaincodes.

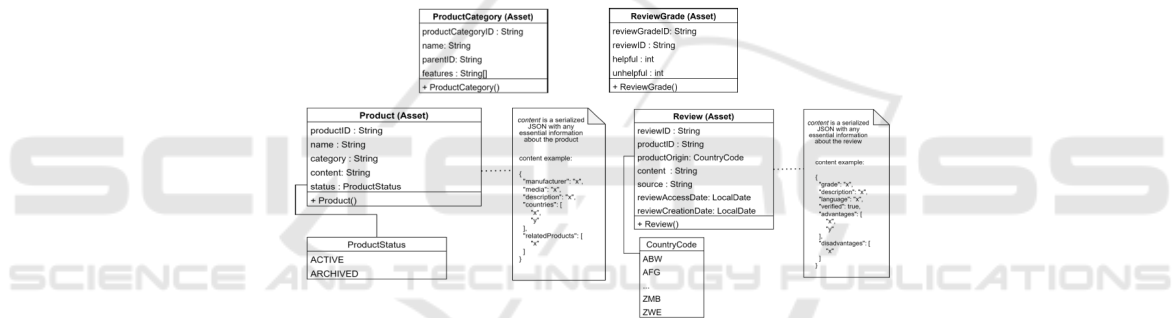


Figure 3: Assets structure for Chaincodes: ProductCategory, ReviewGrade, Product and Review.

result in a bunch of dependent containers. If this process succeeds, the integration tests will allow ensuring the quality of the product. Next, containers will be sent to the repository, where the operations team can take over and proceed with the deployment of the network. This step finalizes the process of CD.

4 DEVELOPMENT ENVIRONMENT

Process of building the network and end-to-end pipelines for chaincode development, testing and delivery are presented below.

4.1 Network

The blockchain network was based on Hyperledger Fabric with the use of external tool called Fablo that simplifies the process of creating and deploying Hy-

perledger Fabric networks locally. The tool provides a set of commands to automate the creation and deployment of blockchain networks.

The first step to create Hyperledger Fabric network with Fablo is to create an empty project and execute *fablo init rest* command. It will create a basic *fablo-config.json* file and enable Fablo REST tool to be useful later. The configuration file contains all information about the network, Fabric version, organizations, channels, chaincodes. The Fablo configuration file specifies OC, AI, and Orderer organizations, pro-rev-channel and its participants, and Fabric version that supports TLS communication. In addition, it specifies the Raft algorithm and number of nodes for Orderer. The network created by Fablo can be deployed elsewhere. It can be achieved by running *fablo generate* and changing both *fabric-docker.sh* and *docker-compose.yaml* files.

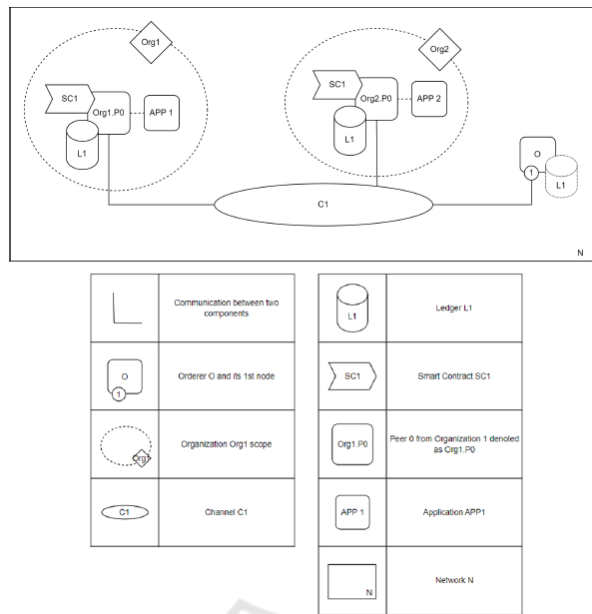


Figure 4: Initial network architecture reference.

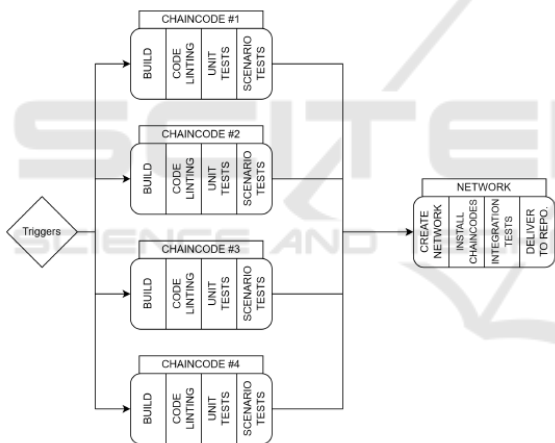


Figure 5: CI/CD pipelines workflow structure with steps.

4.2 Chaincodes

While the network is already up and running, it lacks the business logic to perform its duties such as e.g. storing product reviews in our developing system. The implementation of exemplary assets as Product, ProductCategory, Review, and ReviewGrade was done using Java language and Gradle tool, used for building, testing, and deploying the chaincodes.

The setup of a project (chaincode) requiring the creation of a new empty Gradle (version 7.0.2) project and the modification of fablo-config.json by adding the chaincode definitions that Fablo will attempt to install and initiate in the specified channel. Fablo will also acquire the definitions of all chaincodes. The en-

dorsement key specifies the policy that needs to be met for a given chaincode. Also, the IDE should have access to JDK 11. The *build.gradle* file must then be configured by adding two repositories and plugins. One plugin allows generating fat/uber JAR files and other is for code linting. The Gradle shadow-Jar task must be configured to generate the JARs. The same goes for Checkstyle plugin. Also, the necessary dependencies must be considered. It includes libraries such as Genson that allow objects to be parsed to JSON in a consistent manner, as well as a Java implementation of Hyperledger Fabric chaincode interface SDK.

The next step is process of creating assets, from the definition of asset's attributes to implementation of smart contracts (chaincode) that will handle the asset's transactions and operations. This task can be discussed in detail for example for *Product* assets. It requires defining a new public class *Product* and annotating it with *DataType*. Inside the class object, there should be declarations of class variables according to figure 3 and with another annotation *Property* that can be used to define a schema for String or min/max values for Integer. The class must also offer getters and a constructor.

The smart contract includes functions that represent transactions for blockchain network (Fig. 2). The *ProductContract* class must implement *Contract-Interface* and have *Contract* and *Default* annotations. Following that, each transactional function must have an appropriate *Transaction* annotation with intent defined. The type of transaction is determined by the

intent, such as `SUBMIT`, which changes the ledger, or `EVALUATE`, which only reads it. A *Context* must always be passed in addition to any function parameters. `createProduct()` is a simple function that takes the *Context*, and name, category, content to prepare a *Product* with generated `UUID` for `productID`. Additionally, it checks if it does not already exist. Following that, it serializes the *Product* class with `Genson` and, thanks to *Context*, obtains the `ChaincodeStub` to insert a new state into it.

4.3 CI/CD Pipelines

The next step is to set up CI/CD pipelines using GitLab CI/CD services for previously created fully functional blockchain for one of the main purposes of our system, so the user opinion storage. GitLab CI/CD is a built-in feature of GitLab that allows the automated testing, building, and deployment of code changes. The pipelines is created from parent and children pipelines, this means they contain five `.gitlab-ci.yml` files, one for each chaincode and one core pipeline. These files define all stages and jobs, and enforce automation and performance rules.

All child pipelines dedicated to integrating chaincodes look in the same way as shown in figure 5. Each child pipeline contains two stages and two jobs. The first stage is the build stage with a dedicated job. It simply compiles the code with Gradle and for optimization purposes, the results get cached. This job must succeed in order for the next job to start. The next stage and job is more complex and offers three benefits: code listing, unit testing and scenario testing. Everything is managed by a Gradle tool, `JUnit`, `Checkstyle`, and `Cucumber` plugins. Before the script starts, it sets up the Gradle user home directory and applies it to the current CLI. This step is required for all Gradle-related jobs inside a GitLab CI. When it is done, the core script begin with gradle check command, which implies three functions. First one lints the codebase of chaincode according to the given checkstyle file. Next one proceeds with verifying the unit tests, and last one triggers all scenario tests with `Cucumber` runner. It is worth noting that if Gradle check command fails on any step it will not proceed further. Therefore, it might be a good idea to split the Gradle tasks into three separate jobs. This will also decrease the duration time of a pipeline because these jobs could run simultaneously. However, this GitLab pipeline setup uses a single runner and running linting process automatically runs any other test cases, therefore the results would not be better for this exact setup. When children pipelines were executed for every chaincode that was changed and if all

succeed, the parent pipeline continues with the next stages.

The main pipeline `.gitlab-ci.yml` file is in project root folder, next to `fablo-config.json`. The main pipeline (Fig. 5) must include triggers for child pipelines, build and configure a network for tests, and deliver containers to storage. As a result, the workflow has three stages in the following order: triggers, test and deliver. The first stage includes a job for each chaincode – it has defined conditions known as rules or only that, when met, the main pipeline will include and execute the specified child pipeline. The only defined rule for this job is changes. As a result, it only occurs when GitLab detects a difference between this and the previous commit, in order to save time and resources. After all child pipelines have passed and the triggers stage has finished, it is time for the next step in the workflow, which is the test stage. It involves creating the actual network, installing the previously tested chaincodes and testing it against a real network. The job requires an image with docker and docker-compose already configured. Nonetheless, the environment requires the definition of a few variables that will be selected automatically by Docker and utilized during execution. This includes `DOCKER_HOST`, `DOCKER_DRIVER`, and `DOCKER_TLS_CERTDIR`. `DOCKER_HOST` is an environment variable that is used to specify the URL of Docker daemon. It ensures that Docker client will automatically connect correctly to running Docker daemon. `DOCKER_DRIVER` configuration option is used to specify the storage driver that should be used by Docker. The `overlay2` storage driver is a modern and recommended storage driver that provides better performance and stability in comparison to its predecessors. `DOCKER_TLS_CERTDIR` is an environment variable that is used to specify the directory where the Transport Layer Security (TLS) certificates for Docker daemon are located. When it is set to an empty string, it also requires turning the TLS communication off. It can be achieved by invoking the command `-tls = false` on creation.

Before the script can execute, it must download some basic dependencies, because the image comes as small as possible. Alpine Linux has its own package manager, `apk`, for installing and managing software packages. The main script pulls the Fablo tool and starts the network using `fablo-config.json`. The chaincodes are already defined in the file and is installed after the network is up and running. Next, the job pulls another tool, which is the `Api-test bash script` and runs all test cases against Fablo REST API.

To implement CD, the project first established a strong testing infrastructure and Continuous Integra-

tion. This includes automated unit tests, integration tests and scenario tests to ensure that new code changes will not introduce bugs or break existing functions. The final step in the workflow (Fig. 5) is the delivery of product to a destination where the operators team can take over and focus on its deployment. The delivery stage is located in the main pipeline and requires the previous stage to finish successfully. Regarding all necessities such as services, variables, and tools, the delivery stage is quite comparable to the preceding one. As the network is not intended to be started within this job, only the docker images and *docker-compose.yml* file need to be prepared. It is possible with fablo generate command. It creates fablo-target folder containing all required files within the root folder of the project. Next, the job uses a simple bash script that tags generated docker images with the tag following the GitLab schema `registry URL:/namespace:/project:/image:`. For example, a container with the name *ca.orderer.example.com* is tagged with the command `docker tag ca.orderer.example.com registry/salus/blockchain/ca.orderer.example.com:latest`. The last command docker compose push sends all images to the specified registry.

5 CONCLUSIONS

The paper presented proposal on how a blockchain development environment should look like and its implementation with use of Hyperledger Fabric, Fablo, Docker, Java with Gradle, and GitLab. Fablo and Docker were used to configure and manage the network initially. It consists of three organizations, one channel, and a policy requiring at least fifty percent of participants to validate. Each organization has a single peer which fulfils all peer responsibilities. Peers are responsible for managing smart contracts, assets, hosting ledger copies, endorsement, validation, and transaction invocation. This rule applies to all organizations with the exception of ordering service that was developed using the Raft consensus algorithm and a single node to reduce resource consumption and accelerate the process.

Using the Gradle automation tool, four smart contracts and four assets were developed in the Java language. In addition, the chaincode development process entailed editing the Fablo configuration file, applying code linting with Checkstyle, writing unit tests with JUnit, preparing scenario tests with Cucumber, and creating integration test cases with Postman and Api-test script. In order to automate the development environment, the CI/CD pipelines were config-

ured using GitLab's CI/CD service. The introduced workflow consists of two interdependent components. First, the child pipelines that are responsible for automating the integration of each chaincode only when changes occur. Each includes a build task, linting, unit testing, and scenario testing. The main pipeline proposed a method for triggering the child pipelines, created a network within the runner, tested the network using a suite of integration tests, and then delivered everything to the GitLab registry. Total configuration is limited to five files, making the management of pipelines relatively simple.

ACKNOWLEDGEMENTS

The presented work was supported under the grant of Polish National Center of Research and Development, Project No. INFOSTRATEG-III/0004/2021-00, "Artificial Intelligence and Blockchain for the product quality and safety control system (SALUS)".

REFERENCES

- Bitcoin (2021). *Bitcoin technology*. Online, <https://bitcoin.org/>.
- Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W. (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.
- Dogecoin (2021). *Dogecoin technology*. Online.
- Gaur, N., Desrosiers, L., Ramakrishna, V., Novotny, P., Baset, D., and O'Dowd, A. (2018). *Hands-On Blockchain with Hyperledger*. Packt Publishing.
- GitHub (2020). *Hyperledger Fabric – GitHub repository*. Online, <https://github.com/hyperledger/fabric>.
- Hyperledger (2020a). *Hyperledger documentation*. Online, <https://www.hyperledger.org/>.
- Hyperledger (2020b). *Hyperledger Fabric Documentation*. Online, <https://hyperledger-fabric.readthedocs.io/en/release-2.5/>.
- IBM (2018). *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains*. Online, <https://arxiv.org/pdf/1801.10228.pdf>.
- IBM (2021). *What is blockchain technology?* Online, <https://www.ibm.com/topics/what-is-blockchain>.
- Li, J. and Kassem, M. (2021). Applications of distributed ledger technology (dlt) and blockchain-enabled smart contracts in construction.
- Team, H. (2020). *Hyperledger Fabric Whitepaper*. Online, <https://www.hyperledger.org/wpcontent/>.
- Zhang, R., Xue, R., and Liu, L. (2020). Security and privacy on blockchain.