

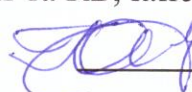
Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Інтеграція Jenkins та Docker для автоматизації
розгортання проєктів»

Виконав: студент 4-го курсу групи КІ-21-2
спеціальності 123 «Комп'ютерна інженерія»

 Максим КОС
Керівник: доцент кафедри КІ та КБ, к.пед.н., доцент

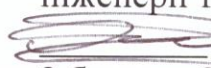
 Олена ГОЛОВНЯ

Рецензент: старший викладач кафедри КІ та КБ

 Вадим МИКОЛАЙЧУК

Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки
 Андрій Єфіменко
3 березня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Кос Максим Олександрович

Тема роботи: «Інтеграція Jenkins та Docker для автоматизації розгортання проєктів»

затверджена Наказом університету від 3 березня 2025 р. № 79/с

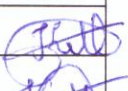
Термін здачі студентом закінченої роботи 9 червня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):
озробка та практична реалізація системи автоматизованого розгортання програмних проєктів на базі інтеграції Jenkins та Docker.

Обладнання та програмне забезпечення:

1. Jenkins
2. Docker
3. Vagrant
4. Віртуальні машини на базі VirtualBox

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Фальковський І. Г.	03.03.2025 	03.03.2025 
2	Фальковський І. Г.	20.03.2025 	20.03.2025 
3	Фальковський І. Г.	11.04.2025 	11.04.2025 

Студент  Максим КОС

Керівник  Олена ГОЛОВНЯ

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	03.03.2025	
2	Аналіз технологій	20.03.2025	
3	Проектування архітектури	10.04.2025	
4	Налаштування середовища використовуючи Vagrant	20.04.2025	
5	Налаштування Jenkins	25.04.2025	
6	Створення pipeline та інтеграція з Docker	03.05.2025	
7	Тестування створеного середовища	20.05.2025	
8	Аналіз отриманих результатів	25.05.2025	
9	Оформлення пояснювальної записки	01.06.2025	
10	Підготовка презентації	03.06.2025	
11	Захист	25.06.2025	

Студент  Максим Кос

Керівник  Олена Головня

АНОТАЦІЯ

Пояснювальна записка до кваліфікованої роботи складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 60 сторінках друкованого тексту. Пояснювальна записка має сторінки додатків. Список використаних джерел містить 20 найменувань і займає 2 сторінки. В роботі наведено 46 рисунків та 4 таблиці. Загальний обсяг роботи - 67 сторінки.

Метою кваліфікаційної роботи є розробка та практична реалізація системи автоматизованого розгортання програмних проєктів на базі інтеграції Jenkins та Docker, що дозволить значно оптимізувати процеси безперервної інтеграції та безперервної доставки (CI/CD), підвищити швидкість, надійність і відтворюваність розгортання застосунків.

У кваліфікаційній роботі визначено основні завдання для створення системи автоматизованого розгортання та проаналізовано базові концепції CI/CD з використанням контейнеризації. Розглянуто архітектурні особливості Jenkins та Docker, а також їхню взаємодію.

Показано розробку інтегрованого рішення, включаючи налаштування Jenkins-сервера та інструментів Docker. Для демонстрації створено тестовий програмний проєкт, для якого реалізовано та налаштовано Jenkins-пайплайни для автоматичної збірки, тестування та розгортання в Docker-контейнерах.

Ключові слова: JENKINS, JENKINSFILE, DOCKER, ДЕПЛОЙ, DOCKERFILE, VAGRANT, VAGRANTFILE, MYSQL, SSH, JDK, NGINX, GITLAB, API, КОНТЕЙНЕР, ОБРАЗ, SLAVE-АГЕНТ, MASTER-АГЕНТ.

					ККІКБ.КР.Б – 123 – 05 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Інтеграція Jenkins та Docker для автоматизації розгортання проєктів	Лім.	Арк.	Аркушів
Розроб.		Кос М. О.		20.06.25				
Керівник		Головня О. С.		20.06.25			4	67
Реценз.		Миколайчук В. В.		20.06.25		ФІКТ, Гр. КІ-21-2		
Н. контр.		Головня О. С.		20.06.25				
Затверд.		Єфіменко А.А.		20.06.25				

ANNOTATION

The explanatory note to the qualification work consists of a list of conventional abbreviations, an introduction, three chapters, conclusions, a list of references and an appendix.

The text part is set out on 60 pages of printed text. The explanatory note contains pages of appendices. The list of references includes 20 titles and occupies 2 pages. The work contains 46 figures and 4 tables. The total volume of the work is 67 pages.

The purpose of the qualification work is to develop and implement a system for automated deployment of software projects based on the integration of Jenkins and Docker, which will significantly optimise the processes of continuous integration and continuous delivery (CI/CD), increase the speed, reliability and reproducibility of application deployment.

The qualification work identifies the main tasks for creating an automated deployment system and analyses the basic concepts of CI/CD using containerisation. The architectural features of Jenkins and Docker, as well as their interaction, are considered.

The development of an integrated solution is shown, including the configuration of the Jenkins server and Docker tools. For demonstration purposes, a test software project has been created for which Jenkins pipelines have been implemented and configured for automatic build, testing, and deployment in Docker containers.

Keywords: JENKINS, JENKINSFILE, DOCKER, DOCKERFILE, DEPLOY, VAGRANT, VAGRANTFILE, MYSQL, SSH, JDK, NGINX GITLAB, API, CONTAINER, IMAGE, SLAVE-AGENT, MASTER-AGENT.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ТА ОГЛЯД ТЕХНОЛОГІЙ РОЗГОРТАННЯ ПРОЄКТІВ	10
1.1. Інструмент для безперервної інтеграції Jenkins.....	10
1.2. Контейнеризація в Docker.....	17
1.3. Керування віртуальними машинами на базі Vagrant.....	23
1.4. Особливості інтеграції проаналізованих технологій для побудови системи автоматизації розгортання проєктів	26
Висновки до розділу 1	27
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ПРОЄКТІВ	28
2.1. Створення схеми та мережі віртуальних машин.....	28
2.2. Опис проєкту	29
2.3. Файл середовища.....	30
2.4. Основні кроки побудови системи.....	32
2.4. Вибір версій образів та програмного забезпечення.....	35
2.5. Інші технології та інструменти	36
Висновки до розділу 2.	38
РОЗДІЛ 3. НАЛАШТУВАННЯ ТА ДОСЛІДЖЕННЯ РОБОТИ СИСТЕМИ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ПРОЄКТІВ	39
3.1. Створення віртуальних машин та налаштування інструментів.....	39
3.2. Налаштування Jenkins та Docker	41
3.3. Тестування роботи інфраструктури	48
Висновки до розділу 3.	56
ВИСНОВКИ.....	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	59
ДОДАТКИ	61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

ВМ – віртуальна машина

JDK – Java Development Kit

SSH – Secure Shell Protocol

CI/CD – Continuous Integration/Continuous Deployment

IP – Internet Protocol

AWS – Amazon Web Services

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

Актуальність теми. Актуальність інтеграції Jenkins та Docker для автоматизації розгортання проєктів зумовлена нагальною потребою сучасного ІТ у швидкому та надійному виведенні продуктів на ринок. Ця комбінація дозволяє автоматизувати весь життєвий цикл розробки, усуваючи ручні помилки та забезпечуючи відтворюваність розгортання завдяки стандартизованим Docker-контейнерам. Це в свою чергу значно прискорює цикл розробки та розгортання, оптимізуючи використання інфраструктури та мінімізуючи людський фактор.

Метою кваліфікаційної роботи є розробка та практична реалізація системи автоматизованого розгортання програмних проєктів на базі інтеграції Jenkins та Docker, що дозволить значно оптимізувати процеси безперервної інтеграції та безперервної доставки (CI/CD), підвищити швидкість, надійність і відтворюваність розгортання застосунків.

Визначена мета обумовлює наступні завдання:

- **Проаналізувати сучасні підходи до CI/CD** та виявити ключові переваги використання Jenkins та Docker у контексті автоматизації розгортання.
- **Дослідити архітектурні особливості Jenkins та Docker**, а також механізми їх взаємодії для побудови ефективних конвеєрів розгортання.
- **Розробити архітектуру інтегрованого рішення**, що включатиме Jenkins-сервер, Docker-образ для збірки/розгортання та репозиторій вихідного коду проєкту.
- **Створити тестовий програмний проєкт** (наприклад, простий веб-додаток), який буде використовуватися для демонстрації автоматизованого розгортання.
- **Налаштувати Jenkins** для автоматичної збірки, тестування та створення Docker-образів тестового проєкту.
- **Реалізувати автоматизоване розгортання Docker-контейнерів** на цільовому сервері за допомогою Jenkins-пайплайнів.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

– **Провести тестування та оцінку ефективності** розробленої системи автоматизованого розгортання за показниками часу деплою, надійності та простоти конфігурації.

Об’єктом дослідження є системи безперервної інтеграції та безперервного розгортання (CI/CD).

Предметом дослідження є процеси та інструменти інтеграції Jenkins та Docker з метою автоматизації розгортання програмних проєктів.

У процесі виконання кваліфікаційної роботи було розроблено систему автоматизованого розгортання проєктів на базі Jenkins, Docker, а також середовища віртуалізації Vagrant. Система розроблена відповідно до обмежень наявного обладнання, та водночас дозволяє продемонструвати автоматизований цикл розгортання: від збірки до розгортання в ізольованих контейнерах, імітуючи реальне серверне середовище.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ТА ОГЛЯД ТЕХНОЛОГІЙ РОЗГОРТАННЯ ПРОЄКТІВ

1.1. Інструмент для безперервної інтеграції Jenkins

Відомості про Jenkins. Для розробки великих проєктів на яких працює багато розробників, потрібно швидко вводити, тестувати та розгортати новий функціонал. Для спрощення даних процесів використовуються інструменти CI/CD (Continuous Integration and Continuous Delivery). Існує декілька інструментів для цього, найпопулярніші з них: GitLab CI/CD, CircleCI та Jenkins.

Jenkins є рішенням з відкритим кодом, яке має сервер автоматизації для забезпечення безперервної інтеграції та безперервної доставки (CI/CD), автоматизуючи певні етапи розробки ПЗ, такі як тестування, збірка та розгортання[2].

Організації можуть використовувати Jenkins для автоматизації та прискорення процесу розробки програмного забезпечення. Jenkins охоплює різноманітні операції життєвого циклу розробки, такі як збірка, розгортання, документування, тестування, пакування, розгортання в тестовому середовищі, статичний аналіз тощо.

Jenkins - це відомий інструмент безперервної інтеграції, розроблений спочатку компанією Hudson, перш ніж він став доступним на ринку з відкритим програмним кодом. Hudson був створений Косуке Кавагучі в 2004 році під час роботи в Sun Microsystems (придбана Oracle). Після того, як Oracle придбала Sun Microsystems у 2010 році, між Oracle та спільнотою Hudson виникли розбіжності щодо використовуваної інфраструктури.

У 2011 році громада Хадсона одноголосно прийняла рішення про зміну назви проєкту з "Hudson" на "Jenkins. Пізніше проєкт Hudson було передано Фонду Eclipse, і роботи над ним більше не ведуться. Наразі розробка Jenkins адмініструється як проєкт з відкритим вихідним кодом під керівництвом CD Foundation, ініціативи Linux Foundation.

На даний момент Jenkins широко використовується програма з більш ніж 300 000 інсталяцій по всьому світу, і їхня кількість щодня зростає.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

Використання Jenkins. Популярність програмного забезпечення Jenkins пояснюється його здатністю відстежувати та контролювати повторювані дії, які виникають протягом розробки проекту. Наприклад, якщо ваша команда працює над проектом, Дженкінс буде постійно тестувати ваші збірки і попереджати вас про будь-які помилки на ранній стадії процесу. Основні варіанти використання включають в себе:

1. Розгортання коду у продуктивне середовище.

Якщо всі тести, розроблені для функції або гілки релізу, є зеленими, Jenkins або інша система CI може автоматично публікувати код на стадії розробки або виробництва. Це часто називають безперервним розгортанням. Зміни вносяться до того, як можна буде побачити дію злиття. Це можна зробити в динамічному середовищі виконання. Потім код розподіляється в центральну систему підготовки, систему препродакшену або навіть у виробниче середовище, коли вони об'єднуються.

2. Включення автоматизації завдань.

Ще одна можливість використання Jenkins - автоматизація робочих процесів і завдань. Наприклад, коли розробник працює в різних середовищах, йому потрібно встановити або оновити елемент у кожному з них. Якщо інсталяція або оновлення вимагає більше 100 кроків, виконання цього вручну може призвести до помилок. Замість цього можна записати всі кроки, необхідні для завершення дії, у Jenkins. Це займе менше часу і спростить завершення інсталяції або оновлення.

3. Скорочення часу, необхідного для відлагодження коду.

Jenkins - це інформаційна система, яка може взаємодіяти з іншими інструментами DevOps і повідомляти користувачів, коли запит на злиття готовий до злиття. Зазвичай це відбувається, коли всі тести пройдені і всі інші умови виконані. Крім того, запит на злиття може вказувати на різницю в покритті коду тестами. Jenkins скорочує час, необхідний для розгляду запиту на злиття, вдвічі. Покриття коду визначається кількістю рядків коду в компоненті і тим, скільки з них виконується. Jenkins підтримує прозорий процес розробки серед членів команди, скорочуючи час, необхідний для перевірки коду.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

4. Сприяє безперервній інтеграції.

Перш ніж випустити зміни до програмного забезпечення, це ПЗ повинно пройти через низку складних процесів. Конвеєр Jenkins забезпечує взаємозв'язок багатьох подій і завдань у певній послідовності для забезпечення безперервної інтеграції. Він має набір плагінів, які спрощують інтеграцію та впровадження конвеєрів безперервної інтеграції та доставки. Головною особливістю конвеєра Jenkins є те, що кожне завдання або робота залежить від іншого завдання або роботи.

5. Збільшення покриття коду.

Jenkins та інші CI-сервери можуть перевіряти код для збільшення тестового покриття. Покриття коду покращується в результаті тестів. Це заохочує членів команди бути відкритими та підзвітними. Результати тестів представлені в конвеєрі збірки, що гарантує, що члени команди дотримуються інструкцій. Як і перегляд коду, комплексне покриття коду гарантує, що тестування є прозорим процесом для всіх членів команди.

6. Підвищення ефективності розробки.

Jenkins значно підвищує ефективність процесу розробки. Наприклад, код командного рядка може бути перетворений на натискання кнопки у графічному інтерфейсі за допомогою Jenkins. Це можна зробити, інкапсулювавши скрипт у завдання Jenkins. Завдання Jenkins можна параметризувати, щоб дозволити користувачеві вносити свої корективи. В результаті можна заощадити сотні рядків коду.

7. Спрощення аудиту.

Коли запускаються Jenkins-завдання вони збирають консольний вивід з параметрів stdout та stderr. Це робить усунення несправностей за допомогою Jenkins надзвичайно простим. Jenkins дозволяє оцінити час виконання і знайти найповільніший крок за допомогою плагіна time stamper, що дозволяє налаштувати продуктивність кожної операції.

8. Використання Slack для синхронізації.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Основним варіантом використання Jenkins є його сумісність зі Slack. Централізована комунікаційна платформа є обов'язковою для великих команд, і однією з найпопулярніших платформ для цієї мети є Slack. Jenkins може бути інтегрований з ним, і це дозволяє ділитися з іншими наступними даними, такими як запуснені дії, час їх виконання, імена користувачів та результати.

Як працює Jenkins. Він може працювати як сервер на різних операційних системах, включаючи Windows, macOS, інші Unix-подібні ОС, особливо Linux. Jenkins працює на Oracle JRE або OpenJDK і потребує віртуальної машини Java версії 8 або новішої версії [1].

Нещодавно Jenkins було модифіковано для роботи в контейнерах Docker. Крім того, він доступний у вигляді архіву Web Application Resource (WAR), інсталяційних пакетів для основних операційних систем, пакунків Homebrew, образів Docker та джерельного коду.

Програмний код рішення написаний переважно на Java, з деякими файлами на Groovy, Ruby та Antlr. WAR в Jenkins може запускатися як окремий модуль або сервлет на сервері Java-додатків, наприклад, Tomcat. В обох випадках він генерує веб-інтерфейс користувача і приймає запити до свого REST API. При першому запуску Jenkins генерує обліковий запис адміністратора з довгим унікальним паролем, який можна ввести на першому сайті інсталяції для розблокування. Важливо мати на увазі, що образи Jenkins доступні лише для читання в онлайн-репозиторії Docker Hub.

Далі, вміст Jenkins зберігається локально у файлі Jenkins у вигляді простого тексту. Файл Jenkins має синтаксис фігурних дужок, що нагадує JSON. Кроки конвеєра беруться у фігурні дужки і вказуються як команди з аргументами. Потім сервер Jenkins читає файл Jenkins і виконує завдання, переміщуючи код із зафіксованого програмного коду до часу виконання у виробництві.

Дженкінс-файли можна створювати за допомогою графічного інтерфейсу користувача (GUI) або шляхом ручного написання коду. Він автоматизує кожен етап процесу розробки, від інтеграції до розгортання. Щоразу, коли розробник вносить зміни до сховища джерельного коду, він виконує збірку.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Зазвичай фіксація відбувається у гілці розробки. Перш ніж випустити збірку у виробництво, Jenkins може розгорнути її в середовищі, яке дозволяє провести будь-яке необхідне користувацьке тестування (UAT). Для досягнення безперервної доставки (CD) ці UAT-тести можуть бути автоматизовані за допомогою такого інструменту, як Selenium.

Якщо тести пройдені, можна об'єднати код в основну гілку, де доступно збирання «золотої» збірки (golden build) і негайний її запуск у виробництво без будь-якого ручного втручання. Наприклад компанії Amazon, Facebook і Google досягли 100% безперервної доставки і можуть розгортатися у виробництво багато разів на день.

Види проєктів в Jenkins. У Jenkins є декілька основних типів проєктів (job types), які можна використовувати для налаштування автоматизації.

У Jenkins існують різноманітні типи проєктів, кожен з яких має своє призначення. Freestyle Project є базовим та найпростішим, дозволяючи виконувати такі завдання, як запуск скриптів або завантаження артефактів, і підтримує інтеграцію з системами контролю версій. Для створення складних CI/CD конвеєрів використовується Pipeline Project, де за допомогою Jenkinsfile визначається послідовність етапів та кроків, включаючи умови та паралельне виконання. Коли мова йде про багатогілкові репозиторії, Multibranch Pipeline Project автоматично створює конвеєри для кожної гілки. Для організації завдань у Jenkins можна використовувати Folder, що дозволяє групувати проєкти та застосовувати спадкові налаштування. Multiconfiguration Project (Matrix Project) ідеально підходить для запуску одного завдання в численних конфігураціях, наприклад, для тестування на різних платформах. Для моніторингу зовнішніх процесів передбачений External Job. Аби уникнути дублювання коду в конвеєрах, є Pipeline with Shared Libraries. Для автоматизації роботи з великими організаціями на GitHub та Bitbucket існують GitHub Organization та Bitbucket Team/Project відповідно, які сканують репозиторії та створюють для них конвеєри. Нарешті, External Plugin-Based Project Types розширюють можливості Jenkins, дозволяючи інтегрувати специфічні рішення, як-

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

от Blue Ocean для візуалізації конвеєрів або Docker Pipeline для роботи з контейнерами.

Кожен із цих типів відповідає певним вимогам, залежно від складності та архітектури проєкту.

Ключові особливості. Jenkins легко налаштовувати та кастомізувати. Jenkins має багато плагінів, які надають йому велику універсальність. Він миттєво надає код, генерує звіт після розгортання, висвітлює помилки в коді або тестах, а також виявляє і вирішує різні проблеми майже в режимі реального часу. Він також ідеально підходить для інтеграції, оскільки все це робиться автоматично. Існує також велика спільнота підтримки.

Основні функції наведено нижче:

1. Різноманітність плагінів.
2. Просте встановлення та налаштування
3. Доступний з відкритим вихідним кодом
4. Значний ступінь розширюваності
5. Безпека на основі серверів

В таблиці 1.1 наведено порівняння Jenkins за основними критеріями з аналогами .

Табл. 1.1

Порівняння CI/CD інструментів.

<i>Критерій</i>	<i>Jenkins</i>	<i>GitLab CI/CD</i>	<i>GitHub Actions</i>	<i>Azure DevOps</i>
Тип ліцензії	Відкритий вихідний код (MIT License)	Частково відкритий (Community Edition), Комерційний (Enterprise Edition)	Безкоштовний для публічних репозиторіїв, платний для приватних (за хвилини)	Безкоштовний план (для 5 користувачів/30 годин на місяць), комерційний
Тип розгортання	Локальний (On-premise), Хмарний (через VM/контейнери)	Локальний (Self-managed), Хмарний (SaaS)	Хмарний (SaaS)	Хмарний (SaaS), Локальний (Azure DevOps Server)
Легкість налаштування та використання	Вимагає знань налаштування JVM, плагінів, пайплайнів. Великий поріг входу.	Конфігурація в YAML-файлі (.gitlab-ci.yml) в репозиторії. Інтуїтивно зрозумілий.	Конфігурація в YAML-файлі (.github/workflows/*.yaml) в репозиторії. Великий маркетплейс екшенів.	Може бути складним для першого ознайомлення, але гнучкий. UI та YAML.

Продовження табл. 1.1

Можливості інтеграції	Величезна екосистема плагінів для будь-яких інструментів (Git, SVN, Docker, Kubernetes, AWS, Azure, GCP, Jira, Slack тощо).	Тісна інтеграція з усіма компонентами GitLab. Добре інтегрується з Kubernetes, Docker.	Нативна інтеграція з GitHub. Великий маркетплейс готових "екшенів" для інтеграції з хмарними провайдерами, сторонніми сервісами.	Нативна інтеграція з усією екосистемою Azure. Широка підтримка інших інструментів (Git, Kubernetes, Jenkins, Chef, Puppet тощо).
Підтримка мов програмування та фреймворків	Завдяки плагінам підтримує практично будь-яку мову та фреймворк.	Підтримує будь-які мови та фреймворки завдяки використанню Docker-образів.	Підтримує будь-які мови та фреймворки завдяки використанню "екшенів" та Docker-образів.	Підтримує будь-які мови та фреймворки, тісна інтеграція з .NET, Java, Node.js, Python.
Масштабованість	Може бути дуже масштабованим з master-agent архітектурою, але вимагає значних зусиль для управління.	Масштабується горизонтально за рахунок Runner'ів. Дуже добре для великих команд та проектів.	Хмарна архітектура дозволяє легко масштабуватися без зусиль з боку користувача.	Хмарна архітектура Azure забезпечує дуже високу масштабованість.
Цінова політика	Безкоштовний але вимагає витрат на інфраструктуру та підтримку.	Є безкоштовний план (GitLab.com). Приватні репозиторії з обмеженими хвилинами. Платні тарифи для збільшення ресурсів та функціоналу.	Безкоштовний для публічних: 2000 хвилин/місяць для приватних репозиторіїв. Далі - плата за хвилини.	Безкоштовний план: 5 користувачів безкоштовно. 1 приватний хостинг агент (1800 хв/міс) та 1 паралельна робота для публічних проектів. Далі - оплата за користувачів та паралельні роботи.
Особливості/Унікальні переваги	Максимальна кастомізація, контроль над середовищем, величезна спільнота. Ідеальний для складних, унікальних робочих процесів.	Все в одному місці (Git, CI/CD, Wiki, Registry, Issues). DAST/SAST.	Величезний маркетплейс готових "будівельних блоків". Легкість створення власних екшенів. Глибока інтеграція з GitHub.	Інтегроване управління проектами, репозиторіями, артефактами, тест-планами. Тісна інтеграція з Azure Cloud.
Недоліки/Обмеження	Вимагає спеціаліста для адміністрування та вирішення проблем. Необхідність власної інфраструктури.	Найкраще працює в екосистемі GitLab. Може бути "перевантаженим" для дуже простих проектів.	Може бути дорогим для дуже активних приватних репозиторіїв. Залежність від GitHub.	Може мати велику криву навчання для новачків. Найкраще працює в екосистемі Microsoft.

Отже, Jenkins - це рішення з відкритим вихідним кодом, що забезпечує безперервної інтеграції та безперервної доставки (CI/CD). Він здатний відстежувати та контролювати повторювані дії, які виникають протягом розробки проекту. Інструмент використовується для наступних завдань:

- розгортання коду у виробництво.
- включення автоматизації завдань.
- скорочення часу, необхідного для відлагодження коду.
- забезпечення безперервної інтеграції.
- збільшення покриття коду.

Jenkins є поширеним та легким в налаштуванні. Має багато плагінів для різних задач, миттєво надає код та генерує звіти після розгортання.

1.2. Контейнеризація в Docker

Огляд інструмента Docker. Дуже часто потрібно, запустити якийсь інструмент під певною операційною системою, або потрібно виконувати розробку під іншу платформу. Спростити ці завдання може технологія контейнеризації, наприклад Docker.

Docker - це програмна платформа, яка дозволяє швидко створювати, тестувати та розгортати додатки. Docker пакує програмне забезпечення у стандартизовані одиниці, які називаються контейнерами, що містять все необхідне для запуску, включаючи бібліотеки, системні інструменти, код та середовище виконання. Використовуючи Docker, можна швидко розгортати та масштабувати додатки в будь-якому середовищі і бути впевненими, що код буде працювати [3].

Оскільки контейнери Docker ізольовані один від одного та від хост-системи, їм притаманний певний рівень безпеки за замовчуванням. Однак ця ізоляція не є абсолютною. Безпека Docker базується на цілісній системі нульової довіри, яка охоплює час виконання, збірку та оркестрування контейнерів.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

Docker дозволяє розробникам отримати доступ до цих вбудованих можливостей контейнеризації за допомогою простих команд і автоматизувати їх за допомогою інтерфейсу прикладного програмування (API), що заощаджує роботу.

Docker є найпоширенішим інструментом контейнеризації з часткою ринку 82,84%. Docker настільки популярний сьогодні, що терміни «Docker» і «контейнери» досить часто використовуються як взаємозамінні. Однак перші технології, пов'язані з контейнерами, були доступні за багато років - навіть десятиліть - до того, як Docker був публічно випущений з відкритим вихідним кодом у 2013 році [4].

Зокрема, у 2008 році в ядро Linux було впроваджено LinuXContainers (LXC), що повністю уможливило віртуалізацію для одного екземпляра Linux. Хоча LXC використовується і сьогодні, доступні новіші технології ядра Linux. Наприклад, Ubuntu, сучасна операційна система Linux з відкритим кодом, також надає таку можливість (LXD-контейнери).

Переваги використання контейнерів для розгортання застосунків. Контейнерна технологія пропонує всі функції та переваги віртуальних машин (ВМ), включаючи ізоляцію додатків та економічно ефективне масштабування. Це досягається завдяки наступним особливостям контейнерів:

- *Менша вага.* На відміну від віртуальних машин, контейнери не несуть корисного навантаження цілого екземпляра ОС та гіпервізора. Вони включають лише процеси та залежності ОС, необхідні для запуску коду. Розміри контейнерів вимірюються в мегабайтах (на відміну від гігабайтів для деяких віртуальних машин), вони краще використовують апаратну потужність і швидше запускаються.
- *Покращена продуктивність.* Контейнерні додатки можна написати один раз і запускати будь-де. У порівнянні з віртуальними машинами, контейнери швидше і простіше розгортати, забезпечувати і перезапускати.
- *Більша ефективність.* Завдяки контейнерам розробники можуть запускати в кілька разів більше копій програми на одному обладнанні, ніж у

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

випадку з віртуальними машинами. Така ефективність може зменшити витрати на хмарні сервіси.

Історія появи Docker. Компанія dotCloud була заснована Камелем Фунаді, Соломоном Хайксом та Себастьяном Палем під час стартап-інкубатора Y Combinator Summer 2010, запущена у 2011 році та перейменована на Docker Inc. у 2013 році. Стартап також був одним з 12 стартапів першої когорти Founder's Den. Хайкс розпочав проект Docker у Франції як внутрішній проект в рамках dotCloud, компанії, що надає платформу як послугу.

Docker дебютував для публіки в Санта-Кларі на конференції PyCon у 2013 р. Він був випущений з відкритим вихідним кодом у березні 2013 р. На той час він використовував LXC як середовище виконання за замовчуванням. Через рік, з виходом версії 0.9, Docker замінив LXC на власний компонент libcontainer, написаний на мові програмування Go [6].

Принцип роботи. Docker працює, надаючи стандартний спосіб запуску коду. Docker - це операційна система для контейнерів. Подібно до того, як віртуальна машина віртуалізує серверне обладнання (усуває необхідність безпосереднього керування ним), контейнери віртуалізують операційну систему сервера. Docker встановлюється на кожному сервері і надає прості команди, за допомогою яких можна створювати, запускати або зупиняти контейнери.

Контейнери стали можливими завдяки можливостям ізоляції та віртуалізації процесів, вбудованим у ядро Linux. Ці можливості включають групи керування (Cgroups) для розподілу ресурсів між процесами та простори імен для обмеження доступу або видимості процесу до інших ресурсів або областей системи.

Порівняння Docker з Podman та LXC наведено в таблиці 1.2

Табл. 1.2.

Порівняння Docker з аналогами.

<i>Критерій</i>	<i>Docker</i>	<i>Podman</i>	<i>LXC (Linux Containers)</i>
<i>Основна функція</i>	Створення, запуск, управління контейнерами.	Створення, запуск, управління	Низькорівнева контейнеризація на

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

Основна функція	Включає клієнт, демон, образи, контейнери.	контейнерами. Альтернатива Docker.	рівні ОС Linux.
Архітектура	Клієнт-сервер (з демоном): Docker CLI спілкується з dockerd (демоном).	Запускає контейнери як дочірні процеси.	Безпосередньо використовує функції ядра Linux (namespaces, cgroups).
Безпека	Демон зазвичай працює з root-правами. Потрібна обережна конфігурація.	За замовчуванням підтримує "rootless" (без root-прав) контейнери.	Забезпечує високий рівень ізоляції на рівні ОС.
Сумісність CLI	Стандартний Docker CLI.	Більшість команд Docker працюють, якщо замінити docker на podman.	Свій власний CLI (lxc-* команди).
Керування образами	Вбудований механізм docker build.	Використовує Buildah для створення образів (хоча Podman може викликати Buildah).	Не має прямої інтеграції зі створенням "образів" у Docker-стилі.
Багатоконтейнерні додатки	Docker Compose: Популярний інструмент для визначення та запуску багатоконтейнерних додатків.	Podman Compose: Аналог Docker Compose, сумісний з його YAML-файлами.	Не має вбудованої підтримки для багатоконтейнерних "композицій".
Портативність	Образи Docker є стандартом і працюють на будь-якій системі з Docker Engine (Linux, Windows, macOS).	Образи сумісні з Docker. На Windows/macOS використовує віртуальні машини.	Переважно для Linux, менш портативний, ніж Docker.
Використання	Локальна розробка, тестування, продакшн (для одиничних контейнерів або з оркестрацією).	Локальна розробка, безпечні середовища, замість Docker у Linux.	Системні контейнери, легковесна віртуалізація ОС.
Переваги над іншими	Найбільша екосистема, стандартизація, легкість початку, Docker Desktop.	Більша безпека (без демона, rootless), сумісність з Docker CLI.	Глибока інтеграція з ядром Linux.
Недоліки / обмеження	Демон, потенційні проблеми безпеки (хоча є rootless mode).	Менш зріла екосистема на Windows/macOS.	Менша портативність, відсутність високорівневих інструментів оркестрації.

Головні особливості Docker є наступними.

- *Платформа з відкритим вихідним кодом.* Одним з найважливіших аспектів платформи з відкритим кодом є можливість вибору технології для виконання завдання. Якщо користувачеві потрібне легке, чисте середовище для тестування, движок Docker може бути корисним для індивідуальних розробників. Особливо враховуючи наявність безкоштовної версії Docker Community Edition (docker-ce).
- *Простий спосіб доставки додатків.* Це одна з найважливіших особливостей Docker, яка дозволяє швидко і просто налаштувати систему. Завдяки цій функціональності коди можна вводити швидко і легко. Оскільки Docker використовується в багатьох місцях, інфраструктура не пов'язана з середовищем додатків. Як уже згадувалося вище, контейнери швидші та споживають менше ресурсів, ніж віртуальні машини. Це робить їх ідеальними для розгортання додатків у різних середовищах, оскільки їх можна легко переміщувати між серверами, хмарними платформами та локальними середовищами розробки.
- *Швидкий та ефективний життєвий цикл розробки.* Docker прагне скоротити час між написанням коду та його тестуванням, розгортанням і використанням. Його мета - зробити ваші програми портативними, простими у створенні та зручними для спільної роботи.
- *Розподіл обов'язків.* Docker призначений для покращення узгодженості, гарантуючи, що середовище, в якому ваші розробники пишуть код, відповідає середовищу, в якому розгортаються ваші додатки. Це зменшує ймовірність того, що «працював у розробці, а тепер проблема в експлуатації».
- *Сервіс-орієнтована архітектура.* Docker також підтримує сервіс-орієнтовані та мікросервісні архітектури. Docker припускає, що кожен контейнер запускає лише одну програму або процес. Це заохочує використання розподіленої моделі додатків, в якій додаток або сервіс представлений набором взаємопов'язаних контейнерів. Це значно спрощує розповсюдження, масштабування, налагодження та перевірку ваших додатків.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

- *Ізоляція та безпека.* Функція управління безпекою Docker зберігає секрети в Docker Swarm. Контейнери Docker забезпечують високий рівень ізоляції між різними додатками, не дозволяючи їм взаємодіяти або впливати один на одного. Це робить їх безпечною та стабільною платформою для запуску декількох додатків на одному хості.

- *Масштабованість.* Платформа легко масштабується, оскільки контейнери Docker мають невелику вагу. Портативність Docker також дозволяє легко динамічно керувати робочими навантаженнями, збільшуючи або зменшуючи додатки та сервіси відповідно до бізнес-вимог майже в режимі реального часу. Docker має простий та інтуїтивно зрозумілий інтерфейс командного рядка, що дозволяє легко створювати, керувати та розгортати контейнери. Він також пропонує широкий спектр інструментів та інтеграцій для різних середовищ розробки, що дозволяє легко інтегрувати його в існуючі робочі процеси.

- *Зменшення розміру.* Docker пропонує великий ступінь можливостей для мінімізації розміру розробки. Це пов'язано з тим, що завдяки контейнерам він може зменшити вплив на операційну систему.

- *Керування образами.* Docker використовує образи для зберігання необхідних файлів і залежностей, необхідних для запуску програм. Ці образи створюються за допомогою Docker-файлу, який є набором інструкцій, що вказує Docker, як створити образ. Розробники можуть використовувати ці образи для створення нових контейнерів або оновлення існуючих. Docker також має реєстр під назвою Docker Hub, який є центральним сховищем для зберігання та обміну образами.

- *Робота в мережі.* Docker дозволяє розробникам легко підключати контейнери та сервіси за допомогою вбудованих мережевих можливостей. Він підтримує створення віртуальних мереж і дозволяє розробникам вказувати, які контейнери можуть взаємодіяти один з одним. Це спрощує налаштування та керування складними архітектурами мікросервісів.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

- *Керування томами.* Docker надає систему керування томами, яка дозволяє розробникам зберігати дані, згенеровані контейнерами. Це корисно для додатків, які потребують бази даних або іншого сховища даних. Томи Docker зберігаються на хост-системі і можуть спільно використовуватися контейнерами.

Висновки. Таким чином, контейнеризація є зручним та широковикористовуваним інструментом для розгортання застосунків. Порівняно з віртуальними машинами, віртуальні контейнери мають менші потреби в ресурсах, кращу продуктивність, завдяки чому можна швидко запустити велику кількість контейнерів на одному обладнанні. Було розглянуто платформи контейнеризації Docker, Podman та LXC. Наведено переваги та недоліки кожного.

Однією з найпопулярніших платформ для віртуальних контейнерів є Docker. Docker - це програмна платформа, яка дозволяє швидко створювати, тестувати та розгортати додатки. На відмінно від аналогів він є простим в роботі та має високу портативність, також варто відзначити велику екосистему Docker.

Docker потужний інструмент який значно зменшує час між написанням коду та тестуванням, а також дає можливість швидко та легко масштабувати додаток в разі потреби.

1.3. Керування віртуальними машинами на базі Vagrant

Огляд інструмента Vagrant. Побудувати систему з безперервною інтеграцією можна на фізичних серверах, але часто не раціонально виділяти багато ресурсів на один сервер. Натомість можна використати розгортання декількох ВМ. Для створення та керування даними машинами допоможе інструмент Vagrant.

Vagrant – це інструмент для автоматизації створення, налаштування та управління віртуальними середовищами. Він дозволяє розробникам створювати стандартні та портативні середовища для програмного забезпечення, використовуючи прості текстові файли конфігурації.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

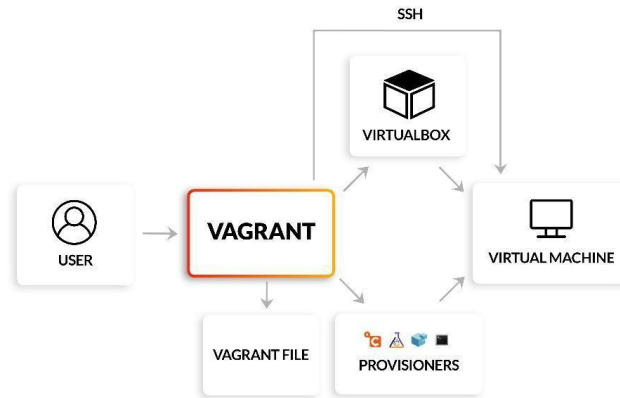


Рис. 1.1. Схема роботи Vagrant.

Vagrant використовує конфігураційний файл (Vagrantfile) для опису параметрів віртуального середовища. Він запускає віртуальні машини через провайдери (наприклад, VirtualBox, Vmware або Docker) і автоматично налаштовує їх за допомогою скриптів або конфігураційних менеджерів (Ansible, Puppet, Chef тощо).

Vagrant використовує провайдерів (наприклад, VirtualBox, Docker, AWS) для створення віртуальних машин та шаблони (box), які є стандартизованими образами цих машин для швидкого розгортання. Також застосовуються скрипти налаштування для автоматизації інсталяції залежностей та конфігурування середовища [5].

Vagrant призначений для створення ізольованих і відтворюваних середовищ розробки, що є ключовим для командної роботи. Завдяки цьому можна тестувати програми на різних конфігураціях систем. Це значно спрощує автоматизацію розгортання навіть дуже складних середовищ і слугує ефективним засобом для навчання інфраструктурних технологій та конфігурацій.

Переваги та недоліки використання. Як і кожний інструмент Vagrant має свої переваги та недоліки. І потрібно враховувати ці параметри для поставленого завдання.

Переваги Vagrant є наступними.

- *Кросплатформеність.* Підтримує роботу на Windows, macOS і Linux.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

- *Автоматизація.* Полегшує налаштування середовищ за допомогою конфігураційного файлу (Vagrantfile).
- *Портативність.* Середовища можна легко перенести між розробниками або командами.
- *Ізольованість.* Кожне середовище повністю ізольоване, що запобігає конфліктам між проектами.
- *Інтеграція з провайдерами:* Підтримує VirtualBox, Vmware, Docker, AWS та інші провайдери.
- *Відтворюваність.* Дає змогу створювати ідентичні середовища для різних користувачів, що покращує консистентність розробки.

Водночас, Vagrant має наступні недоліки.

- *Витрати ресурсів.* Віртуальні машини, які створює Vagrant, можуть бути ресурсоемними.
- *Складність налаштування.* Новачкам може бути важко зрозуміти всі аспекти конфігурації.
- *Швидкість.* Використання віртуальних машин може бути повільніше порівняно з контейнерами без використання віртуальних машин (наприклад, Docker без Vagrant).
- *Застарілість.* З появою легших рішень, як Docker, Vagrant використовується менше в певних сценаріях.
- *Залежність від провайдерів.* Робота залежить від належної роботи обраного провайдера, наприклад, VirtualBox.

Таким чином, Vagrant – зручний інструмент для керування ВМ, який значно полегшує налаштування віртуальних машин під певні задачі. Vagrant призначений для автоматизації створення, налаштування та управління віртуальними середовищами, використовуючи прості текстові файли конфігурації. Для роботи Vagrant може мати інтеграцію з різними провайдерами, використовувати готові шаблони віртуальних машин, скрипти для автоматизації налаштування віртуальних машин.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

1.4. Особливості інтеграції проаналізованих технологій для побудови системи автоматизації розгортання проєктів

Вище було проаналізовано технології, які може бути використано як інструменти для автоматизації розгортання проєктів. Кожна з даних технологій виконує певний функціонал який забезпечує гнучку та швидку розробку. Завдяки даним технологіям можна керувати виділенням ресурсами для кожного сервера. Також Vagrant може керувати Docker контейнерами, що дає змогу швидко запустити якийсь додатковий сервіс.

Отже, ролі технологій в інтеграції з іншими є наступними:

- *Vagrant* - створює та керує середовищами розробки на основі віртуальних машин чи контейнерів.
- *Jenkins* - автоматизує процеси CI/CD (безперервна інтеграція та доставка).
- *Docker* - забезпечує легке створення та ізоляцію контейнерів для додатків.

Кожна з перелічених технологій може застосовується окремо для швидкого, зручного розгортання проєктів з швидкою зміною налаштувань. Використані разом, дані технології можуть створити цікаву інфраструктуру, яка буде ефективною та масштабованою.

При гарно спроектуваній топології та налаштуванні. Можна створити систему яка зменшить витрати часу на розгортання та масштабування проєктів, що в свою чергу позитивно вплине на розробку продукту.

Даний стек технологій не прив'язаний до конкретного додатка чи проєкту. При необхідності можна внести деякі зміни, оскільки інструменти є гнучкими в роботі.

Таким чином, завдяки інтеграції технологій Vagrant, Jenkins та Docker можна створити зручну та ефективну систему розгортання додатків, де кожний інструмент матиме свої обов'язки: Vagrant створюватиме та керуватиме середовищами розробки на основі віртуальних машин або віртуальних контейнерів, Jenkins

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

автоматизуватиме процеси CI/CD, Docker забезпечить легке створення та необхіду ізоляцію контейнерів для додатків.

Висновки до розділу 1

В першому розділу було розглянуто технології Jenkins, Docker, Vagrant. Розглянуто функціонал, історію виникнення та проблеми які вирішують наведені інструменти. Також було зазначено їхні переваги та недоліки. Розглянуто роботу та використання їх разом для забезпечення рішення автоматизації.

Jenkins є інструментом з відкритим вихідним кодом. Виконує: розгортання коду у виробництво, включення автоматизації завдань, зменшення часу для відлагодження коду, безперервні інтеграції, покриття коду тестами.

Програмна платформа Docker, що дозволяє швидко створювати, розгортати та тестувати додатки. Docker має переваги по відношенню до віртуальних машин. Контейнери потребують менше ресурсів, тому що включають лише процеси та залежності ОС, які необхідні для запуску програмного коду. Docker-контейнери швидко та легко розгортаються, що значно впливає на продуктивність, також контейнери є більш ефективними, тому що можна запускати більше копій на одному обладнанні.

Також було розглянуто інструмент Vagrant для автоматизації створення, налаштування та управління віртуальними середовищами. Для роботи Vagrant використовує файли конфігурації, провайдери, шаблони (box) та скрипти автоматичного налаштування.

Завдяки інтеграції перелічених технологій, можна створити зручну та ефективну систему розгортання додатків. Де кожний інструмент матиме певні завдання.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ПРОЄКТІВ

2.1. Створення схеми та мережі віртуальних машин

Перш за все було розроблено топологію та сервіси, які будуть використані. Побудова архітектури є одним з важливих завдань на першому етапі. Для цього потрібно зрозуміти, що необхідно і як це краще досягнути.

На початку проектування було створено архітектурну схему системи (рис. 2.1). В ній вказано основні ВМ, сервіси та зв'язки між ними. Ця схема є простою, проте допоможе зрозуміти загальну топологію. Подібна схема теоретично може бути використана у реальному проєкті, проте зазвичай вона матиме складнішу топологію. Зокрема може бути збільшено кількість серверів бази даних, веб-серверів. Коли на даному сервері з'явиться декілька екземплярів одного сервіса, наприклад серверів з API - тоді доцільним буде використання проксі-сервера, який буде отримувати та розподіляти запити в залежності від налаштування. Таким чином навіть у віртуальному середовищі можна буде нарощувати спроможність серверів в залежності від ситуації. Таким чином можна налаштувати, щоб запит перенаправлявся до іншої фізичної машини.

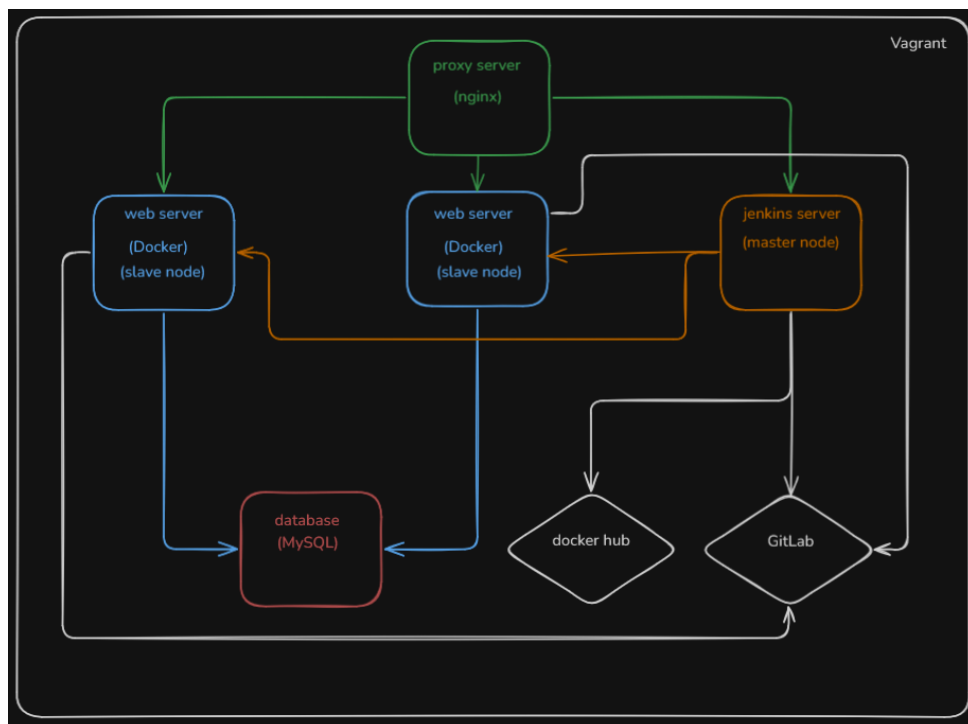


Рис. 2.1. Архітектурна схема автоматизованої системи розгортання.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

Далі було розроблено схему адресації віртуальних машин та сервісів роботи яких вони забезпечують.

Таблиця. 2.1

Адресація серверів та сервісів.

Хост/Віртуальна машина	ІР-адреса	Сервіси/ Технології	Порти сервісів
Основний хост	192.168.56.1	Vagrant	-
jenkins_server	192.168.56.10	Jenkins, Docker	8080
web_server	192.168.56.20	Docker, web-app	8080
web_server	192.168.56.21	Docker, web-app	8080
database_server	192.168.56.30	Mysql 8.0	3306
proxy_server	192.168.56.100	Nginx (proxy)	80, 8080
Host (фізичний ПК)	Динамічна ір-адреса	-	5600, 5680

Отже, було розроблено схему віртуальних машин та запущених сервісів на них. Також було розроблено схему ІР-адресації мережі серверів.

2.2. Опис проєкту

Використання Vagrant. Завдяки Vagrant буде створено мережу з віртуальних машин на кожній з яких буде запущено свій сервіс. Провайдером для віртуальних машин буде виступати VirtualBox (проєкт є локальним).

Сервер Jenkins (master-агент). Наведений сервер матиме запущений сервіс «головного агента» Jenkins (master Jenkins agent). Він буде відповідати за автоматизацію розгортання та доставки додатків. На ньому будуть налаштовані slave-агенти (веб-сервери) та створені завдання в яких будуть виконуватися отримання коду з GitLab. Далі відбудуватиметься створення образу Docker та запуск контейнера на певних slave-агентах. Також передбачається що буде виконуватися завантаження створених Docker образів до реєстру Docker образів.

Веб-сервер (slave-агент). Виконує роль обробки запитів користувачів, надаватиме доступ до веб-додатку. При запуску завдяки vagrant provision матиме встановлений Docker та JDK для обробки команд які буде надавати головний Jenkins сервер. Планується, що таких підпорядкованих серверів може бути

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

декілька. Наприклад одні виконують роль як тестові сервери, інші як сервери для розгортання.

Сервер бази даних. Хост на якому буде запущено базу даних яку зможе використовувати веб-додаток. На даний момент передбачається, що буде використовуватися MySQL. Проте при створенні віртуальної машини, можна буде запустити іншу систему керування змінивши `vagrant provision`.

Проксі-сервер. Даний сервер буде використовувати Nginx [13] та виконувати наступні ролі як проксі та балансувальник навантаження [17]. Тобто проксі-сервер буде приховувати ір-адреси клієнтів та перенаправляти запити між веб-серверами.

Інші сервіси: Варто відзначити використання GitLab та Docker Hub. В проєкті роботи GitLab, який є джерелом для створення та запуску Jenkins завдання та розгортання Docker-контейнера. В свою чергу Docker Hub виступає для зберігання образів різних версій які створюються під час запуску пайплайнів. Це дає змогу зручно та швидко змінити версію додатка на попередню.

Отже, було розглянуто основні компоненти проєкту та описано функціонал кожного сервера та технології які будуть використані. Застосовується Jenkins master та slave агенти (веб-сервери), сервер бази даних MySQL та проксі сервер Nginx. На додачу розглянуто GitLab та Docker Hub які використовуються під час роботи Jenkins завдання.

2.3. Файл середовища

Використання файлу `config.yml` [19] для Vagrant — це поширена практика, хоча це не є вбудованою функціональністю Vagrant, як Vagrantfile. Основна причина його використання полягає у зручності, гнучкості та розділенні конфігурацій.

По-перше, це дозволяє розділити конфігурацію від логіки Vagrantfile. Сам Vagrantfile написаний на Ruby і містить логіку для визначення віртуальних машин, мережі, синхронізованих папок та провізійування. Натомість, `config.yml` дає змогу винести змінні та специфічні налаштування середовища, наприклад, версії

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

програмного забезпечення, імена хостів, IP-адреси, шляхи до файлів, в окремий, легко читабельний файл. Це робить Vagrantfile значно чистішим та легшим для підтримки.

По-друге, *config.yml* забезпечує легку читабельність та редагування. YAML є дуже зрозумілим форматом даних, що дозволяє легко сприймати та змінювати налаштування без необхідності глибоких знань Ruby. Це особливо цінно для команд, де не всі учасники є розробниками Ruby.

По-третє, це спрощує керування різними середовищами. Можна мати різні *config.yml* файли для різних середовищ, наприклад, для розробки чи тестування. Це дозволяє швидко переключатися між конфігураціями, не вносячи змін до самого Vagrantfile. У *config.yml* можна задати різні розміри пам'яті, кількість CPU або навіть різні базові образи для віртуальних машин залежно від конкретного середовища.

Крім того, *config.yml* спрощує середовища з кількома віртуальними машинами. Коли ви працюєте з декількома віртуальними машинами у Vagrant, наприклад, веб-сервер, база даних та кеш-сервер, *config.yml* дозволяє централізовано визначити параметри для кожної VM, уникнувши повторення коду у Vagrantfile. Можна використовувати цикли у Vagrantfile для ітерації по налаштуваннях з *config.yml* та динамічного створення віртуальних машин.

Проте дана система базується на тому, що одній VM відповідає один Vagrantfile.

Тому передбачається, що він буде містити наступні дані:

- IP адреси VM
- назви хоста
- дані для запуску docker-контейнерів
- порти доступу до ресурса
- набір назви скриптів для ініціалізації
- та інші необхідні дані для коректної роботи VM та сервісів

Наведений файл буде запускатися в кожному Vagrant файлі та використовувати потрібні дані.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

2.4. Основні кроки побудови системи

Встановлення необхідних інструментів та створення сценаріїв запуску.

Перш за все потрібно завантажити Vagrant та VirtualBox. Для того, щоб створювати та керувати віртуальними машинами. Після успішного встановлення потрібно виконати наступні кроки для запуску VM:

1. Перейти до директорії в якій будемо створювати проєкт.
2. Створити Vagrantfile використовуючи команду

***vagrant init* <image_name>**

3. Далі потрібно редагувати файл конфігурації VM. Наприклад, змінити назву хоста, вказати кількість VM, зазначити ресурси які будуть доступні віртуальній машині, налаштувати мережні інтерфейси.

Наступним, опціональним кроком, можна вказати провізори. Тобто скрипти, дії, сценарії які будуть виконуватися при створенні VM чи можна буде запустити на VM завдяки Vagrant. В таблиці 2.2 вказані скрипти які використовуються для ініціалізації VM.

4. Для цього можна використати команду

***vagrant provision* –with-provision <provision_name>**

5. Далі можна запускати скрипт для створення VM. Використовуючи команду ***vagrant up***.
6. Щоб підключитися до машини можна використати ***vagrant ssh*** та працювати безпосередньо на ній.

Якщо потрібно завершити роботу VM вводиться команда ***vagrant halt***. Часто потрібно переглянути всі VM якими керує vagrant та можливо знищити. Для цього потрібно використати відповідно дві команди ***vagrant global-status*** та ***vagrant destroy* <vm_id>**

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Скрипти для автоматичного налаштування ВМ.

Назва налаштування середовища	Назва скрипта	Застосування
install-jenkins	install-jenkins.sh	Для встановлення Jenkins та інших залежностей
install-docker	install-docker.sh	Встановлення та налаштування Docker
init-jenkins-agent	init-jenkins-agent.sh	Встановлення JDK
set-bind-address	set-bind-address.sh	Зміна налаштувань mysql для віддалених підключень
install-mysql	install-mysql.sh	Встановлення mysql
install-nginx	install-nginx.sh	Встановлення Nginx
backup-jenkins.sh	backup-jenkins.sh	Створення архіву Jenkins директорії

Налаштування Jenkins. Після успішного підняття віртуальних машин та завантаження потрібного ПЗ завдяки скриптам ініціалізації. Далі потрібно провести базове налаштування Jenkins. Основні налаштування включають наступні кроки:

1. Для отримання доступу до Jenkins потрібно відкрити браузер за адресою `<ip_private_jenkins_vm>:8080`. Для входу потрібно використати адміністративний пароль, що знаходиться в за шляхом `/var/jenkins/`.
2. Далі буде запропоновано створити користувача-адміністратора.
3. Після створення користувача буде доступно вибрати потрібні плагіни Jenkins. В подальшому можна буде видаляти чи інсталювати плагіни.
4. Тепер можна створити просте завдання в Jenkins для перевірки роботи

Щоб отримувати програмний код з GitHub потрібно виконати наступне:

1. Встановити плагін GitHub Integration Plugin або Git Plugin.
2. Далі потрібно створити токен GitHub.
3. Додати облікові дані для підключення Jenkins до GitHub.
4. Також для спрацювання тригера потрібно створити Webhook у GitHub.
5. Тепер можна створювати Jenkins job та застосувати тригер GitHub hook trigger for GITScm polling.

6. Для перевірки роботи потрібно зробити коміт зі змінами та завантажити це на GitHub.

Також, щоб розгорнути отриманий код на сервері потрібно налаштувати на Jenkins підключення до сервера через SSH протокол. Для цього потрібно виконати наступні дії:

1. Встановити плагін Publish over SSH.
2. Налаштувати облікові дані для підключення до сервера.
3. Застосувати цей плагін в потрібному Jenkins job та вказати команди які потрібно виконати на сервері.

Це основні кроки з налаштування Jenkins які потрібно виконати, проте потрібно зазначити також написання пайплайнів який буде знаходитися в репозиторії та виконуватися коли будуть отримані зміни в репозиторію.

Розгортання проекту завдяки Docker. Важливим є розгляд Docker для запуску контейнера з отриманим з GitHub кодом. Для цього перш за все потрібний встановлений Docker. Далі потрібний коректний Dockerfile який буде використовувати потрібний базовий образ, що буде обробляти код також потрібно відкрити необхідні порти, щоб мати доступ з мережі віртуальних машин. Для коректної роботи контейнера знадобиться встановлення певних утиліти та залежностей.

Для запуску контейнера потрібно виконати в теці, де знаходиться Dockerfile команду ***docker build -t <container_name>*** для створення свого образу, та команду ***docker run -d -p <host_port>:<container_port>***. Щоб запустився контейнер в фоні використовується параметр ***-d*** а для прокидання портів параметр ***-p*** з портами хоста та контейнера. Також важливими командами є зупинка контейнера ***docker stop <container_name_or_id>***. Для отримання інформації про всі контейнери можна використати команду ***docker ps -a***, для списку образів – ***docker images***.

Щоб не засмічувати систему потрібно видаляти попередні образи та контейнери. Команди ***docker rmi <image_name>*** ***docker rm <container_name>*** відповідно.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Таким чином, у даному пункті було розглянуто основні кроки які потрібно виконати для налаштування. Це, зокрема створення Vagrantfile та скриптів для налаштування сервісів для кожної ВМ. Після підняття всіх ВМ опрацьовано налаштування Jenkins. Також було представлено роботу з Docker-контейнерами, застосування яких буде детальніше показане у розділі 3.

2.4. Вибір версій образів та програмного забезпечення

Для забезпечення стабільної роботи системи потрібно вибрати сумісне та коректне програмне забезпечення та стежити за регулярним оновленням. Для основного використаного у проєкті ПЗ (Vagrant, Jenkins, Docker) було встановлено останні стабільні версії.

Для Vagrant машин було вибрано один з найпоширеніших на момент розробки проєкта образів дистрибутива Linux ubuntu/jammy. Нижче наведено список доступних образів в Vagrant Cloud.

Box name	Latest Version	Downloads	Latest Release	Providers	Architectures
generic/ubuntu2204	4.3.12	203,857	344 days ago	qemu, libvirt, vmware_desktop, parallels, hyperv, virtualbox	amd64, unknown
ubuntu/jammy64	20241002.0.0	193,489	83 days ago	virtualbox	unknown
alvistack/devel-22.04	20241223.1.1	17,752	2 days ago	libvirt, virtualbox	amd64
alvistack/ubuntu-22.04	20241223.1.1	13,808	2 days ago	libvirt, virtualbox	amd64
generic-x64/ubuntu2204	4.3.12	2,808	344 days ago	parallels, virtualbox, qemu, libvirt, vmware_desktop, hyperv	amd64
luminositylabs/ci/ubuntu-22.04	20241125.01	1,725	29 days ago	virtualbox, parallels	amd64, unknown
fasmat/ubuntu2204-desktop	22.0509.1	1,308	137 weeks ago	vmware_desktop, virtualbox	unknown

Рис. 2.2. Доступні образи Ubuntu для Vagrant.

На момент виконання роботи актуальна версія Vagrant 2.4.3. Також варто виділити потрібну кількість ресурсів під кожну віртуальну машину в залежності від потреби. Оскільки даний проєкт буде створюватися на локальному комп'ютері, потрібно врахувати ресурси самого хоста. Через ці обмеження було виділено по 1 Гб оперативної пам'яті та 1 ядру центрального процесора. Цього вистачить для стабільної роботи ВМ. Ці значення можна змінити в подальшому, за потреби.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

Для роботи Jenkins для master та slave нод було обрано версію Java (JDK) 17. На момент виконання дана версія для Jenkins підходить, проте у перспективі доцільно буде оновити версію JDK, згідно з рекомендаціями. А сам Jenkins має версію 2.504.1.

Важливо відмітити, що для роботи Jenkins потрібно використовувати різні плагіни наприклад: Git, Publish Over SSH чи плагін Docker або інші плагіни які розширюють інструментарій.

Так як додаток написаний на PHP на базі фреймворку Laravel 11 [10], який підтримує версію PHP не менше 8.2. Виходячи з цього було обрано для створення Docker образу вже готовий *php:8.2-apache*. Наведений образ містить в собі сервер Apache який за допомогою модуля *mod_php* має можливість обробляти php-запити.

Отже, у наведеному пункті було підібрано та розглянуто версії образів та програмного забезпечення для стабільної роботи проєкту. Підібрано найкращі параметри для віртуальних машин, зважаючи на потреби та можливості виділення ресурсів.

2.5. Інші технології та інструменти

SSH ключі. *SSH keys* — це метод автентифікації для безпечного підключення до віддалених серверів через протокол *SSH (Secure Shell)*. Вони використовуються замість паролів (або разом із паролями) для забезпечення захищеного доступу до серверів, особливо під час роботи з Git, хостингами типу GitLab, або при вході на сервери Linux.

SSH працює наступним чином.

1. Створюється пара ключів.
 - Приватний ключ (private key) – зберігається на комп'ютері/сервері. Його ніколи не можна передавати або ділитися.
 - Публічний ключ (public key) – можна надсилати будь-кому. Його додають, наприклад, на сервер або в GitHub-акаунт.
2. Під час підключення відбуваються наступні дії:

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

- Сервер перевіряє, чи публічний ключ відповідає приватному, який зберігається на клієнті.
- Якщо так – дозволяє доступ без введення пароля.

Наведена технологія використовується для підключення slave-серверів Jenkins, а також для отримання репозиторія

Веб-додаток. Для перевірки роботи та повного відображення всього проєкту створено звичайний REST-full API (Application Programming Interface). Для цього використано PHP, а саме фреймворк Laravel [10]. Розроблено схему для «Вітуторини».

API матиме ендпоінти для автентифікації, перегляду, додавання та видалення певних об'єктів. На рисунку 2.3 зображено схему моделі бази даних (в додатку І міститься збільшене зображення).

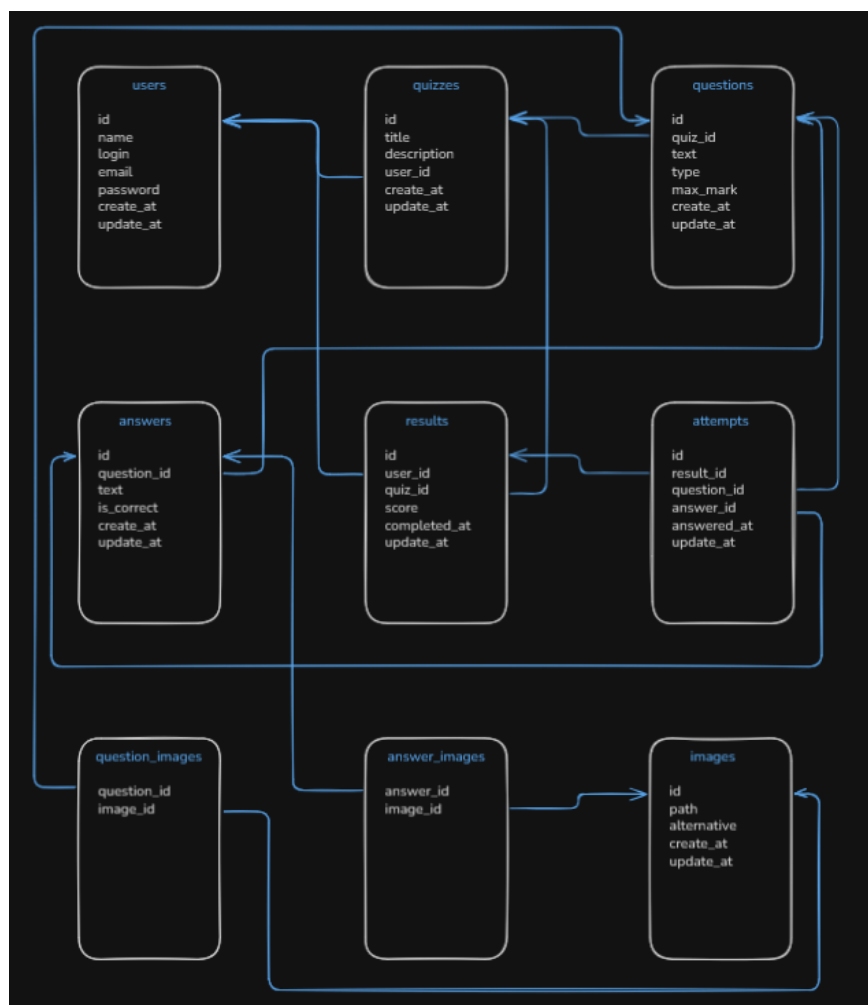


Рис. 2.3. Модель бази даних додатка.

Проксі сервер та балансування трафіка.

Також передбачається, що проєкт буде мати окрему ВМ для NGINX який буде слугувати як проксі-сервер для доступу до сервісів та додатків запущених в локальній мережі. Також даний сервер може виконувати роль балансувальника навантаження при використанні системи як готового продукту для користувачів.

Проксі-сервер в середовищі для тестування призначений для надання доступу до Jenkins master та до інших Jenkins slave серверів.

Висновки до розділу 2.

В другому розділі виконувалось проектування та розробка системи автоматичного розгортання проєктів. Створено схему проєкта та таблицю IP-адресації. Також було описано складові які буде мати проєкт та їхній функціонал в системі. На додачу було розглянуто основні кроки для побудови системи, здійснено вибір версій образів та програмного забезпечення. Виконана робота стала основою для виконання 3 розділу.

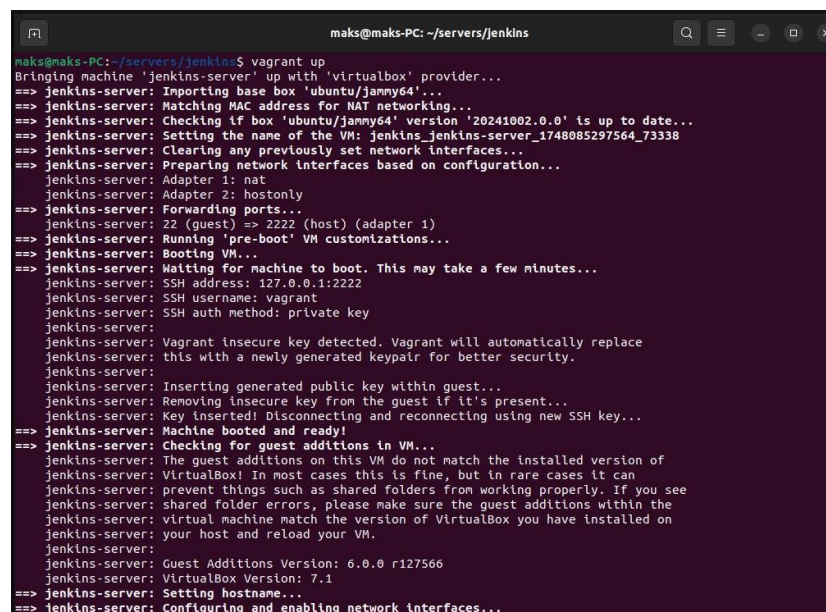
		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

РОЗДІЛ 3. НАЛАШТУВАННЯ ТА ДОСЛІДЖЕННЯ РОБОТИ СИСТЕМИ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ПРОЄКТІВ

3.1. Створення віртуальних машин та налаштування інструментів

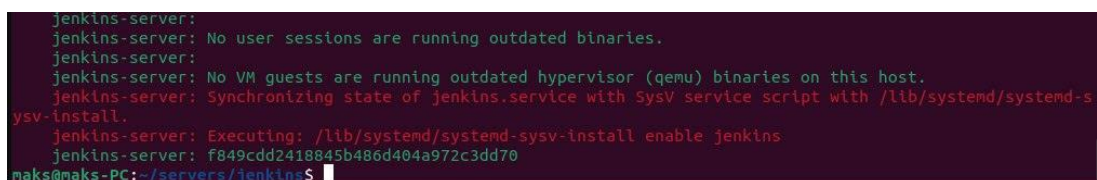
Перш за все було створено структуру каталогів проєкта. Після створення потрібних каталогів, відбулось створення Vagrantfile для кожної віртуальної машини проєкта. На додачу до цього застосовані bash скрипти для ініціалізації VM.

Так як було прийнято рішення використовувати окремі Vagrantfile для кожної VM, то для кожної VM створено окрему директорію з файлом конфігурації vagrant, в якій також за необхідності можуть міститися інші допоміжні файли та каталоги для роботи VM. Також створено директорію, яка буде містити скрипти для автоматичної ініціалізації серверів. Створення та ініціалізація VM зображена на рис. 3.1 та 3.2. Для зручності налаштування конфігурації створено файл *config.yml* з основними параметрами для запуску кожної VM.



```
maks@maks-PC: ~/servers/jenkins
maks@maks-PC:~/servers/jenkins$ vagrant up
Bringing machine 'jenkins-server' up with 'virtualbox' provider...
==> jenkins-server: Importing base box 'ubuntu/jammy64'...
==> jenkins-server: Matching MAC address for NAT networking...
==> jenkins-server: Checking if box 'ubuntu/jammy64' version '20241002.0.0' is up to date...
==> jenkins-server: Setting the name of the VM: jenkins_jenkins-server_1748085297564_73338
==> jenkins-server: Clearing any previously set network interfaces...
==> jenkins-server: Preparing network interfaces based on configuration...
jenkins-server: Adapter 1: nat
jenkins-server: Adapter 2: hostonly
==> jenkins-server: Forwarding ports...
jenkins-server: 22 (guest) => 2222 (host) (adapter 1)
==> jenkins-server: Running 'pre-boot' VM customizations...
==> jenkins-server: Booting VM...
==> jenkins-server: Waiting for machine to boot. This may take a few minutes...
jenkins-server: SSH address: 127.0.0.1:2222
jenkins-server: SSH username: vagrant
jenkins-server: SSH auth method: private key
jenkins-server:
jenkins-server: Vagrant insecure key detected. Vagrant will automatically replace
jenkins-server: this with a newly generated keypair for better security.
jenkins-server:
jenkins-server: Inserting generated public key within guest...
jenkins-server: Removing insecure key from the guest if it's present...
jenkins-server: Key inserted! Disconnecting and reconnecting using new SSH key...
==> jenkins-server: Machine booted and ready!
==> jenkins-server: Checking for guest additions in VM...
jenkins-server: The guest additions on this VM do not match the installed version of
jenkins-server: VirtualBox! In most cases this is fine, but in rare cases it can
jenkins-server: prevent things such as shared folders from working properly. If you see
jenkins-server: shared folder errors, please make sure the guest additions within the
jenkins-server: virtual machine match the version of VirtualBox you have installed on
jenkins-server: your host and reload your VM.
jenkins-server:
jenkins-server: Guest Additions Version: 6.0.0 r127566
jenkins-server: VirtualBox Version: 7.1
==> jenkins-server: Setting hostname...
==> jenkins-server: Configuring and enabling network interfaces...
```

Рис. 3.1. Запуск віртуальної машини для Jenkins.



```
jenkins-server:
jenkins-server: No user sessions are running outdated binaries.
jenkins-server:
jenkins-server: No VM guests are running outdated hypervisor (qemu) binaries on this host.
jenkins-server: Synchronizing state of jenkins.service with SysV service script with /lib/systemd/systemd-s
ysv-install.
jenkins-server: Executing: /lib/systemd/systemd-sysv-install enable jenkins
jenkins-server: f849cdd2418845b486d404a972c3dd70
maks@maks-PC:~/servers/jenkins$
```

Рис. 3.2. Завершення ініціалізації сервера.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

Далі було виконано генерування ssh-ключів (рис. 3.3). Після чого було використано згенеровані публічні ключі для налаштування доступу до GitLab та slave-агентів Jenkins (продемонстровано на рис. 3.4 та рис. 3.5).

```
maks@maks-PC: $ ssh-keygen -b 2048 -t rsa -C "GitLab for Jenkins" -f '/home/maks
/cred/id_rsa'
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match. Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/maks/cred/id_rsa
Your public key has been saved in /home/maks/cred/id_rsa.pub
The key fingerprint is:
SHA256:MNjVSp/7EorYzkao4LcQMh868tu6Str32IHr7hPN+Y GitLab for Jenkins
The key's randormat image is:
+----[RSA 2048]-----+
|o.  ..
|.o.. o . .
|.. o +. o .
|.. o. o
|.. S .
|.. o. o
|.. +=0 * . o
|o=B0oB o . .
|.. +@B=E . .
|.. +----[SHA256]-----+
maks@maks-PC: $
```

Рис. 3.3. Генерування ssh-ключів для GitLab.

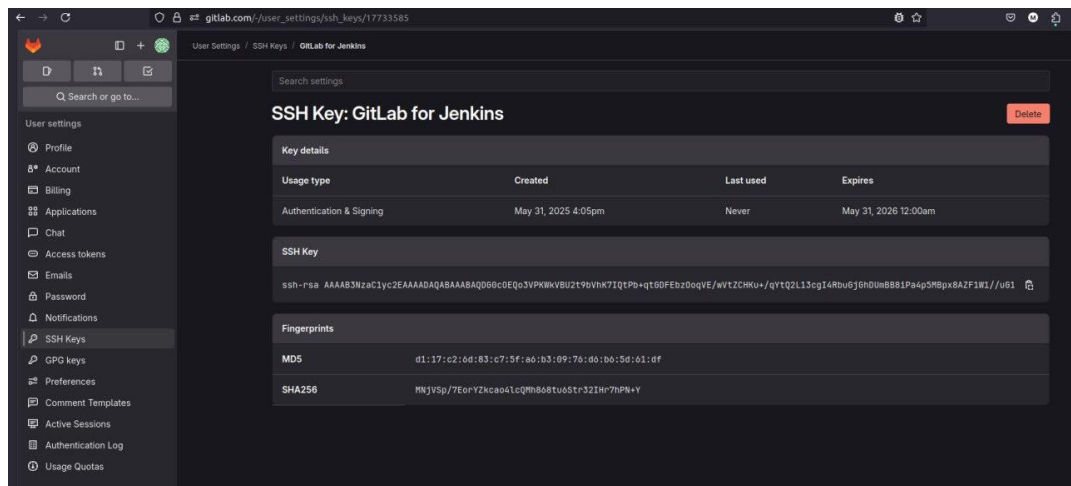


Рис. 3.4. Додання публічного ключа до GitLab.

```
vagrant@web-server: ~
Swap usage: 0%
Processes: 105
Users logged in: 0
IPv4 address for enp0s3: 10.0.2.15
IPv6 address for enp0s3: fd00::cd:f9ff:fe82:e7fc

Expanded Security Maintenance for Applications is not enabled.

81 updates can be applied immediately.
54 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '24.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

vagrant@web-server: $ echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAQC9JXGKsRn6Bsn86Fq2Y55nJZGxE+AmVB5UBhwY9WCx1Bm13bZPka4A8sfXh715vSuZsa
296g1pPxUhwde6TdJ8kgq10zwQ0t8dNrNM2gFXi133Lo68dSghHC5aQUM01/wQ91lhm62VZ1DJGv6B9LRHyLC6XtcogsaJ03XMPBea7yTEionfLD+XsSydqp6yq1w3D0uLGdN
REC4F/ttLnh2dgF57nPs5AvaNRlBr3YzvgqYXf6n31lgNVv4lpX5JQwhJh7LV5LHKH0K4sem5WguowJ0ZuzdH7yLMTJPImqHDFb7KfSLueGnEuK5TKZQ12yFA06MMrEk1L2RrB9
Yt9ZASAw02KZuV8thd08cZInt1Q25ahYMPVbGVLc91jwXGo6wbcl42w3Pw/5CtG6ZKNhro1fhVsZ1V1KDNpGf7193oE4qJ1HMDjgkenz03AVwkbhcy8XldZtYw/21W/xEZx
80A7TTC/QableFmX4B+1RL1PjpTeIzH5TWkGrUlhaxylTHv6UX0Hf2C7Pobronzh9K104M/GLSEdV/pUv4Nk75Hxtog0J6HOR6xZJ745HfGpVvqPQfJlyUFBKre1Q7eHUCB
0CZ021150137Zyna06GvQY9ZqppNF3CLRdQ1GcpnMECHs8tHa1FoJbyY9RCfmoIBnbacs9qW== jenkins-agent" >> .ssh/authorized_keys
vagrant@web-server: $
vagrant@web-server: $
vagrant@web-server: $
```

Рис. 3.5. Додання публічного ключа для Jenkins на сервері.

3.2. Налаштування Jenkins та Docker

Налаштування Jenkins.

Для доступу до Jenkins було введено адміністративний пароль (рис. 3.6) та створено користувача з повним доступом (рис. 3.7).

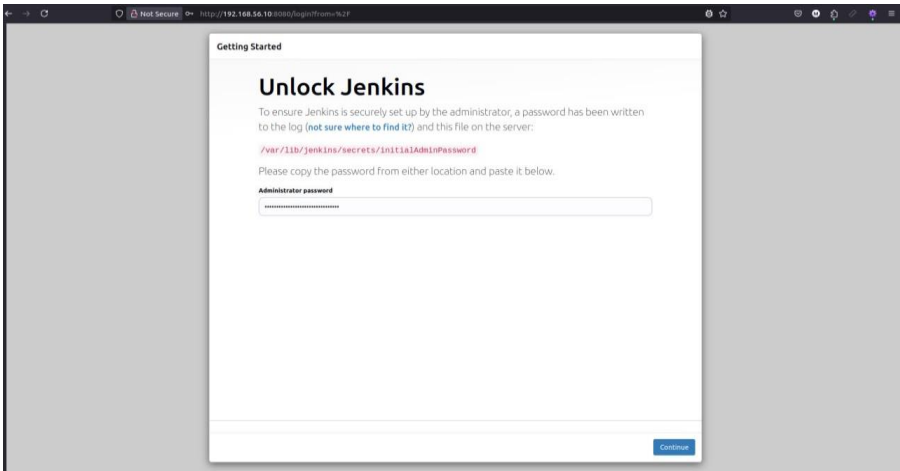


Рис. 3.6. Перший вхід до Jenkins.

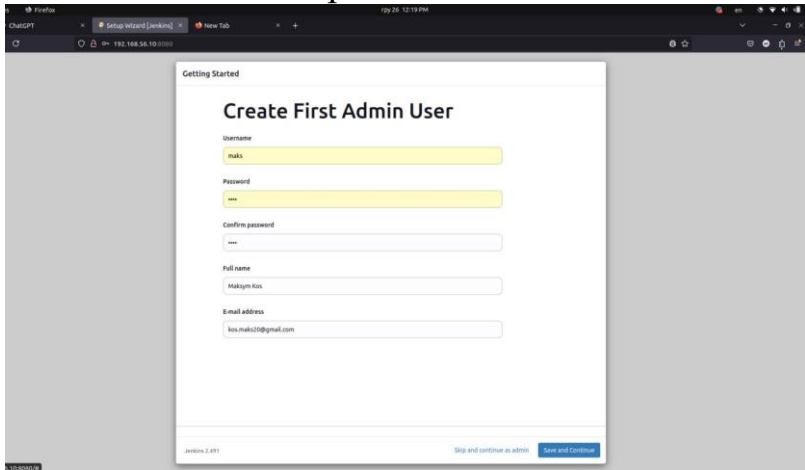


Рис. 3.7. Створення користувача-адміністратора.

Далі було встановлено необхідні плагіни для роботи з пайплайнами, docker та для візуалізації процесів виконання Jenkins-завдань (рис. 3.8 та рис. 3.9).

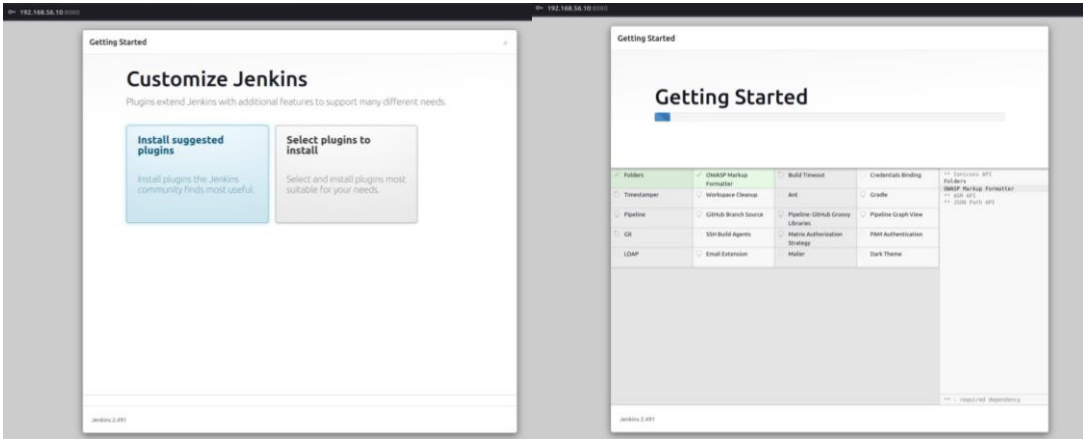


Рис. 3.8. Встановлення плагінів.

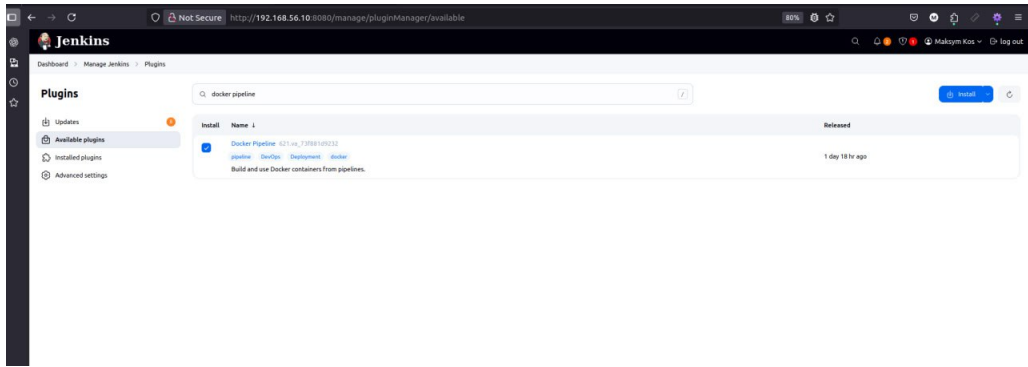


Рис. 3.9. Довстановлення плагіна Docker pipeline.

Далі було додано облікові дані для доступу до GitLab репозиторія. Та інші важливі дані для роботи проєкта (зображено на рис. 3.10 - 3.13).

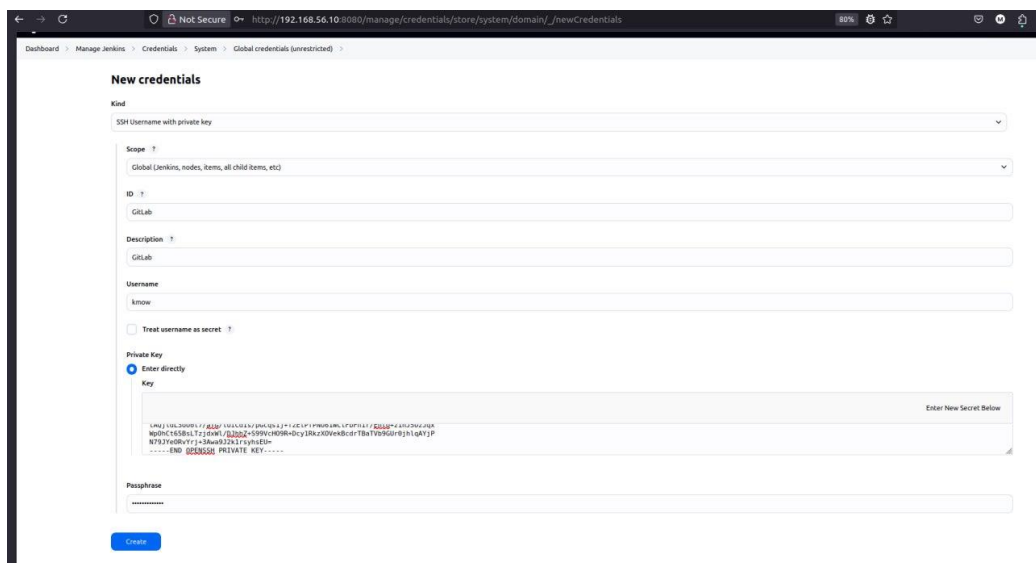


Рис. 3.10. Налаштування даних для підключення до сервера.

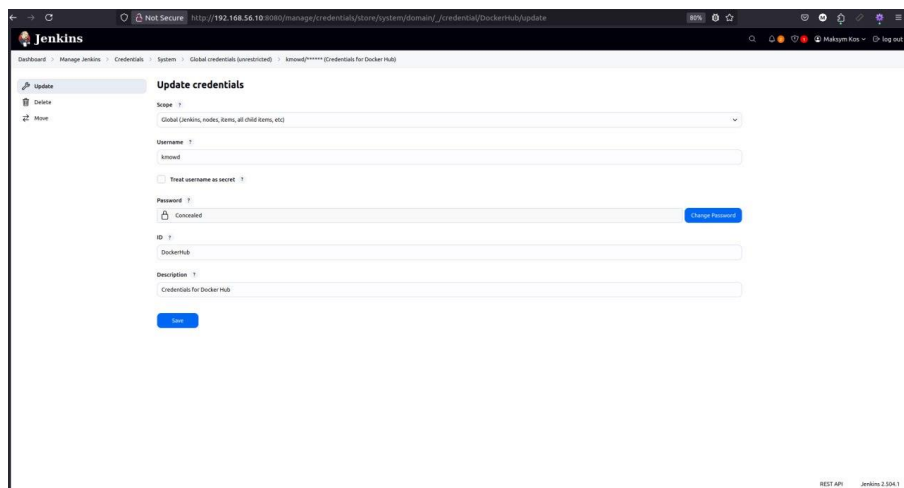


Рис. 3.11 Додання даних автентифікації для Docker Hub.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

Рис. 3.12. Додання до Jenkins відомостей для підключення до бази даних.

T	P	Store	Domain	ID	Name
		System	(global)	web-server1	vagrant (web-server1)
		System	(global)	jenkins-agent	vagrant (jenkins-agent)
		System	(global)	GitLab	know (GitLab)
		System	(global)	DockerHub	knowd/***** (Credentials for Docker Hub)
		System	(global)	db-host	Hostname (IP) for deploy application
		System	(global)	db-database	Database name for application
		System	(global)	db-username	User for db connection
		System	(global)	db-password	Password for connection to db
		System	(global)	db-connection	Type database for connection
		System	(global)	db-port	Port for connection to DB

P	Store	Domains
	System	(global)

Рис. 3.13. Облікові дані які застосовуються.

На рисунку 3.14 можна побачити, що не вдалось підключити Jenkins slave-сервер.

Рис. 3.14. Спроба підключення першого slave-сервера.

Після невдалої спроби підключити slave-агента (рис. 3.15.) було проведено ручне додавання публічного ключа. Після чого виконано успішне підключення slave-агента (рис. 3.16).

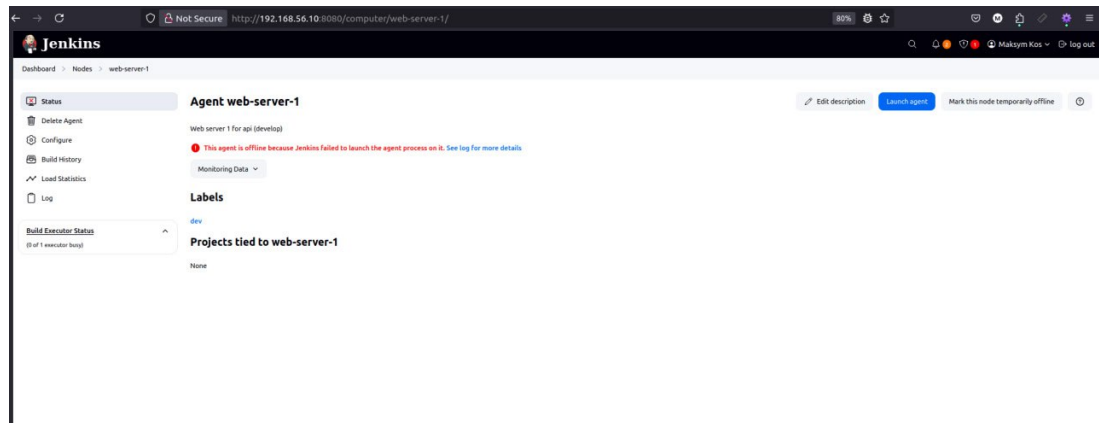


Рис. 3.15. Відображення повідомлення про те, що агент не доступний.

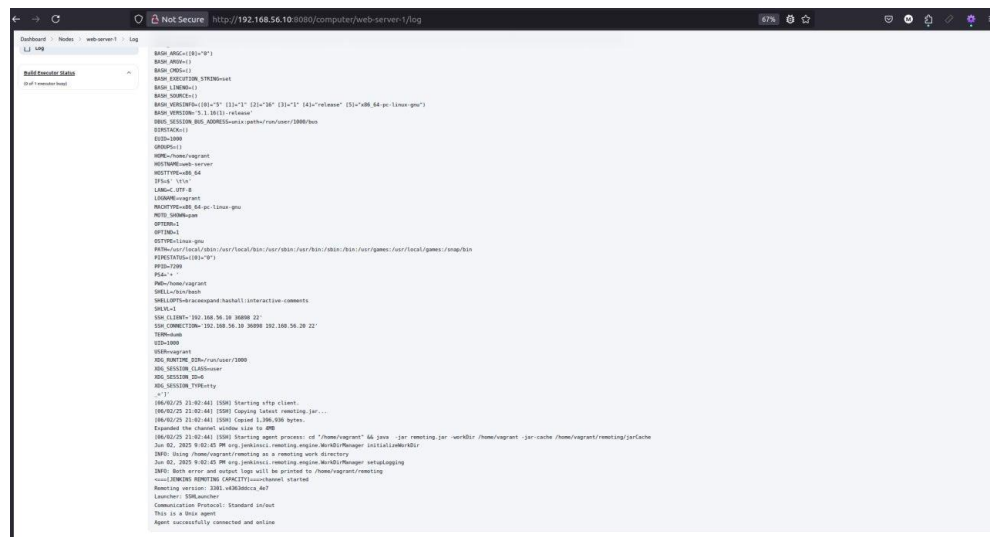


Рис. 3.16. Успішне підключення агента.

Таким же чином як і *web-server-1*, було створено *web-server-2* та налаштовано на master ноді, що зображено на рис. 3.17-3.18.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

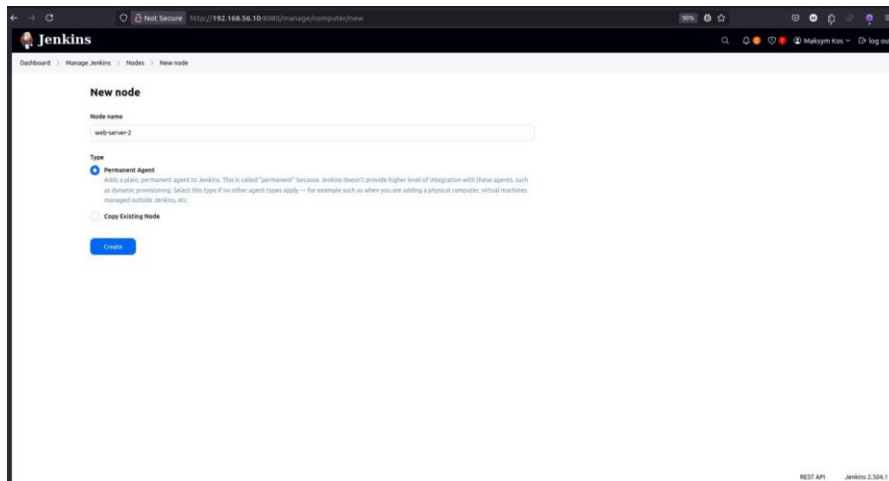


Рис. 3.17. Конфігурація web-server-2.

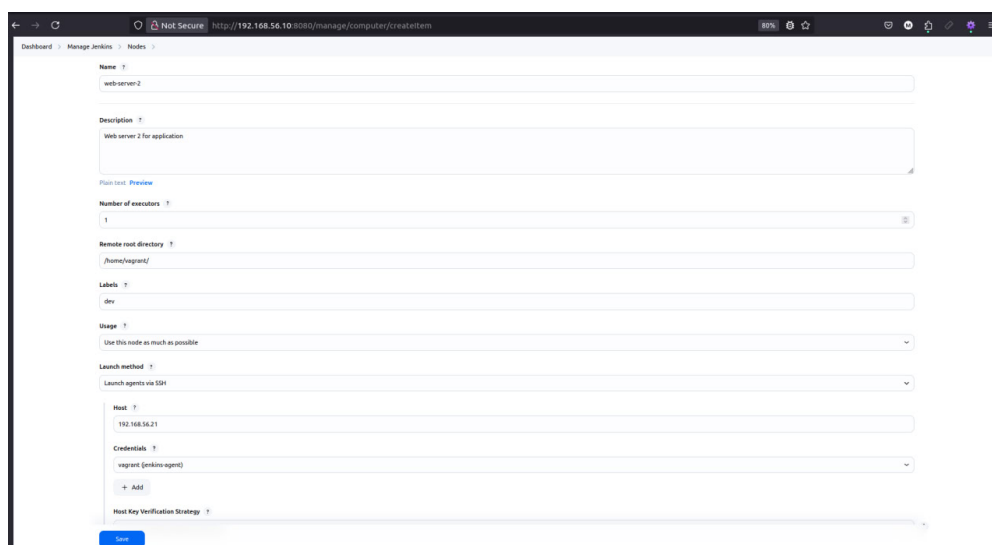


Рис. 3.18. Конфігурація Jenkins ноди.

Налаштування Jenkins завдань.

Наступним кроком проведено налаштування завдання (рис. 3.19 - 3.20), яке отримує пайплайн з репозиторія та виконує всі необхідні дії для запуску додатка на веб-сервері.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

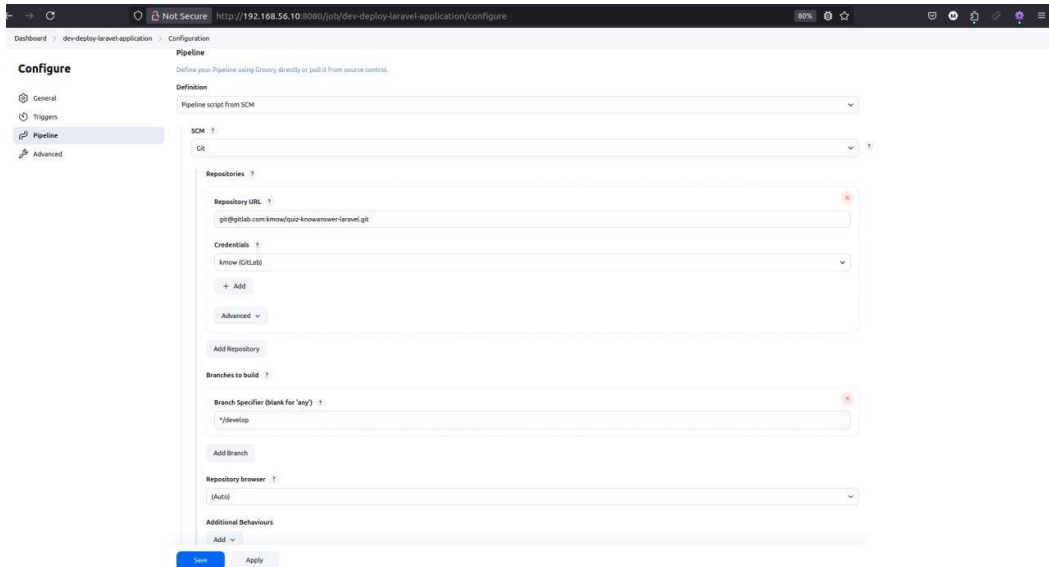


Рис. 3.19. Створення Jenkins завдання.

На додачу було налаштовано інше Jenkins завдання (рис. 3.21) для деплою іншої гілки додатка.

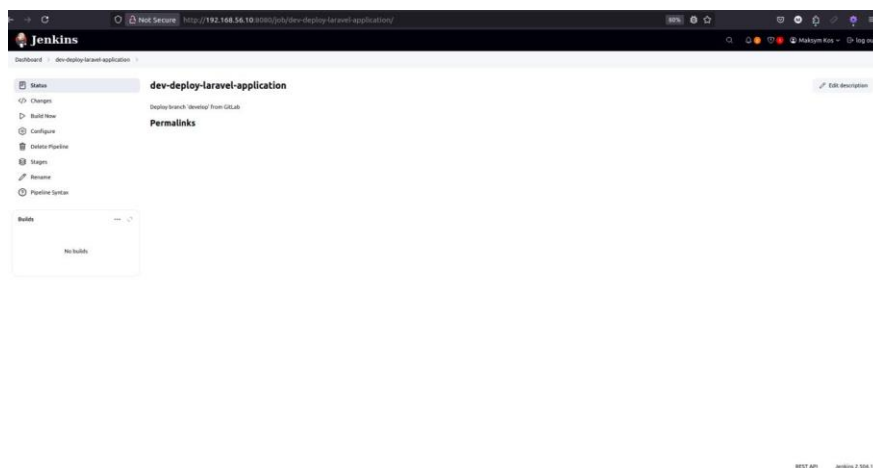


Рис. 3.20. Перше створене завдання.

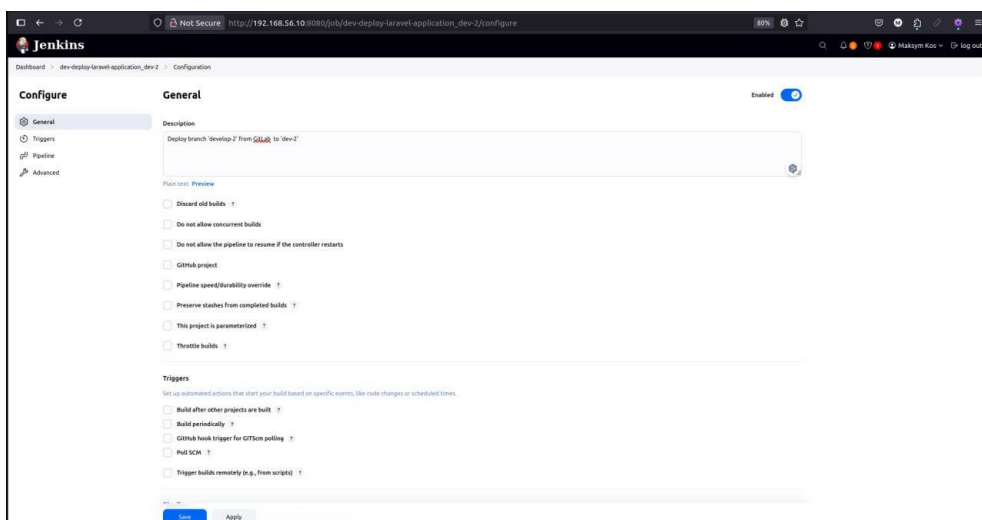


Рис. 3.21. Створення завдання для розгортання гілки *develop-2*.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

На рисунку 3.22 наведено налаштування тригерів для запуску Jenkins завдань.

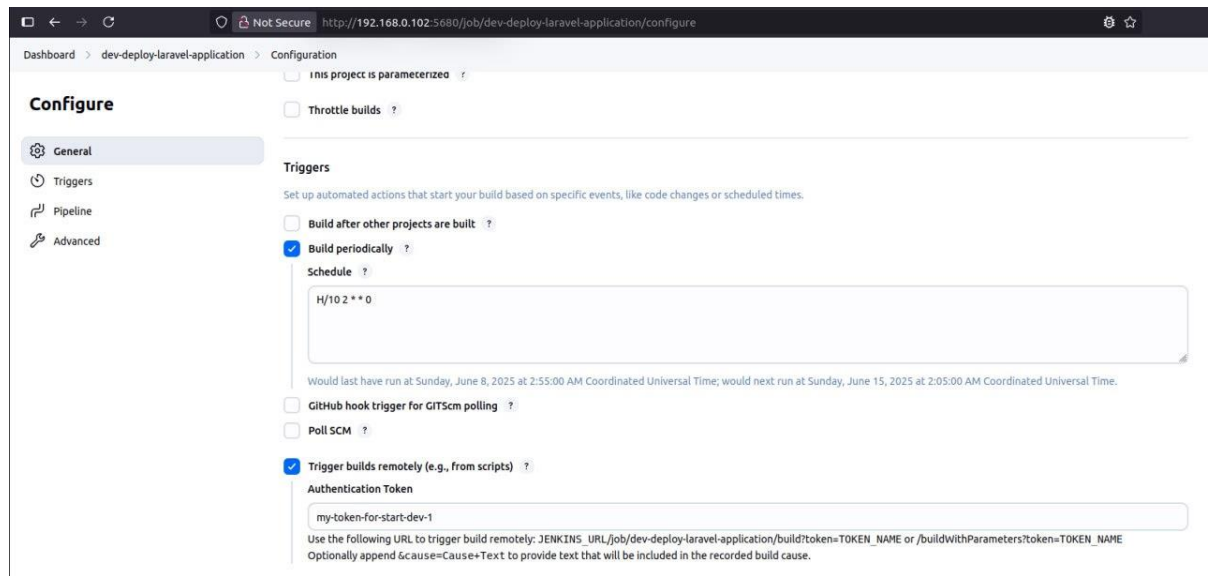


Рис. 3.22. Налаштування часу коли завдання буде виконуватися та віддаленого тригера.

Також було створено Jenkinsfile який буде запускатися створеними завданнями. Повний вміст пайплайна міститься в додатку А.

Створення Dockerfile.

Далі було створено Dockerfile, щоб можна було запускати контейнери. Лістинг контейнера міститься в додатку Б.

Після запуску контейнера було запущено команду, яка виконує міграції бази даних (рис. 3.23).

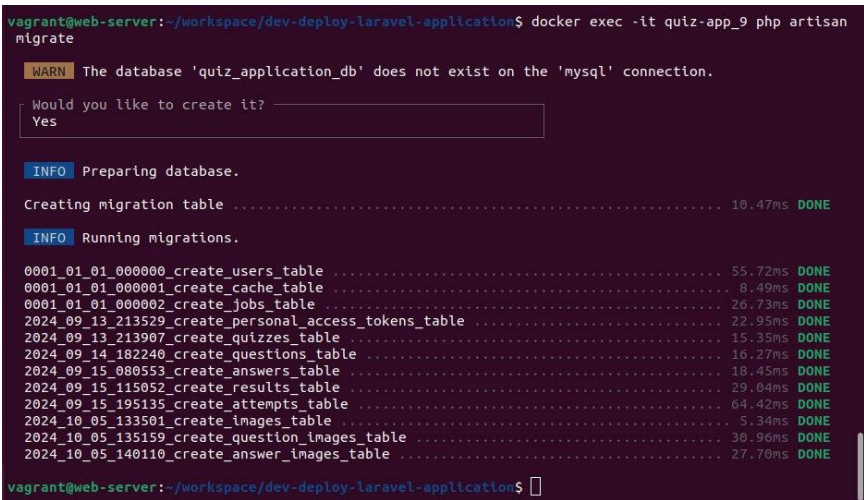


Рис. 3.23. Виконання міграцій.

Вище згаданий крок не є обов'язковим, якщо база даних вже має певні записи.

3.3. Тестування роботи інфраструктури

Після проведених налаштувань настав час тестувати проєкт. Наведений рисунок 3.24 показує, що під час запуску завдання виникли проблеми, тому було внесено деякі зміни до пайплайна.

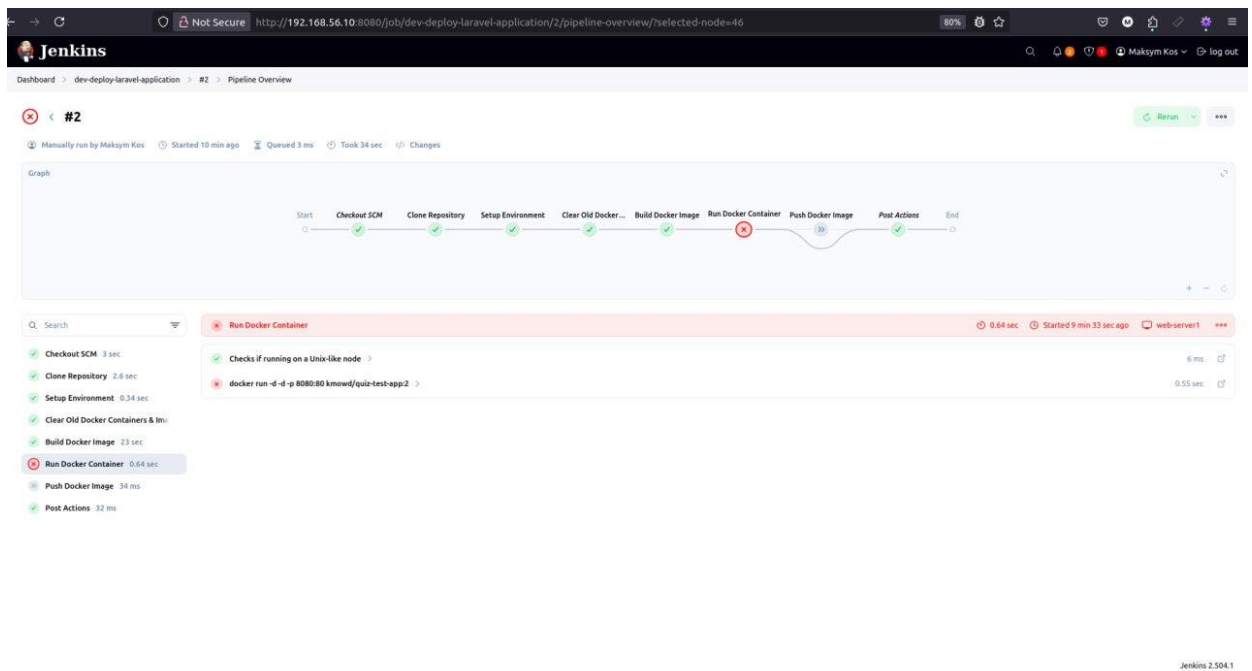


Рис. 3.24. Запуск завдання.

Після запуску завдання було отримано помилку (рис. 3.24) на етапі запуску контейнера. Тому було проведено певні зміни.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

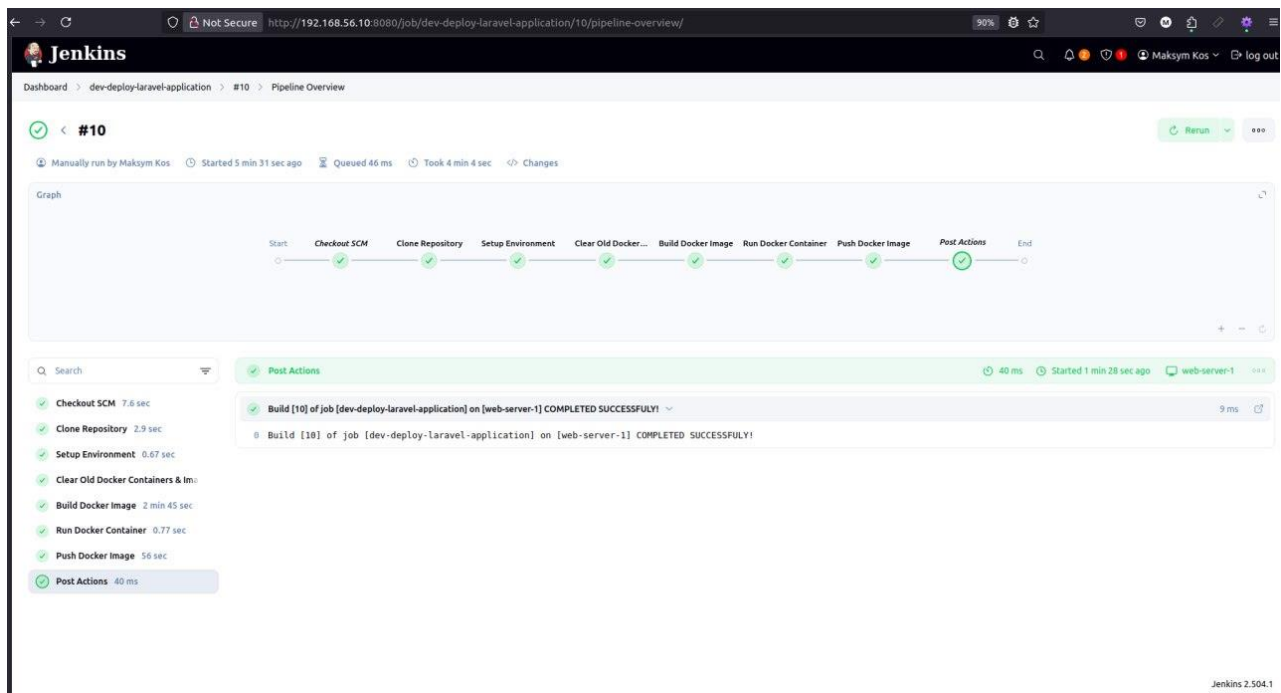


Рис. 3.25. Інша спроба запуску.

Далі все відпрацювало добре (рис. 3.25) і можна переглянути результати виконання, а саме на розгорнутий додаток завдяки Docker (рис. 3.26) та вивантажений створений образ (рис. 3.27). Також варто переглянути образи та контейнери, які є на веб-сервері (рис. 3.28).

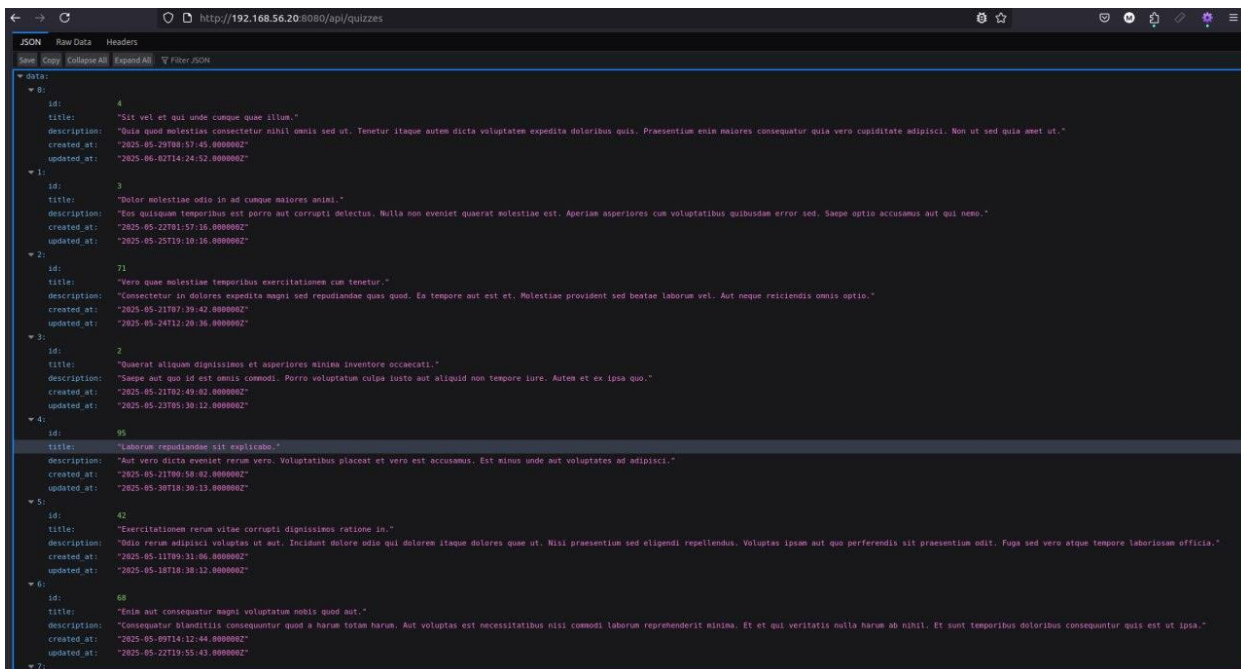


Рис. 3.26 Перевірка доступу до API після виконання створення контейнера.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

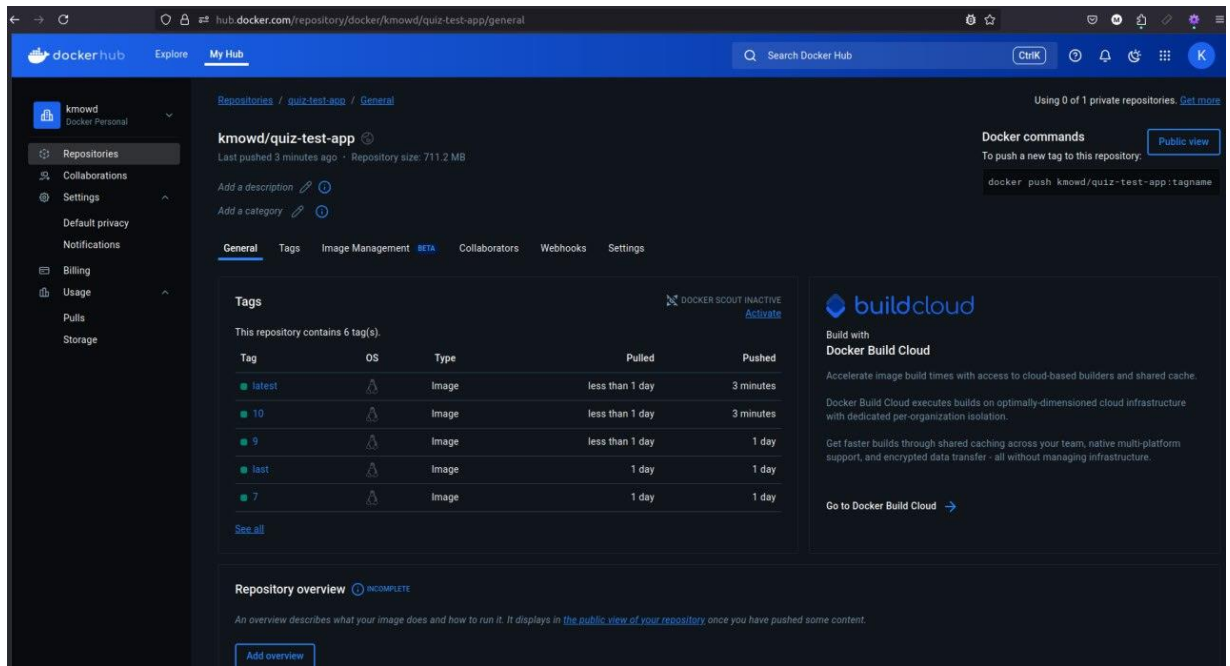


Рис. 3.27. Перевірка наявності нових образів в Docker Hub.

```
vagrant@web-server:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
c92fe9499f79   kmowd/quiz-test-app:10             "docker-php-entrypoi..." 8 minutes ago  Up 8 minutes  0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   quiz-app
vagrant@web-server:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
c92fe9499f79   kmowd/quiz-test-app:10             "docker-php-entrypoi..." 8 minutes ago  Up 8 minutes  0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   quiz-app
vagrant@web-server:~$ docker images
REPOSITORY      TAG        IMAGE ID      CREATED        SIZE
kmowd/quiz-test-app  10        94495786d35a  8 minutes ago  1.13GB
kmowd/quiz-test-app  latest    94495786d35a  8 minutes ago  1.13GB
vagrant@web-server:~$
```

Рис.3.28. Контейнер на web-сервері.

У початковій реалізації 2 агенти мають мітки *dev*, тобто при білді одного завдання будуть запусненні docker контейнери на кожному з них. Для перевірки роботи потрібно звернути увагу на ір адреси серверів 192.168.56.20 та 192.168.56.21. На рисунку 3.29 можна побачити, що інший веб-сервер також працює коректно.

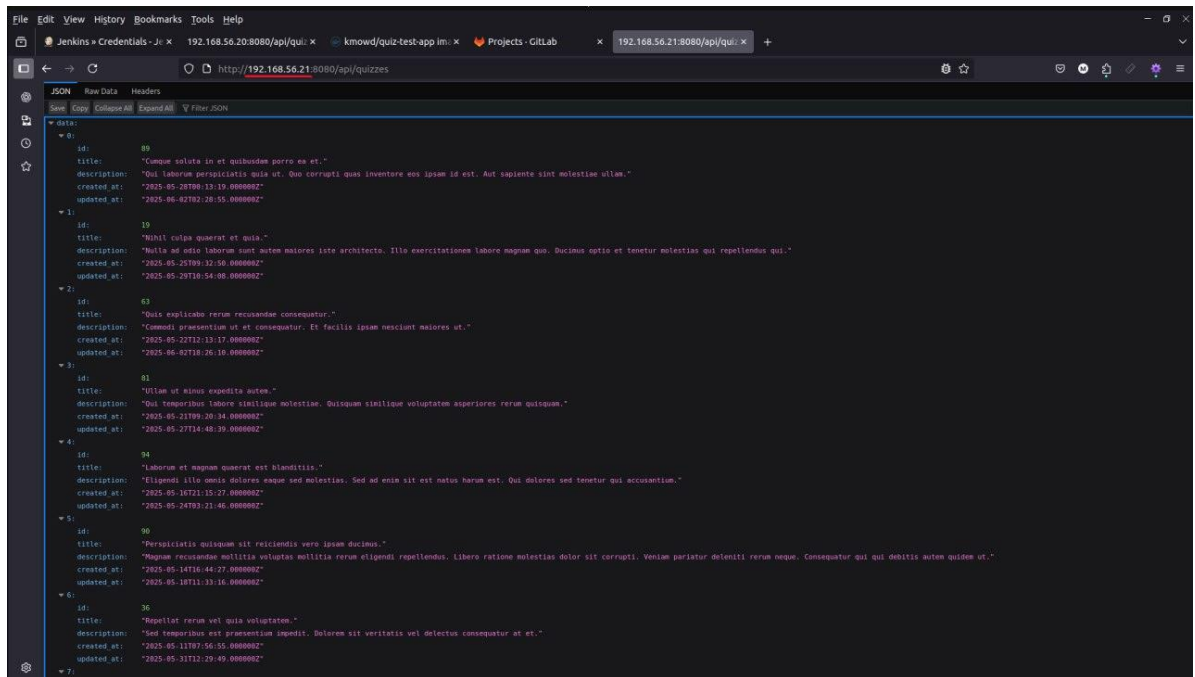


Рис. 3.29. Доступ до web-server-2.

Далі було розглянуто налаштування проксі сервера (рис. 3.30), який виконує ще роль балансувальника навантаження. Також було протестовано роботу завдяки створенню скрипта, який звертається до проксі-сервера, який в свою чергу дані запити розподіляє між нашими веб-серверами (рис. 3.31 – 3.34).

```
! config.yml  $ start-container.sh  Vagrantfile web  Vagrantfile proxy  proxy.conf  Vagrantfile jenkins

proxy > nginx-conf > proxy.conf
1 upstream api_servers {
2     server 192.168.56.20:8080;
3     server 192.168.56.21:8080;
4 }
5
6
7 # web application API
8 server {
9     listen 80;
10    server_name 192.168.56.100;
11
12    location / {
13        proxy_pass http://api_servers;
14        proxy_http_version 1.1;
15        proxy_set_header Upgrade $http_upgrade;
16        proxy_set_header Connection 'upgrade';
17        proxy_set_header Host $host;
18        proxy_cache_bypass $http_upgrade;
19
20        add_header X-Content-Type-Options nosniff;
21        add_header X-Frame-Options "SAMEORIGIN";
22        add_header X-XSS-Protection "1; mode=block";
23    }
24 }
25
26 # jenkins
27 server {
28     listen 8080;
29     server_name 192.168.56.100;
30     location / {
31         proxy_pass http://192.168.56.10:8080;
32     }
33 }
```

Рис. 3.30. Налаштування nginx.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

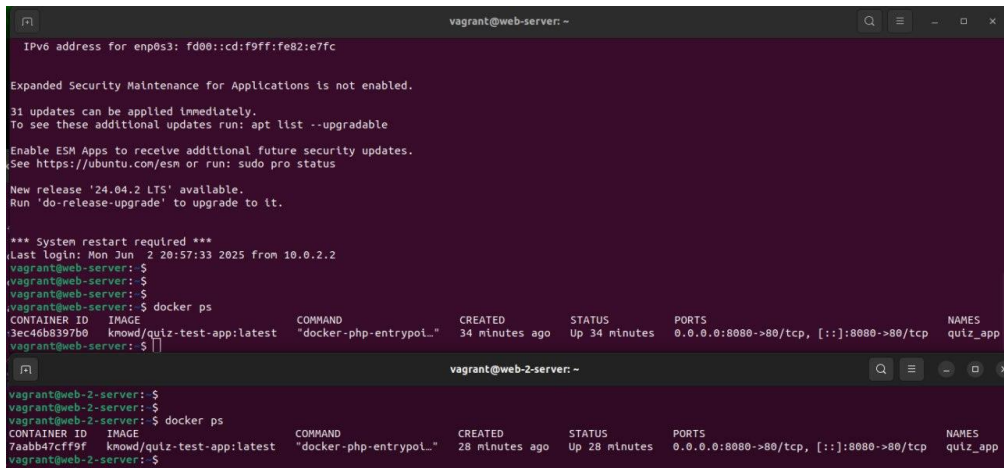


Рис. 3.31. Перевірка успішності запуску контейнерів на серверах.

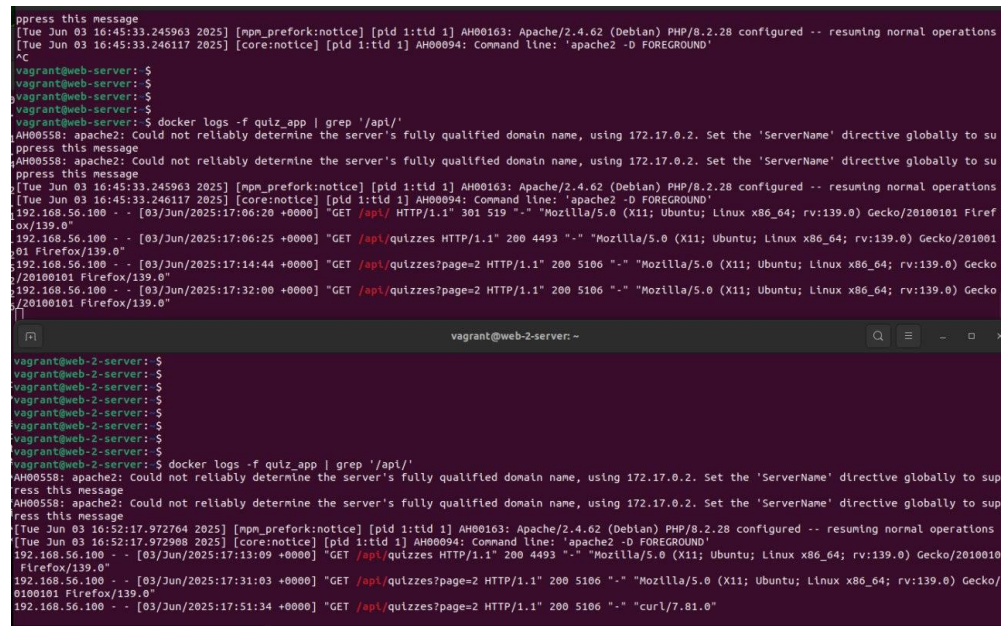


Рис. 3.32. Перегляд логів контейнерів пов’язаних з доступом до АРІ.

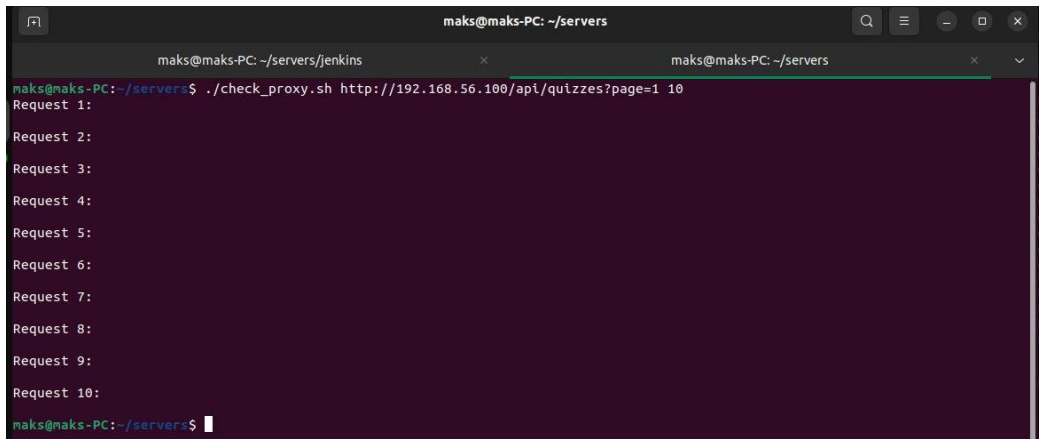


Рис. 3.33. Запуск скрипта, який робить 10 запитів до певного маршрута.

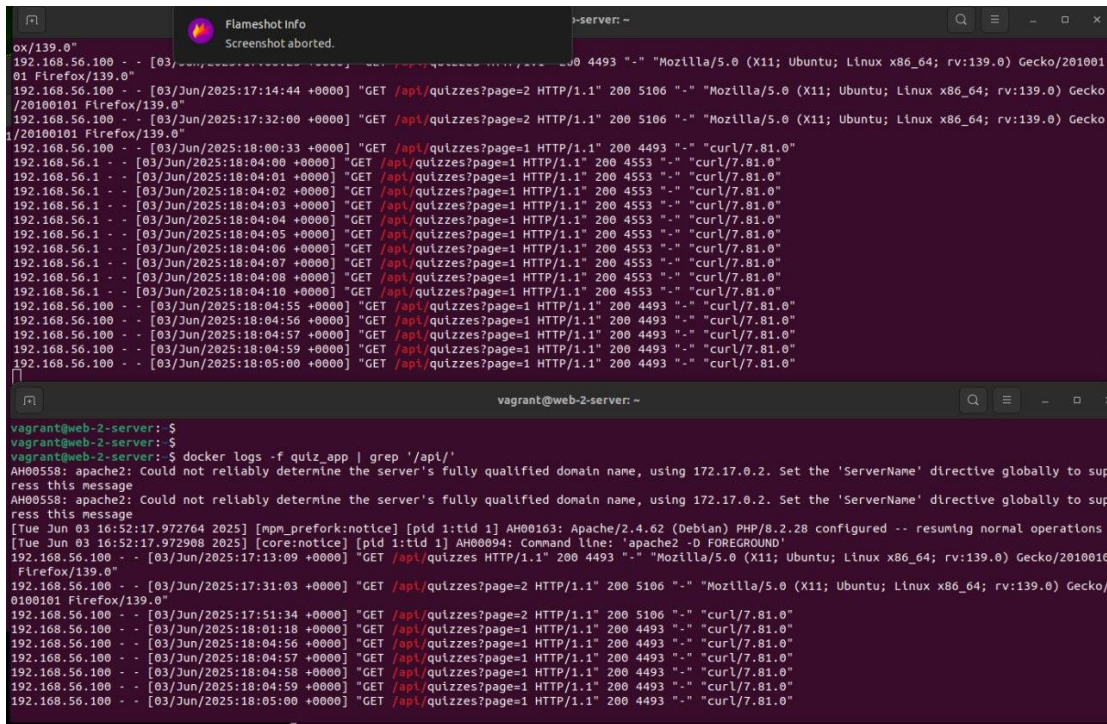


Рис. 3.34. Логи після виконання запитів.

Далі було змінено на одному slave-агенті мітку *dev* на *dev-2* та створено нову гілку, в якій було змінено Jenkinsfile і виконано невеликі зміни в самому API. Було додано новий ендпоінт та завантажено до GitLab (рис. 3.35 – 3.36).

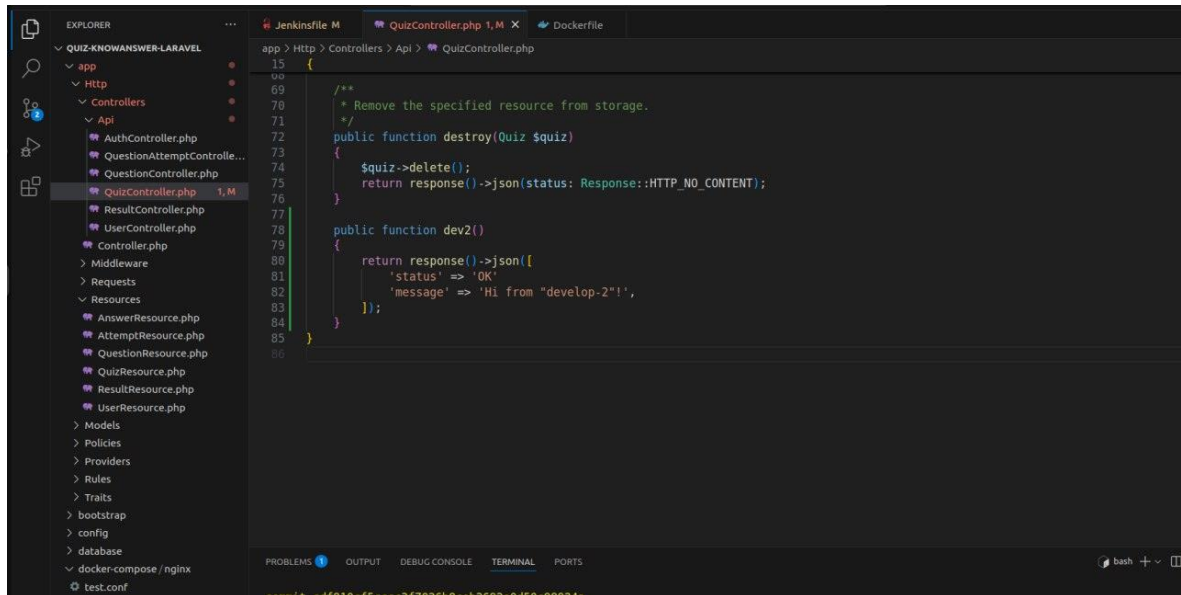


Рис. 3.35. Змінення API.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• maks@maks-PC:~/quiz-knowanswer-laravel$ git checkout develop-2
Switched to branch 'develop-2'
• maks@maks-PC:~/quiz-knowanswer-laravel$ git status
On branch develop-2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   Jenkinsfile
    modified:   app/Http/Controllers/Api/QuizController.php
    modified:   routes/api.php

no changes added to commit (use "git add" and/or "git commit -a")
• maks@maks-PC:~/quiz-knowanswer-laravel$ git add .
• maks@maks-PC:~/quiz-knowanswer-laravel$ git commit -m "Added route dev2"
[develop-2 17d580b] Added route dev2
 3 files changed, 10 insertions(+), 2 deletions(-)
• maks@maks-PC:~/quiz-knowanswer-laravel$ git push origin develop-2
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 981 bytes | 490.00 KiB/s, done.
Total 10 (delta 6), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for develop-2, visit:
remote:   https://gitlab.com/know/quiz-knowanswer-laravel/-/merge_requests/new?merge_request%5Bsource_branch%5D=develop-2
remote:
To gitlab.com:know/quiz-knowanswer-laravel.git
 * [new branch]   develop-2 -> develop-2
• maks@maks-PC:~/quiz-knowanswer-laravel$

```

Рис. 3.36. Створення нової гілки на GitLab.

Також для розгортання цієї гілки було створено інше Jenkins-завдання яке отримує код з гілки *develop-2*. Виконання даного завдання продемонстровано на рисунку 3.37.

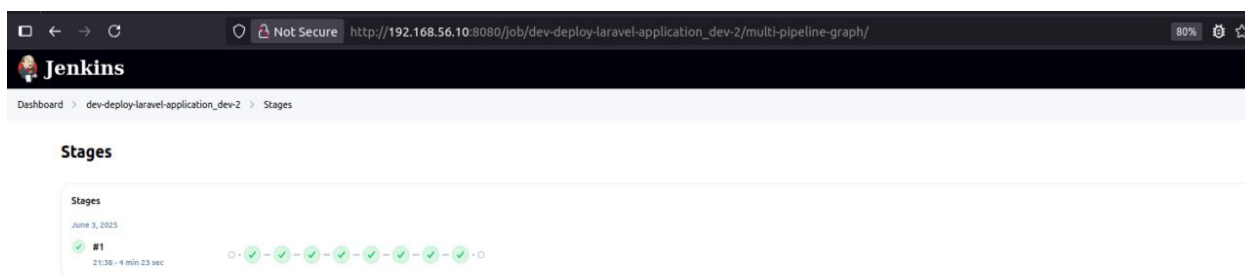


Рис. 3.37. Успішне виконання pipeline.

Після цього було перевірено, що на серверах містяться різні збірки завдань (рис. 3.38). Як видно на одному сервері маршрут */api/dev2* працює і повертає повідомлення, а на іншому – ні.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

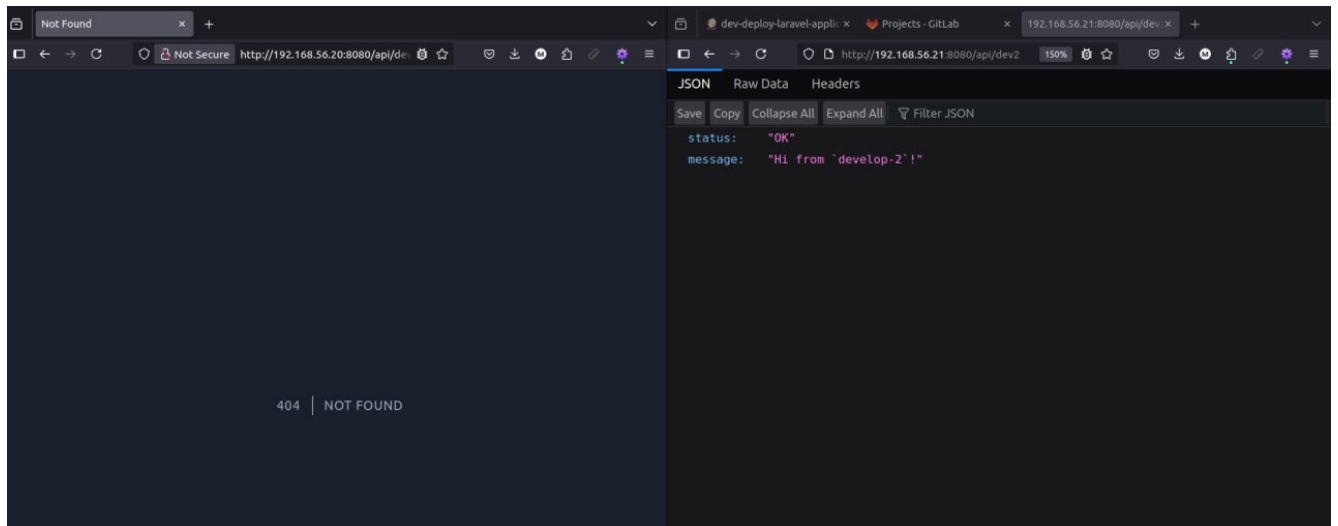


Рис. 3.38. Перевірка, що сервери містять різні контейнери.

До речі, тепер можна перевірити застосування скрипта для створення бекапів Jenkins (рис. 3.39 - 3.4). Для цього потрібно застосувати vagrant-провіжин скрипт для Jenkins-сервера.

```
maks@maks-PC:~/servers/jenkins$ vagrant provision --provision-with backup-jenkins
==> jenkins-server: Running provisioner: backup-jenkins (shell)...
jenkins-server: Running: script: backup-jenkins
jenkins-server: Backup saved to: /home/vagrant/jenkins_backups/jenkins_backup_2025-06-12-11-13.tar.gz
maks@maks-PC:~/servers/jenkins$
maks@maks-PC:~/servers/jenkins$
maks@maks-PC:~/servers/jenkins$
maks@maks-PC:~/servers/jenkins$ ls jenkins_backups/
jenkins_backup_2025-06-02-16-58.tar.gz jenkins_backup_2025-06-12-11-13.tar.gz
maks@maks-PC:~/servers/jenkins$
```

Рис. 3.39. Бекап Jenkins.

```
maks@maks-PC: ~/servers/jenkins
maks@maks-PC:~/servers/jenkins$ cd jenkins_backups/
maks@maks-PC:~/servers/jenkins/jenkins_backups$ cp jenkins_backup_2025-06-12-11-13.tar.gz ~/restore-jenkins/backup.ta
r.gz
cp: cannot create regular file '/home/maks/restore-jenkins/backup.tar.gz': No such file or directory
maks@maks-PC:~/servers/jenkins/jenkins_backups$ mkdir ~/restore-jenkins
maks@maks-PC:~/servers/jenkins/jenkins_backups$ cp jenkins_backup_2025-06-12-11-13.tar.gz ~/restore-jenkins/backup.ta
r.gz
maks@maks-PC:~/servers/jenkins/jenkins_backups$ cd ~/restore-jenkins/
maks@maks-PC:~/restore-jenkins$ ls
backup.tar.gz
maks@maks-PC:~/restore-jenkins$ sudo tar -xzf backup.tar.gz
Command 'suo' not found, did you mean:
  command 'sudo' from deb sudo (1.9.9-1ubuntu2.4)
  command 'sudo' from deb sudo-ldap (1.9.9-1ubuntu2.4)
  command 'su1' from deb hxttools (20211204-1)
  command 'sur' from deb subtle (0.11.3224-xi-2.2build4)
  command 'sup' from deb sup (20100519-3)
  command 'su' from deb util-linux (2.37.2-4ubuntu3.4)
  command 'sum' from deb coreutils (8.32-4.1ubuntu1.2)
  command 'sumo' from deb sumo (1.12.0+dfsg1-1)
Try: sudo apt install <deb name>
maks@maks-PC:~/restore-jenkins$ sudo tar -xzf backup.tar.gz
[sudo] password for maks:
maks@maks-PC:~/restore-jenkins$ ls
backup.tar.gz jenkins
```

Рис. 3.40. Архів з налаштованим Jenkins.

Для перевірки роботи бекапа було виконана запуск Jenkins завдяки Docker, разом з вже налаштованим Jenkins. Рисунки 3.41 – 3.42 демонструють, що це (бекап) працює.

```
maks@maks-PC: ~/restore-jenkins$
maks@maks-PC: ~/restore-jenkins$ docker run -d --name jenkins-restored -p 8080:8080 -p 50000:50000 -v $HOME/restore-jenkins:/var/jenkins_home jenkins/jenkins:lts
Unable to find image 'jenkins/jenkins:lts' locally
lts: Pulling from jenkins/jenkins
3e6b9d1a9511: Pull complete
1c0b6121482a: Pull complete
cbd1283dc3b2: Pull complete
b9b1866598bd: Pull complete
88ce37b37f54: Pull complete
57daf441aeec: Pull complete
10e8fd64a503: Pull complete
d53560724658: Pull complete
c10eb6e1ab33: Pull complete
37306438988a: Pull complete
82349c49f528: Pull complete
d9f1facce911: Pull complete
Digest: sha256:2abbd073f06850c3c962eedaa10e7aa21c517db1e582ad8c89b83ecfb6cb7510
Status: Downloaded newer image for jenkins/jenkins:lts
e040cff821fbe617a806e46c1df5fb874b907fb7b72e7367e57f2e6ec76d2c5d
maks@maks-PC: ~/restore-jenkins$
```

Рис. 3.41. Перевірка відновлення Jenkins завдяки Docker.

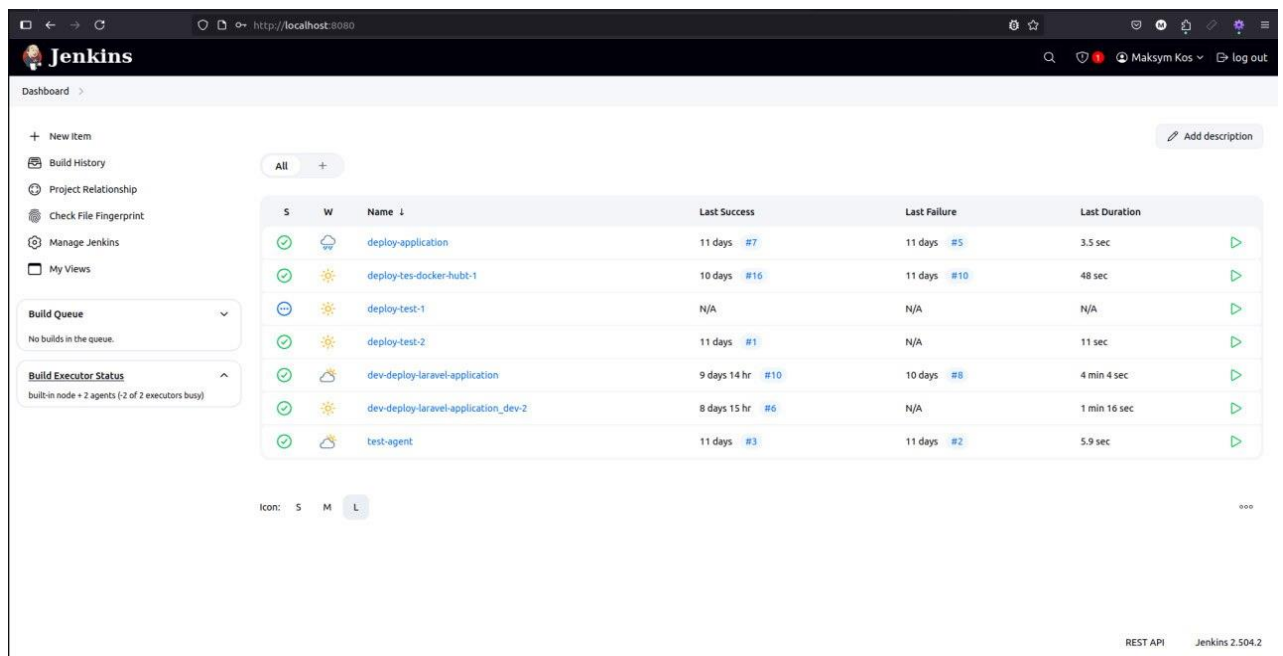


Рис. 3.42. Доступ до Jenkins контейнера.

Висновки до розділу 3.

У даному розділі було створено проєкт з використанням технологій Jenkins, Docker а також Vagrant. Спочатку було створено інфраструктуру серверів та налаштовано Jenkins та пайплайни. Потім було проведено перевірку роботи проєкту, та виправлення наявних помилок. Після цього було протестовано доступ до створеного API. Переглянуто інші функціональні можливості проєкта, які полегшують роботу.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було досліджено та використано такі технології: Jenkins, Docker, Vagrant та інші допоміжні.

Проект успішно продемонстрував ефективність інтеграції Jenkins та Docker для повної автоматизації процесів безперервної інтеграції та безперервного розгортання (CI/CD). Використання Vagrant та VirtualBox дозволило створити ізольоване, відтворюване та стандартизоване середовище для розробки та тестування, що імітує реальну інфраструктуру.

Проект містить віртуальні машини, на яких запущений певний сервіс. Для демонстрування роботи було реалізовано простий API завдяки фреймворку Laravel. Використано сервер Jenkins який налаштований як сервер, що отримує програмний код з GitLab, та запускає з цього коду Jenkins-пайплайн який здійснює збірку, тестування та розгортання docker-контейнера та вивантаження docker-образа в Docker Hub.

Крім того, було використано Nginx завдяки якому налаштовано проксі та балансувальник навантаження який розподіляє навантаження між серверами з API.

Основні висновки з реалізації проекту полягають у значному прискоренні циклу розробки та розгортання, оскільки автоматизація збірки, тестування та деплою через Jenkins-пайплайни суттєво скорочує час від написання коду до його доставки. Це підвищує надійність та відтворюваність розгортання, адже Docker-контейнери забезпечують послідовне середовище для застосунків, усуваючи проблеми залежностей та гарантуючи однакову поведінку коду на різних етапах CI/CD. Крім того, автоматизація максимально знижує вплив людського фактора, мінімізуючи кількість помилок, що можуть виникнути при ручному розгортанні, та оптимізує використання ресурсів, оскільки кожен застосунок запускається в легкому, ізольованому контейнері.

Таким чином, розроблена та реалізована система підтверджує, що інтеграція Jenkins та Docker є потужним інструментом для побудови сучасних, гнучких та надійних CI/CD конвеєрів, що є критично важливим для будь-якого сучасного програмного проекту.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

В майбутньому даний проект можна розвивати та вдосконалювати під потрібні процеси. Одним з поліпшень може стати створення всіх ВМ з одного Vagrantfile, а всі дані про налаштування будуть міститися в файлі конфігурації. Також можна внести зміни до налаштувань Jenkins, наприклад додати плагіни для комфортної роботи та відстеження роботи Jenkins-завдань чи інтегрувати системи для інформування користувачів.

В цілому виконання кваліфікаційної роботи насамперед дало змогу застосувати набуті під час навчання знання та застосувати їх на практиці під час розробки даного проєкта. На додачу до цього було отримано та освоєно нові технології та їх застосування на практиці. Набуті навички є ціними під час працевлаштування та подальшого розвитку.

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ластер Б. Jenkins 2. Up and Running. O`Reilly, 2018. 653 с.
2. BasuMallick C. What Is Jenkins? Working, Uses, Pipelines, and Features. <https://www.spiceworks.com>. URL: <https://www.spiceworks.com/tech/devops/articles/what-is-jenkins/> (date of access: 25.12.2024).
3. Docker for the Absolute Beginner - Hands On - DevOps, 22. <https://ua.udemy.com>. URL: <https://ua.udemy.com/course/learn-docker/learn/lecture/7838182?start=0#overview> (date of access: 27.09.2024).
4. GeeksforGeeks. Features of Docker - GeeksforGeeks. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/features-of-docker/> (date of access: 28.12.2024).
5. Hashicorp. What is Vagrant? | Vagrant | HashiCorp Developer. <https://developer.hashicorp.com>. URL: <https://developer.hashicorp.com/vagrant/tutorials/getting-started/getting-started-index> (date of access: 27.12.2024).
6. IBM. What Is Docker? | IBM. *IBM - United States*. URL: <https://www.ibm.com/think/topics/docker> (date of access: 28.12.2024).
7. Blum, R., & Bresnahan, C. *Linux Command Line and Shell Scripting Bible (4th ed.)*. Wiley, 2020.
8. The Linux Documentation Project. *Advanced Bash-Scripting Guide*. URL: <https://tldp.org/LDP/abs/html/>
9. LinuxCommand.org *The Linux Command Line* URL: <https://linuxcommand.org/tlcl.php>
10. Jura P. Master Laravel 11 & PHP: From Beginner to Advanced, 2020. <https://ua.udemy.com>. URL: <https://ua.udemy.com/course/laravel-beginner-fundamentals/learn/lecture/46135863#overview> (date of access: 08.09.2024).
11. Teli I. DevOps Beginners to Advanced with Projects, 2022. *A Technology School*. URL: <https://ua.udemy.com/course/decodingdevops/learn/lecture/26462854#overview> (date of access: 07.12.2024).

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

12. Valentin Despa. Introduction to Jenkins, CI/CD, and DevOps for Beginners (2024), 2024. *YouTube*. URL: <https://www.youtube.com/watch?v=soIpt3c7tVE> (date of access: 10.12.2024).
13. Viljoen R. NGINX Fundamentals: High Performance Servers from Scratch, 2020. *https://ua.udemy.com*. URL: <https://ua.udemy.com/course/nginx-fundamentals/learn/lecture/3843610#overview> (date of access: 18.10.2024).
14. Таненбаум, Е. Комп'ютерні мережі (5-е видання). – СПб: Пітер, 2013. – 960 с. ISBN: 978-5-496-00554-3
15. Cisco Networking Academy. Introduction to Networks. URL: <https://www.netacad.com/>
16. Thomas A. Limoncelli, Strata R. Chalup, Christina J. Hogan
The Practice of System and Network Administration (3rd ed.). – Addison-Wesley, 2016. ISBN: 978-0-321-94731-5
17. NGINX as an HTTP Load Balancer
URL: <https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/>
18. RFC 7230. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing
URL: <https://datatracker.ietf.org/doc/html/rfc7230>
19. Owen, Kelsey *YAML Essentials: A quick and practical guide to YAML*. Leanpub, 2021. URL: <https://leanpub.com/yamlessentials>
20. Carla Schroder
Linux Cookbook: Tips and Techniques for Everyday Use. – O'Reilly Media, 2004. – 608 p.
ISBN: 978-0-596-00640-9

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Додаток А – Jenkinsfile

```

pipeline {
  agent { label 'dev' }

  environment {
    // Variable for build image
    IMAGE_NAME = "kmowd/quiz-test-app"
    IMAGE_TAG = "${BUILD_NUMBER}"
    CONTAINER_NAME = "quiz-app"

    // Variable for setup .env file
    DB_CONNECTION = credentials('db-connection')
    DB_HOST = credentials('db-host')
    DB_PORT = credentials('db-port')
    DB_DATABASE = credentials('db-database')
    DB_USERNAME = credentials('db-username')
    DB_PASSWORD = credentials('db-password')
  }

  stages {

    stage('Clone Repository') {
      steps {
        git branch: 'develop', url: 'git@gitlab.com:kmow/quiz-knowanswer-laravel.git', credentialsId: 'GitLab'
      }
    }

    stage('Setup Environment') {
      steps {
        sh '''
          # Create .env file
          cp .env.example .env

          # Update database configuration
          sed -i "s/DB_CONNECTION=.*DB_CONNECTION=${DB_CONNECTION}/" .env
          sed -i "s/# DB_HOST=.*DB_HOST=${DB_HOST}/" .env
          sed -i "s/# DB_PORT=.*DB_PORT=${DB_PORT}/" .env
          sed -i "s/# DB_DATABASE=.*DB_DATABASE=${DB_DATABASE}/" .env
          sed -i "s/# DB_USERNAME=.*DB_USERNAME=${DB_USERNAME}/" .env
          sed -i "s/# DB_PASSWORD=.*DB_PASSWORD=${DB_PASSWORD}/" .env
          '''
      }
    }

    stage('Clear Old Docker Containers & Images') {
      steps {
        sh '''
          echo "Stopping all containers..."
          docker ps -q | xargs -r docker stop

          echo "Removing all containers..."
          docker ps -a -q | xargs -r docker rm

          echo "Removing all Docker images..."
          docker images -q | xargs -r docker rmi -f

          echo "Removing dangling images (if any)..."
          docker images -f "dangling=true" -q | xargs -r docker rmi -f
          '''
      }
    }
  }
}

```

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

```

    }

    stage('Build Docker Image') {
        steps {
            script {
                dockerImage = docker.build("${IMAGE_NAME}:${IMAGE_TAG}")
            }
        }
    }

    stage('Run Docker Container') {

        steps {
            script {
                dockerImage.run("-p 8080:80 --name ${CONTAINER_NAME}")
            }
        }
    }

    stage('Push Docker Image') {

        steps {
            withDockerRegistry([credentialsId: 'DockerHub', url: ""]) {
                script {
                    dockerImage.push("${IMAGE_TAG}")
                    dockerImage.push("latest")
                }
            }
        }
    }

}

post {
    success {
        echo "Build [${BUILD_NUMBER}] of job [${JOB_NAME}] on [${NODE_NAME}] COMPLETED SUCCESSFULLY!"
    }
    failure {
        echo "Build [${BUILD_NUMBER}] of job [${JOB_NAME}] on [${NODE_NAME}] FAILED!!!"
    }
}
}

```

Додаток Б – Dockerfile.

```

# Use the official PHP image with Apache
FROM php:8.2-apache

```

```

# Set working directory
WORKDIR /var/www/html

```

```

# Install system dependencies
RUN apt-get update && apt-get install -y \
    git \
    curl \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip \
    nodejs \
    npm

```

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

```

# Clear cache
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Install PHP extensions
RUN docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd

# Get latest Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Enable Apache mod_rewrite
RUN a2enmod rewrite

# Copy existing application directory contents
COPY . /var/www/html

# Copy existing application directory permissions
COPY --chown=www-data:www-data . /var/www/html

# Set the correct document root for Laravel
ENV APACHE_DOCUMENT_ROOT /var/www/html/public
RUN sed -ri -e 's!/var/www/html!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/sites-available/*.conf
RUN sed -ri -e 's!/var/www/!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/apache2.conf /etc/apache2/conf-available/*.conf

# Install PHP dependencies
RUN composer install --optimize-autoloader

# Install Node.js dependencies and build assets
RUN npm install && npm run build

# Generate key
RUN php artisan key:generate

# Set permissions
RUN chown -R www-data:www-data /var/www/html/storage /var/www/html/bootstrap/cache

# Start Apache
CMD ["apache2-foreground"]

```

Додаток В – Файл середовища конфігурації.

```

jenkins:
  ip: "192.168.56.10"
  hostname: "jenkins-server"
  backup_folder: '/home/vagrant/jenkins_backups'
  memory: "1024"
  cpus: 1

web:
  ip: "192.168.56.20"
  hostname: 'web-server'
  container:
    name: 'quiz_app'
    image: 'kmowd/quiz-test-app:latest'
    run: 'always'
  ports:
    host: 8080
    guest: 80
  memory: "1024"
  cpus: 1

web-2:
  ip: "192.168.56.21"

```

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

```

hostname: 'web-2-server'
container:
  name: 'quiz_app'
  image: 'kmowd/quiz-test-app:latest'
  run: 'always'
ports:
  host: 8080
  guest: 80
memory: "512"
cpus: 1

```

```

database:
  ip: "192.168.56.30"
  hostname: "mysql-server"
  root_password: "root"
  user: "maks"
  user_password: "vagrant"
  db_name: "quiz_application_db"
  memory: "1024"
  cpus: 1

```

```

proxy:
  ip: "192.168.56.100"
  hostname: "proxy-server"
  forwarded_ports:
    - guest: 8080
      host: 5680
      id: jenkins
    - guest: 80
      host: 5600
      id: web-server
  memory: "512"
  cpus: 1

```

```

credentials:
  dockerhub:
    username: "dockerhub_username"
    password: "dockerhub_password"

```

Додаток Г-1 – Vagrantfile для jenkins-server.

```

require 'yaml'
# Load configuration from YAML file
config_file = File.join(File.dirname(__FILE__), '..', 'config.yml')
if File.exist?(config_file)
  config = YAML.load_file(config_file)
  jenkins = config['jenkins']
else
  exit
end

Vagrant.configure("2") do |config|

  config.vm.define "jenkins-server" do |vm|
    vm.box = "ubuntu/jammy64"
    vm.network "private_network", ip: jenkins['ip']
    vm.hostname = jenkins['hostname']
    vm.provider "virtualbox" do |vb|
      vb.memory = jenkins['memory']
      vb.cpus = jenkins['cpus']
    end

    vm.synced_folder "jenkins_backups/", jenkins['backup_folder']
  end
end

```

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

```

vm.vm.provision :shell,
  name: "install-jenkins",
  path: "../provisions/install-jenkins.sh",
  run: 'once'
vm.vm.provision :shell,
  name: "gen-ssh-key",
  path: "../provisions/gen-ssh-key-jenkins.sh",
  run: 'once'
vm.vm.provision :shell,
  name: "backup-jenkins",
  path: "../provisions/backup-jenkins.sh",
  args: [jenkins['backup_folder']],
  run: 'never'
end
end

```

Додаток Г-2 – Vagrantfile для web-server.

```

require 'yaml'
# Load configuration from YAML file
config_file = File.join(File.dirname(__FILE__), '..', 'config.yml')
if File.exist?(config_file)
  config = YAML.load_file(config_file)
  web = config['web']
  container = web['container']
  dockerhub = config['credentials']['dockerhub']
else
  exit
end

Vagrant.configure("2") do |config|

config.vm.define "web-server" do |vm|
  vm.vm.box = "ubuntu/jammy64"
  vm.vm.network "private_network", ip: web['ip']
  vm.vm.hostname = web['hostname']
  vm.vm.provider "virtualbox" do |vb|
    vb.memory = web['memory']
    vb.cpus = web['cpus']
  end
  vm.vm.provision :shell, name: "install-docker", path: "../provisions/install-docker.sh"
  vm.vm.provision :shell, name: "init-jenkins-agent", path: "../provisions/init-jenkins-agent.sh"
  vm.vm.provision :shell,
    name: "start-container",
    path: "../provisions/start-container.sh",
    args: [container['name'], container['image'], container['ports']['host'], container['ports']['guest'],
    dockerhub['username'], dockerhub['password']],
    run: container['run']
  end
end

```

Додаток Г-3 – Vagrantfile для database-server.

```

require 'yaml'
# Load configuration from YAML file
config_file = File.join(File.dirname(__FILE__), '..', 'config.yml')
if File.exist?(config_file)
  config = YAML.load_file(config_file)
  db = config['database']
else

```

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

```
exit
end
```

```
Vagrant.configure("2") do |config|
```

```
config.vm.define "mysql-server" do |vm|
  vm.vm.box = "ubuntu/jammy64"
  vm.vm.network "private_network", ip: db['ip']
  vm.vm.hostname = db['hostname']
  vm.vm.provider "virtualbox" do |vb|
    vb.memory = db['memory']
    vb.cpus = db['cpus']
  end
  vm.vm.provision :shell,
    name: "install-mysql",
    path: "../provisions/install-mysql.sh",
    args: [db['root_password'], db['user'], db['user_password'], db['db_name']]
  vm.vm.provision :shell, name: "set-bind-address", path: "../provisions/set-bind-address.sh"
end
end
```

Додаток Г-4 – Vagrantfile для proxy-server.

```
require 'yaml'
# Load configuration from YAML file
config_file = File.join(File.dirname(__FILE__), '..', 'config.yml')
if File.exist?(config_file)
  config = YAML.load_file(config_file)
  proxy = config['proxy']
  forwarded_ports = proxy.fetch('forwarded_ports', [])
else
  exit
end
```

```
Vagrant.configure("2") do |config|
```

```
config.vm.define "proxy-nginx-server" do |vm|
  vm.vm.box = "ubuntu/jammy64"
  vm.vm.network "private_network", ip: proxy['ip']
```

```
  forwarded_ports.each do |port_rule|
    config.vm.network "forwarded_port",
      guest: port_rule['guest'],
      host: port_rule['host'],
      id: port_rule['id']
  end
```

```
  vm.vm.hostname = proxy['hostname']
  vm.vm.provider "virtualbox" do |vb|
    vb.memory = proxy['memory']
    vb.cpus = proxy['cpus']
  end
  vm.vm.provision :shell, name: "install-nginx", path: "../provisions/install-nginx.sh"
  vm.vm.synced_folder "nginx-config/", "/etc/nginx/sites-enabled/"
end
end
```

		Кос М. О.			ККІКБ.КР.Б – 123 – 05 – ПЗ	Арк.
		Головня О. С.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

