DOI: 10.20998/2079-0023.2025.01.15
UDC 004.72

P. Y. ZHERZHERUNOV, Student, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine; e-mail: Pavlo.Zherzherunov@cs.khpi.edu.ua; ORCID: https://orcid.org/0009-0005-7240-9395
O. V. SHMATKO, Doctor of Philosophy (PhD), Docent, National Technical University "Kharkiv Polytechnic Institute", Ass. Prof of Software Engineering and Management Intelligent Technologies Department, Kharkiv, Ukraine, e mail: oleksandr.shmatko@khpi.edu.ua, ORCID: https://orcid.org/0000-0002-2426-900X

DESIGNING THE ARCHITECTURE AND SOFTWARE COMPONENTS OF THE DOCKERISED BLOCKCHAIN MEDIATOR

Small and medium enterprises are not adopting blockchain solutions in their supply chains and business processes due to the cost of implementing and deploying the solutions. The architecture, which is described, is aimed at lowering the barrier for smaller-scale businesses to adopt distributed technologies in their supply chains. Docker's containerization capabilities are leveraged to achieve these goals due to improved horizontal scaling and providing a unified environment for the application deployment. This architecture leverages tools provided by the Docker to design scalable and robust system that is easily maintainable. Some of the key challenges are addressed by the proposed architecture, such as high development costs, incompatibility with existing systems, and complicated setup processes, which are required for every participant in the supply chain. This research describes how utilizing Docker system capabilities can help enable smaller-scale businesses to adapt distributed solutions in their supply chains and cooperation with other companies by tackling the issues of traceability, transparency, and trust. The main components of the architecture are a mediator server containerized within a Docker network, a blockchain node, and an NGINX proxy server container. They are implemented to process request data, store relevant information, and secure it on the Ethereum blockchain ledger. The proposed architecture is also aimed at integrating smoothly with existing company applications to reduce adoption costs. Security of the data in the Ethereum ledger is achieved via security measures such as cryptography mechanisms and hashing already integrated into the Ethereum platform.

Keywords: dockerized blockchain architecture, supply chain management, containerized blockchain nodes, small-medium enterprises, supply chain, cryptography, hashing, hashing algorithms, ethereum.

Introduction. Blockchain technology has entered the space of supply chain management, allowing for increased transparency, traceability, and security of the supply and logistics operations that are performed in the chain. Large companies are more likely to incorporate distributed technology in their workflow due to the number of resources available and robustness of their structure, which allows for taking risks adopting new technology [1]. Smaller-scale businesses though usually don't have the resources needed to start using distributed solutions or augmenting their existing applications with blockchain tools because of not lack the resources to do so. Small to medium enterprises, or SME in short, are hesitant to apply blockchain solutions to their existing business models because of its financial and time costs, but are interested in trying this technology because of the benefits it could bring them [1]. There are some solutions present that are targeted at SMEs specifically. One of them is a dockerized architecture approach to the blockchain solution, focusing on containerized virtual nodes for the blockchain network and tools that allow for storing necessary data in the ledger [2]. It is described at a high-level in the work mentioned above. The goal of the current research is to design the architecture and software components for dockerized blockchain system in more details, describing how elements of this system could be implemented, so that development and deployment costs for applying distributed solutions would decrease for the small-medium enterprises, allowing for more companies trying the technology in their supply chains.

Designing system architecture. There are several concerns regarding the implementation and deployment of the blockchain solutions, which are mentioned in this work [2]. Dockerized blockchain system is being designed based on the problems that are tackled with that approach. Those problems are mostly with the incorporating of the blockchain solution for smaller businesses and they can be named like this:

- Expensive development and deployment process for blockchain tools
- Incompatibility of new blockchain tools with the existing businesses process and existing applications
- Difficulties in setup process of the new tools for each participant of the supply chain
- Lack of pre-developed blockchain solutions that would allow for quick and easy integration of the solution into existing applications and systems

Based on the problems discussed above, it was decided to use Docker as a platform for packaging and running applications [3], among which are going to be server mediator for processing data before it goes to the ledger, blockchain nodes and a proxy server for redirecting requests to a necessary service in the docker network or outside of it, to the existing applications.

Docker is selected for the solution due to it having several cons that directly help with the issues mentioned above. Docker packages applications and their dependencies into isolated containers, ensuring consistent execution across different environments. This eliminates the "it works on my machine" problem [3]. That allows for an easier process of development and deployment, which reduces financial and time costs of the solution adoption. Docker is scalable, as it simplifies horizontal scaling by allowing you to easily run multiple instances of a containerized application to handle increased load [4]. It allows for scaling existing containers in the network and

Research Article: This article was published by the publishing house of ***NTU "KhPI"*** in the collection "Bulletin of the National Technical University "KhPI" Series: System analysis, management and information technologies." This article is distributed under a Creative Common Creative Common Attribution (CC BY 4.0). **Conflict of Interest:** The author/s declared no conflict of interest.

adding new ones, which is allowed by isolation and security of the containers used. That enables easier integration of the company-participant of the supply chain or business cooperation. But all the dockerized blockchain mediators

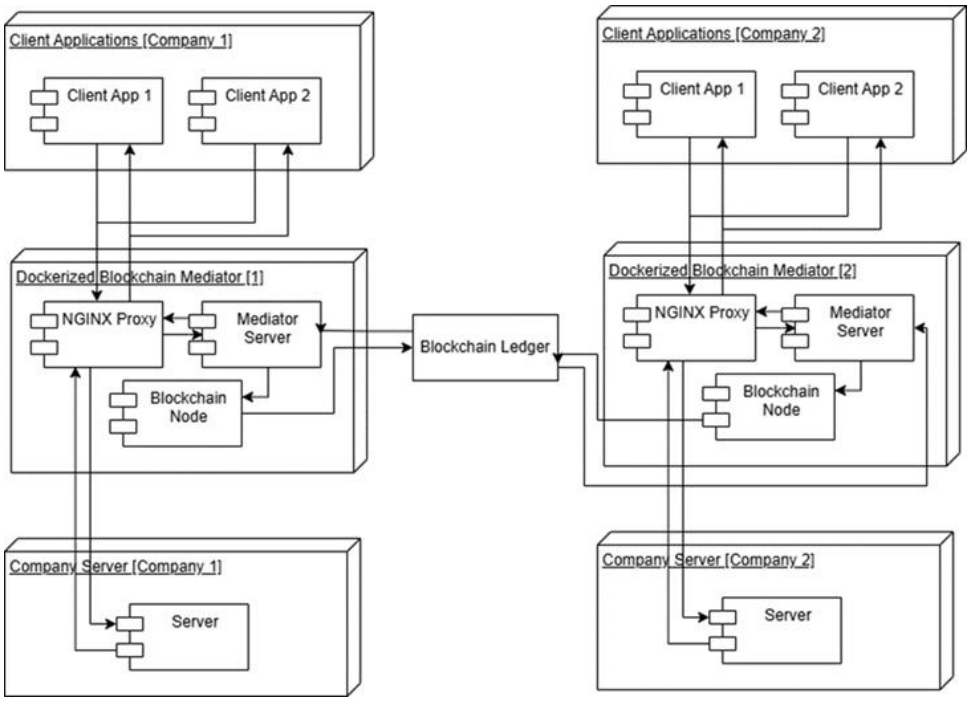

Fig. 1. Component diagram of the architecture

tools in the existing systems, as they do not have to be implemented in the existing codebase, but communicated to as a separate server, having its own interface.

System architecture using Docker has to account for the existing systems of the supply chain participants, allowing connecting them the blockchain network or not isolating requests in the blockchain node. That is achieved via mediator server, which is responsible for extracting data from the request, saving it to the ledger and passing request to the original server or application.

Solution's architecture consists of several elements:

- User client. This part of the application is not described in detail as it is usually a browser or OS application, which sends requests to the server, where logic is handled
- Docker Application Host. It is a server machine, which is responsible for hosting the docker network, which in basic configuration includes mediator server written in Python and Django framework, and blockchain Ethereum node. There is also a NGINX proxy that can redirect requests to the mediator, existing server outside of Docker network or additional modules, that can be manually added to the docker network
- Existing server. This is a server machine or proxy for existing server or network of servers, which are responsible for handling original business logic for that client in the supply chain processes

On the component diagram of the architecture main elements are displayed, where User client is name Client Applications, Docker Application Host is Dockerized Blockchain Mediator and Existing Server is Company Server. Each of these components is present for every connect via Ethereum nodes to a shared blockchain network with a shared ledger.

Client Applications component includes Client App instances, which are displayed as multiple elements in the component. Their number is not limited to two, and these can be different application, the main requirement to them is that they communicate with the existing Company Server. This logic is true for every company participant of the supply chain.

Dockerized Blockchain Mediator includes NGINX Proxy, Mediator Server and Blockchain Node containers. Their purpose and process details are described further in this research.

The architecture, including Docker implementation details, is displayed in the fig. 2. Details of the docker specific deployment of solution's components are described there.

The procedure of deploying a dockerized blockchain instance can be described as follows: host is running a virtual machine (VM) instance, which is responsible for managing docker daemon, main process in the Docker application, which manages images, containers and provides interface to control them [5]; docker daemon on docker-compose command is collecting image schemes and necessary resources from the registry and creates images based on them; containers are created based on the images that docker pulled from the registry; pre-developed mediator server and blockchain node are using python (which includes Django by default) and ethereum/client-go images respectively; NGINX proxy container is using a nginx/nginx-ingress image to build a proxy server, which is responsible for redirecting requests and returning modified responses [6, 7]; After these operations are completed,

application is ready and operational and other clients are communicating with it via docker-network, which is exposed on the real machine server.

Machine is a real server machine that hosts operating system which itself hosts a Docker network. To redirect a request to the NGINX Container, Mediator Server and

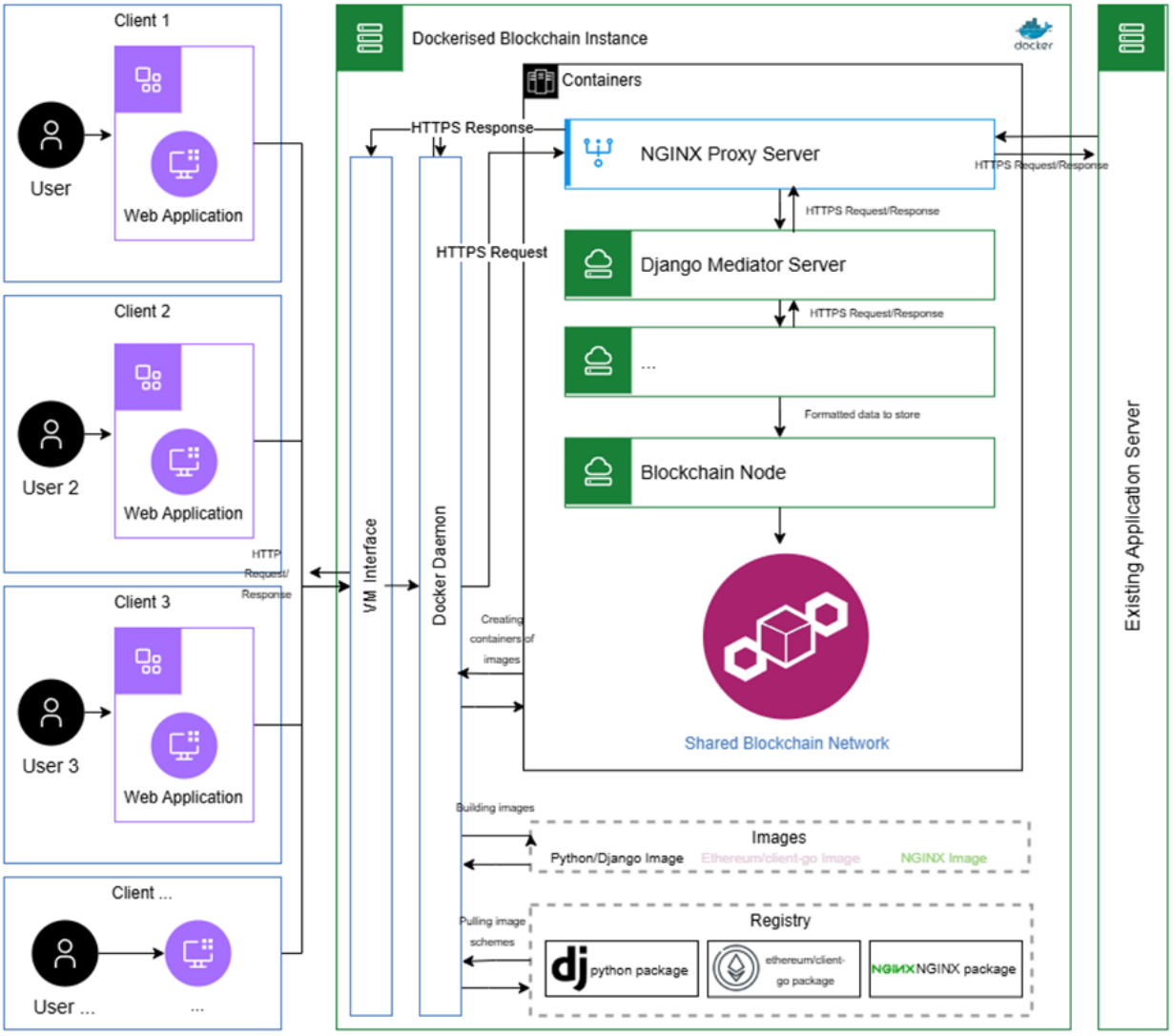

Fig. 2. Detailed Architecture of Blockchain Mediator Instance

Default Architecture's Process Sequence. Sequence diagram in the fig. 3 displays a default process of the application handling client requests to their existing server application, where request is temporarily intercepted by the blockchain mediator server. Diagram is composed taking into account structure of the dockerized application and specific of its deployment.

Objects of this diagram are Client App, Host Machine, Networking Subsystem, NGINX Container, Mediator Server, Blockchain Node and Original Server. Sequence diagram shows full process of request processing including different Docker layers [8].

Client App is a client already implemented and used by business company to communicate with their Original Server, where Client App is usually a web application or website and original server is a webserver responsible for handling logic.

Client App sends a HTTP request to a Host Machine, which is shown as a message arrow in the diagram. Host Blockchain Node which are parts responsible for handling logic of the request, Host Machine sends received request to the Networking Subsystem of the Docker, which then maps this request to a NGINX Container, which acts as a proxy in this architecture. These actions are also shown as message arrows in the sequence diagram, as synchronous calls, as they are waiting for response from the mentioned objects.

After NGINX Container receives request, it is now present in the virtual Docker network, where main objects are NGINX Container, Mediator Server and Blockchain Node. Multiple scenarios are possible within NGINX Container logic, but sequence diagram covers the default one, where request is processed, data is saved in the blockchain ledger via smart contract. To do this, NGINX container redirects request to a Mediator Server object, which is responsible for processing all request data and separating parts of it to be stored in the blockchain ledger. This logic is intended to be customizable, so it is not a focus

of the current research. After Mediator Server formats request metadata and sends it to a Blockchain Node, it waits for the completion event from it. When completion event is received, Mediator Server returns request to the NGINX Proxy and issues its redirect to the original server. All these operations are displayed as message arrows in the diagram.

NGINX Proxy sends an original request to the original server, if completion event is received, and then waits for response from it. From that point on, NGINX Container, Networking Subsystem and Host Machine are acting as sequential transmitters of original server response, which is aimed to return to the Client App.

Processing Request Data for Blockchain. Architecture is aimed at processing and saving essential parts of the request passing through the system to enabling tracking of the actions in the supply chain but also not create too large data storages which are hard to maintain.

Request structure consists of several parts, which are request line, headers and body which is an optional part of the requests that stores data sent in some of the HTTP methods [9].

Request line is the first line of the HTTP request and contains essential information about the request. It has three parts, separated by spaces: HTTP method (e.g., GET, POST, DELETE), request target (e.g., /api-endpoint) and HTTP version, which can be one of the HTTP protocols. We want to store the whole line in the ledger except HTTP version [9, 10].

Headers contain a lot of key:value pairs, where key is a name of a specific header and values is usually a string value with specific information. For example: Host: www.example.com, Accept-Language: en-US,en;q=0.5, etc. We would like to store not all of the headers, but those that are request essential, for example Host, User-Agent, Accept, Content-Type, Content-Length, Referer. But we must not store and security relevant information, like Cookie or Authorization header, as blockchain network is fully visible for the participants of the network, and providing such information would be a direct violation of security rules.

In the ledger we would also like to store body of the request. The request body is the part of an HTTP request that contains data being sent from the client to the server. It follows the request line and the headers. The presence and content of the request body depend on the HTTP method used and the purpose of the request.

Format of the request body is dependent on the Content-Type header, which is passed with the request. Examples of the content types are text/plain, application/json, application/xml. Body can also be a file, but that is a rarer case of the request, so it is not going to be looked at in the scope of this research.

Text/plain content type can be stored in the ledger directly as a body: content pair, where content is String type. Application/json, application/xml and application/x-www-form-urlencoded are using different formatting, but they can be converted to the dictionary data structure, where each field is either a single String or Number or another dictionary which contains other fields. This transformation of JSON, text, XML or form data has to be done in the Mediator Server prior to sending to the Blockchain Node and a corresponding smart contract to reduce load on the onchain operations.

Securing distributed request data. After resulting data object representing the request is sent to smart contract and from there stored to ledger, new Ethereum ledger records are secured through a combination of cryptographic techniques and a robust consensus mechanism. The default Ethereum approach is used to securing newly obtained data.

Ethereum uses elliptic curve cryptography (specifically secp256k1), the same as Bitcoin, to manage accounts and transactions. Each Ethereum account has a public key and a private key pair. The public key acts like an account number, allowing others to connect new data to other members of the blockchain network. It can be shared publicly without risk. The private key is like a digital signature. It's a secret key that allows the account owner to authorize transactions which change the latest state of the ledger. [11].

For hashing Ethereum uses cryptographic hash functions (like Keccak-256) extensively, to create unique identifiers for operations in the ledger and to link blocks together in the blockchain, forming a tamper-evident chain. Each block's header contains a hash of the previous block's header [12].

Conclusions. This research describes the design of architecture and basic software components for a dockerized blockchain mediator system aimed at lowering the barrier for small to medium enterprises to adopt blockchain technology in their supply chains.

To achieve these goals, Docker's containerization capabilities are leveraged, and as a result the proposed architecture addresses key challenges such as high development costs, incompatibility with existing systems, and complex setup processes. This research describes how using Docker system can help addressing these issues to enable more smaller scale businesses to adapt distributed solutions in their supply chains and cooperation with other companies.

The architecture features a mediator server within a Docker network, alongside blockchain nodes and a proxy server, to process request data, store relevant information securely on the Ethereum blockchain ledger, and integrate smoothly with existing company applications. Security of the data in Ethereum ledger is achieved via security measures such as cryptography mechanisms and hashing already integrated in the Ethereum platform.

This approach simplifies deployment, enhances scalability, and maintains security through established cryptographic methods, ultimately offering a more feasible path for SMEs to explore the benefits of blockchain for improved supply chain transparency and traceability.

References

1. Shubham Joshi, Anil Audumbar Pise, Manish Shrivastava, C. Revathy, Harish Kumar, Omar Alsetoohy, Reynah Akwafo. Adoption of blockchain technology for privacy and security in the context of industry 4.0. *Wireless Communications and Mobile Computing*. 2022. iss. Explorations in Pattern Recognition and Computer Vision for Industry 4.0. article 4079781. DOI: https://doi.org/10.1155/2022/4079781.
2. Шматко О. В., Коломійцев О. В., Жержерунов П. Ю., Третяк В. Ф., Сінчук А. В. Survey and categorization of blockchain solutions for supply chain management. *Системи обробки*

інформації. 2024. №. 3 (178), P. 84–92. DOI: https://doi.org/10.30748/soi.2024.178.10.
3. Jangla Kinnary. *Accelerating Development Velocity Using Docker: Docker Across Microservices*. Berkeley: Apress, 2018. P. 27-53. URL: https://link.springer.com/book/10.1007/978-1-4842-3936-0 (access date: 15.04.2025).
4. Miell I., Sayers A. *Docker in practice*. New York City: Simon and Schuster, 2019. P. 384. URL: https://www.simonandschuster.com/books/Docker-in-Practice-Second-Edition/Ian-Miell/9781617294808 (access date: 15.04.2025).
5. *dockerd*. URL: docs.docker.com/reference/cli/dockerd/ (access date 10.04.2025).
6. *NGINX Ingress Controller*. URL: https://hub.docker.com/r/nginx/nginx-ingress (access date: 15.04.2025).
7. *ethereum/client-go*. URL: https://hub.docker.com/r/ethereum/client-go (access date: 15.04.2025).
8. *What Is a Sequence Diagram*. URL: www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (access date: 18.04.2025).
9. Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*. United States: RFC Editor, 1999. DOI: https://doi.org/10.17487/RFC2616.
10. Pollard B. *HTTP/2 in Action*. New York City: Simon and Schuster, 2019. P. 416. URL: https://www.simonandschuster.com/books/HTTP-2-in-Action/Barry-Pollard/9781617295164 (access date: 18.04.2025)
11. Andreas M., Gavin W. *Mastering Ethereum*. Sebastopol: O'Reilly Media, 2019. P. 415.
12. Danen C. *Introducing Ethereum and solidity*. Berkeley: Apress, 2017. Vol. 1.. P. 185.

References (transliterated)

1. Shubham Joshi, Anil Audumbar Pise, Manish Shrivastava, C. Revathy, Harish Kumar, Omar Alsetoohy, Reynah Akwafo. Adoption of blockchain technology for privacy and security in the context of industry 4.0. *Wireless Communications and Mobile Computing*. 2022. issue Explorations in Pattern Recognition and Computer Vision for Industry 4.0. article 4079781. DOI: https://doi.org/10.1155/2022/4079781.
2. Shmatko O. V., Kolomiitsev O. V., Zherzherunov P. Y., Tretiak V. F., Sinchuk A. V. Survey and categorization of blockchain solutions for supply chain management. *Systemy obrobky informatsii* [Information processing systems] 2024 no. 3 (178), pp. 84-92. DOI: https://doi.org/10.30748/soi.2024.178.10.
3. Jangla Kinnary. *Accelerating Development Velocity Using Docker: Docker Across Microservices*. Berkeley: Apress, 2018. pp. 27-53. Available at: https://link.springer.com/book/10.1007/978-1-4842-3936-0 (accessed: 15.04.2025).
4. Miell I., Sayers A. *Docker in practice*. New York City: Simon and Schuster, 2019. 384 p. Available at: https://www.simonandschuster.com/books/Docker-in-Practice-Second-Edition/Ian-Miell/9781617294808 (accessed: 15.04.2025).
5. *dockerd*. URL: docs.docker.com/reference/cli/dockerd/ (accessed 10.04.2025).
6. *NGINX Ingress Controller*. Available at: https://hub.docker.com/r/nginx/nginx-ingress (accessed: 15.04.2025).
7. *ethereum/client-go*. URL: https://hub.docker.com/r/ethereum/client-go (accessed: 15.04.2025).
8. *What Is a Sequence Diagram*. Available at: www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (accessed: 18.04.2025).
9. Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*. United States: RFC Editor, 1999. DOI: https://doi.org/10.17487/RFC2616.
10. Pollard B. *HTTP/2 in Action*. New York City: Simon and Schuster, 2019. 416 p. Available at: https://www.simonandschuster.com/books/HTTP-2-in-Action/Barry-Pollard/9781617295164 (accessed: 18.04.2025)
11. Andreas M., Gavin W. *Mastering Ethereum*. Sebastopol: O'Reilly Media, 2019. 415 p.
12. Danen C. *Introducing Ethereum and solidity*. Berkeley: Apress, 2017. Vol. 1. 185 p.

Received 14.05.2025

УДК 004.72

П. Ю. ЖЕРЖЕРУНОВ, студент, Національний технічний університет «Харківський політехнічний інститут», м. Харків, Україна, e-mail: Pavlo.Zherzherunov@cs.khpi.edu.ua, ORCID: https://orcid.org/0009-0005-7240-9395
О. В. ШМАТКО, доктор філософії (PhD), доцент, Національний технічний університет «Харківський політехнічний інститут», доцент кафедри програмної інженерії та інтелектуальних технологій управління, м. Харків, Україна, e-mail: oleksandr.shmatko@khpi.edu.ua, ORCID: https://orcid.org/0000-0002-2426-900X

РОЗРОБКА АРХІТЕКТУРИ ТА ПРОГРАМНИХ КОМПОНЕНТІВ ДОКЕРИЗОВАНОГО БЛОКЧЕЙН-МЕДІАТОРА

Малі та середні підприємства не впроваджують блокчейн-рішення у свої ланцюги постачання та бізнес-процеси через високу вартість їх впровадження та розгортання. Описана архітектура спрямована на зниження бар'єрів для малих підприємств у впровадженні розподілених технологій у свої ланцюги постачання. Для досягнення цих цілей використовуються можливості контейнеризації Docker завдяки покращеному горизонтальному масштабуванню та забезпеченню єдиного середовища для розгортання додатків. Ця архітектура використовує інструменти, надані Docker, для проектування масштабованої та надійної системи, яка легко підтримується. Пропонована архітектура вирішує деякі ключові проблеми, такі як високі витрати на розробку, несумісність з існуючими системами та складні процеси налаштування, які необхідні для кожного учасника ланцюга постачання. У цьому дослідженні описано, як використання можливостей системи Docker може допомогти малим підприємствам адаптувати розподілені рішення у своїх ланцюгах постачання та співпраці з іншими компаніями шляхом вирішення проблем простежуваності, прозорості та довіри. Основними компонентами архітектури є контейнерний сервер-посередник у мережі Docker, вузол блокчейну та контейнерний проксі-сервер NGINX. Вони реалізовані для обробки даних запитів, зберігання відповідної інформації та її захисту в реєстрі блокчейну Ethereum. Запропонована архітектура також спрямована на плавну інтеграцію з існуючими додатками компанії для зменшення витрат на впровадження. Безпека даних у реєстрі Ethereum забезпечується за допомогою таких заходів безпеки, як механізми криптографії та хешування, вже інтегровані в платформу Ethereum.

Ключові слова: докеризована архітектура блокчейну, управління ланцюгами поставок, контейнеризовані вузли блокчейну, малі та середні підприємства, ланцюг поставок, криптографія, хешування, алгоритми хешування, Ethereum.

Повні імена авторів / Author's full names

Автор 1 / Author 1: Жержерунов Павло Юрійович / Zherzherunov Pavlo Yuriiovych
Автор 2 / Author 2: Шматко Олександр Віталійович / Shmatko Olexandr Vitaliiovych