

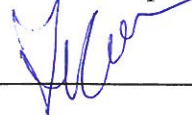
Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА
до випускної кваліфікаційної роботи бакалавра
на тему: «Проект побудови та розгортання
високодоступного веб-стеку на основі контейнерів
Docker»

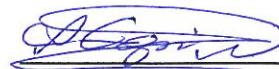
Виконав студент 4-го курсу групи КІ-19-1
спеціальності 123 «Комп'ютерна інженерія»

 Олег Лузік

Керівник старший викладач кафедри КІ та КБ

 Ігор Фальковський

Рецензент завідувач кафедри КІ та КБ, к.т.н., доцент

 Андрій Єфіменко

Житомир – 2023

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

1 березня 2023 р.

ЗАВДАННЯ

на випускну кваліфікаційну роботу бакалавра

Студент: Лузік Олег Сергійович

Тема роботи: «Проект побудови та розгортання високодоступного веб-стеку на основі контейнерів Docker»

затверджена Наказом університету від 1 березня 2023 р. № 127/с

Термін здачі студентом закінченої роботи 27 червня 2023 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

Предмет дослідження: побудова та розгортання високодоступного веб-стеку на основі контейнерів Docker.

Об'єкт дослідження: контейнеризація, Docker, веб-стек, високодоступність, CI/CD процес, Kubernetes, Helm.

Обладнання та програмне забезпечення:

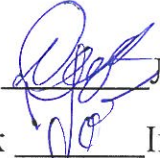
1. Docker, Kubernetes, Helm для контейнеризації компонентів веб-стеку

2. Gitlab CI/CD для автоматизації розгортання веб-стеку в Kubernetes

3. Інструменти для розробки веб-додатків: React, NodeJS та PostgreSQL

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Фальковський І. Г.	01.03.2023	01.03.2023
2	Фальковський І. Г.	01.03.2023	01.03.2023
3	Фальковський І. Г.	01.03.2023	01.03.2023

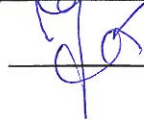
Студент  Лузік Олег

Керівник  Ігор Фальковський

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	20.03.2023 – 01.04.2023	
2	Написання вступної частини кваліфікаційної роботи	01.04.2023 – 03.04.2023	
3	Аналіз основних теоретичних про контейнеризацію	03.04.2023 – 11.04.2023	
4	Вибір технологій для веб-стека	11.04.2023 – 18.04.2023	
5	Написання Helm чарта	18.04.2023 – 24.04.2023	
6	Підключення Kubernetes кластера	24.04.2023 – 01.05.2023	
7	Налаштування CI/CD конвеєра	01.04.2023 – 08.05.2023	
8	Написання висновків	16.05.2023 – 20.05.2023	
9	Оформлення кваліфікаційної роботи	20.05.2023 – 25.05.2023	

Студент  Лузік Олег

Керівник  Ігор Фальковський

АНОТАЦІЯ

Пояснювальна записка містить: 73 сторінки текстової частини, 29 рисунків та 20 джерел.

Метою роботи є дослідження та розробка інфраструктури CI/CD для розгортання веб-стеку в середовищі Kubernetes з використанням контейнеризації за допомогою Docker та інструменту Helm.

В даній роботі проведено огляд теоретичних відомостей про контейнеризацію, Docker і Kubernetes. Розглянуто особливості проектування високодоступного веб-стеку, включаючи вибір технологій та інструментів, архітектуру системи та вибір інструменту для автоматизації CI/CD процесу. Налаштовано CI/CD пайплайн для розгортання веб-стеку в Kubernetes кластері за допомогою Helm.

Ключові слова: CI/CD, РОЗГОРТАННЯ, ВЕБ-СТЕК, KUBERNETES, HELM, КОНТЕЙНЕРИЗАЦІЯ, АВТОМАТИЗАЦІЯ, ТЕСТУВАННЯ.

					КІК.КР.Б – 123 – 11 – ПЗ									
Змн.	З	Арк.	Ар	№ док.	№	Підпис	Під	Дата						
Розроб.				Лузік О. С.				20.06.23	Проект побудови та розгортання високодоступного веб-стеку на основі контейнерів Docker		Літ.	Арк.	Аркуші	
Керівник				Фальковський І. Г.				20.06.23					4	73
Реценз.				Єфіменко А.А.				20.06.23			ФІКТ, Гр. КІ-19-1			
Н. контр.				Фальковський І. Г.				20.06.23						
Затверд.				Єфіменко А.А.				20.06.23						

ANNOTATION

The explanatory note contains: 73 pages of text, 30 figures and 20 sources.

The aim of the work is to research and develop a CI/CD infrastructure for deploying a web stack in a Kubernetes environment using containerisation with Docker and the Helm tool.

This paper provides an overview of theoretical information about containerization, Docker and Kubernetes. The features of designing a highly available web stack are considered, including the choice of technologies and tools, system architecture, and the choice of a tool for automating the CI/CD process. A CI/CD pipeline for deploying a web stack in a Kubernetes cluster using Helm.

Keywords: CI/CD, DEPLOYMENT, WEB STACK, KUBERNETES, HELM, CONTAINERISATION, AUTOMATION, TESTING.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ВІДОМОСТІ.....	9
1.1 Вступ до контейнеризації.....	9
1.2 Docker.....	11
1.3 Огляд Kubernetes.....	20
Висновки до першого розділу	25
РОЗДІЛ 2 ПРОЕКТУВАННЯ ВИСОКОДОСТУПНОГО ВЕБ-СТЕКУ	26
2.1 Вибір технологій та інструментів	26
2.2 Архітектура високодоступного веб-стеку.....	29
2.3 Огляд та вибір інструменту для автоматизації CI/CD процесу	34
Висновки до розділу 2	39
РОЗДІЛ 3 НАЛАШТУВАННЯ CI/CD ДЛЯ РОЗГОРТАННЯ ПРОЕКТУ В KUBERNETES КЛАСТЕРІ ЗА ДОПОМОГОЮ HELM.....	40
3.1 Створення Docker-файлів.....	40
3.2 Використання Helm для керування розгортанням та управлінням релізами	41
3.3 Налаштування CI/CD пайплайну для розгортання веб-стеку в Kubernetes кластері	44
3.4 Перевірка працездатності.....	51
Висновки до розділу 3	56
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	59
ДОДАТКИ.....	61

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – Операційна система

ВМ – Віртуальна машина

CI – Continuous Integration

CD – Continuous Delivery/Deployment

gRPC – Google Remote Procedure Call

CRI – Container Runtime Interface

CNI – Container Networking Interface

REST – Representational State Transfer

CRUD – Create, Read, Update Delete

JSON – JavaScript Object Notation

YAML – YAML Ain't Markup Language

OCI – Open Container Initiative

CSI – Container Storage Interface

GCP – Google Cloud Platform

GKE – Google Kubernetes Engine

DNS – Domain Name System

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. У сучасному інформаційному суспільстві, де веб-додатки відіграють ключову роль у взаємодії користувачів з різноманітними сервісами, питання стабільності, швидкості та безперервності роботи цих додатків стають особливо важливими. Забезпечення високодоступності та автоматизованого розгортання веб-додатків є нагальним завданням для розробників та адміністраторів систем. Використання контейнеризації з використанням Docker та інструментів CI/CD для розгортання веб-стеку стає все більш поширеним та актуальним підходом для забезпечення швидкого та надійного впровадження змін у веб-додатки.

Метою кваліфікаційної роботи є реалізація проекту побудови та розгортання високодоступного веб-стеку на основі контейнерів Docker, а також налаштування процесу CI/CD для автоматичного розгортання цього проекту в Kubernetes кластері за допомогою Helm. Основними завданнями дослідження є вивчення теоретичних основ контейнеризації, високодоступності та CI/CD, розробка архітектури веб-стеку з використанням Docker та інтеграція його з Kubernetes та Helm.

Об'єктом дослідження є процес побудови та розгортання високодоступного веб-стеку на основі контейнерів Docker та використання CI/CD для автоматичного розгортання цього стеку в Kubernetes кластері.

Предметом дослідження є технології контейнеризації з використанням Docker, веб-стеки та їх архітектура, процеси CI/CD та інструменти для автоматичного розгортання веб-додатків в Kubernetes кластері. Особлива увага приділятиметься налаштуванню проекту з використанням Helm для спрощення розгортання та управління контейнеризованим веб-стеком.

Результати дослідження та практичної реалізації цієї роботи виявляться корисними для розробників, системних адміністраторів та фахівців з інформаційних технологій, які займаються розгортанням та управлінням веб-додатками.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Вступ до контейнеризації

Програми є основою бізнесу. Якщо програми ламаються - бізнес страждає. Іноді навіть розоряється. Ці твердження стають ще актуальнішими з кожним днем. Більшість програм працюють на серверах. Раніше можна було запускати лише одну програму на сервері. У світі відкритих операційних систем Windows і Linux просто не було технологій для безпечного та ефективного запуску кількох програм на одному сервері.

У підсумку, історія склалася наступним чином. Кожного разу, коли у бізнесу з'являлася нова програма, ІТ-відділ купував новий сервер. Зазвичай, ніхто не знав вимог до продуктивності нової програми, що змушувало ІТ-відділ робити припущення при виборі моделі та розміру сервера для покупки.

В результаті, спеціалісти з ІТ робили єдине, що було у їхніх силах - купували великі та швидкі сервери, які коштували значну суму грошей. Зрештою, останнє, чого хотів бізнес, - це мати недостатньо потужні сервери, які не здатні виконувати транзакції, що призводило до втрати клієнтів та доходу. Це призвело до того, що сервери займали занадто багато непотрібної потужності та працювали лише на 5-10% своєї потенційної продуктивності. Це була трагічна втрата капіталу компанії та екологічних ресурсів.

Серед усього цього компанія VMware, Inc. подарувала світу ВМ. І майже за одну ніч світ став набагато кращим. Нарешті з'явилася технологія, яка дозволила нам безпечно та надійно запускати кілька бізнес-додатків на одному сервері.

Незважаючи на це, ВМ далекі від ідеальності. Факт того, що для кожної віртуальної машини потрібна окрема ОС – це є серйозним недоліком. Кожна ОС вимагає використання процесора, оперативної пам'яті та інших ресурсів, які можна було б ефективніше використати для більшої кількості програм. Кожна

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

ОС потребує патчів та моніторингу, а в деяких випадках навіть ліцензій. Все це витрачає час та ресурси.

У моделі віртуальних машин є проблеми. Віртуальні машини завантажуються повільно, а їхню переносимість можна було б покращити. Міграція та переміщення робочих навантажень між гіпервізорами та хмарними платформами виявляються складнішими, ніж цього хотілося б.

Протягом тривалого часу великі гравці веб-індустрії, такі як Google, використовували контейнерні технології для усунення недоліків моделі віртуальної машини.

У контейнерній моделі контейнери фактично є аналогом ВМ. Проте основна відмінність полягає в тому, що контейнери не вимагають окремої повноцінної ОС. Насправді, всі контейнери на одному хості спільно використовують ОС хоста. Це дозволяє значно звільнити системні ресурси, такі як процесор, оперативна пам'ять і диск. Крім того, це зменшує потенційні витрати на ліцензування та зменшує накладні витрати на оновлення операційної системи та інші обслуговування. Результатом є значна економія часу, ресурсів і капіталу.

Контейнери швидко запускаються та легко переносяться. Переміщення контейнеризованих робочих навантажень з ноутбука в хмарове середовище, а потім на віртуальні машини або фізичне обладнання в центрі обробки даних – це досить просте завдання.

Сучасні контейнери почалися у світі Linux і є продуктом величезної кількості роботи багатьох людей протягом тривалого періоду часу. Деякі з основних технологій, які забезпечили масове зростання контейнерів за останні роки, включають: namespaces, cgroups, union filesystems та, звичайно, Docker.

Незважаючи на все це, контейнери залишалися складними та недоступними для більшості організацій. Лише після появи Docker контейнери стали фактично демократизованими та доступними для більшості. Docker був

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

тією магією, яка зробила контейнери Linux доступними. Іншими словами, Docker, Inc. зробив контейнери простими.

Протягом останніх кількох років Microsoft наполегливо працювала над впровадженням технологій Docker і контейнерів на платформу Windows.

На момент написання, контейнери Windows доступні як Windows Desktop, так і для платформ Windows Server. Щоб досягти цього, Microsoft тісно співпрацювала з Docker, Inc. та відкритою спільнотою. Основні технології ядра Windows, необхідні для використання контейнерів, називаються контейнерами Windows. Таким чином, досвід роботи з Docker на Windows майже не відрізняється від досвіду на Linux. Розробники та системні адміністратори, які вже знайомі з набором інструментів Docker на платформі Linux, можуть почуватись комфортно, використовуючи контейнери Windows.

Важливо розуміти, що запущений контейнер використовує ядро хост-машина, на якій він працює. Це означає, що контейнерна програма Windows не буде працювати на хості Docker з базовою системою Linux, і навпаки, контейнери Windows потребують хоста Windows, а контейнери Linux - хоста Linux, але це не завжди так.

Контейнери Linux можна запускати на машинах з операційною системою Windows. Наприклад, Docker Desktop, що працює на Windows, надає два режими роботи: "контейнери Windows" і "контейнери Linux". Залежно від версії Docker Desktop, контейнер Linux запускається або всередині легковагової віртуальної машини Hyper-V, або за допомогою WSL.

1.2 Docker

Docker — це програмне забезпечення, яке працює в Linux і Windows. Він створює, керує та навіть може оркеструвати контейнери. Програмне забезпечення наразі складається з різних інструментів проекту з відкритим кодом. Docker, Inc. — це компанія, яка створила цю технологію та продовжує створювати

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

технології та рішення. Коли більшість людей говорять про Docker, вони мають на увазі технологію, яка запускає контейнери. Однак є принаймні три речі, про які слід пам'ятати, коли йдеться про Docker як про технологію: середовище виконання(runtime), daemon, orchestrator.

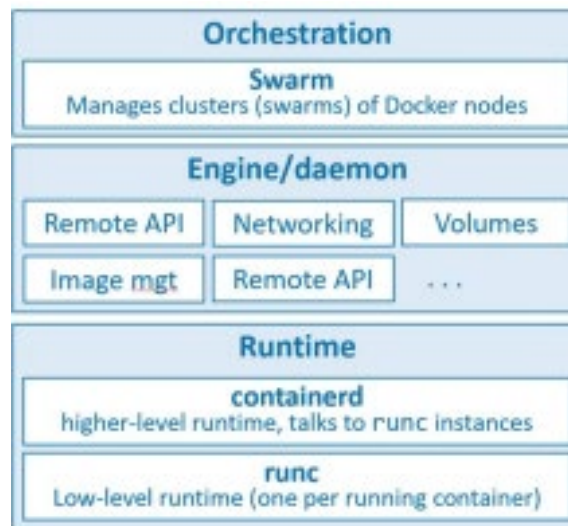


Рисунок 1.1 – Архітектура Docker[5]

Середовище виконання працює на найнижчому рівні та відповідає за запуск і зупинку контейнерів (це включає створення всіх конструкцій ОС, таких як namespaces та cgroups). Docker реалізує багаторівневу архітектуру середовища виконання з високорівневими та низькорівневими середовищами виконання, які працюють разом.

Середовище виконання низького рівня називається runc і є еталонною реалізацією специфікації середовища виконання Open Containers Initiative (OCI). Його робота полягає в тому, щоб взаємодіяти з основною ОС і запускати та зупиняти контейнери. Кожен запущений контейнер на вузлі Docker має екземпляр runc, який ним керує.

Середовище виконання вищого рівня називається containerd. Containerd робить набагато більше, ніж runc. Він керує всім життєвим циклом контейнера, включаючи завантаження зображень, створення мережевих інтерфейсів і керування екземплярами runc нижчого рівня.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Типова інсталяція Docker має один процес контейнера (dockercontainerd), який керує екземплярами runc (docker-runc), пов'язаними з кожним запущеним контейнером.

Демон Docker (dockerd) знаходиться над контейнером і виконує завдання вищого рівня, такі як; відкриття віддаленого API Docker, керування зображеннями, керування томами(volumes), керування мережами тощо. Основне завдання демона Docker — забезпечити простий у використанні стандартний інтерфейс, який абстрагує нижчі рівні.

Docker складається з двох основних компонентів: client Docker та daemon Docker (іноді його називають "Docker engine").

Демон реалізує середовище виконання, API та все інше, що потрібно для запуску Docker. У інсталяції Linux за замовчуванням клієнт спілкується з демоном через локальний сокет IPC/Unix за адресою /var/run/docker.sock. У Windows це відбувається через іменований канал за адресою pipe:////./pipe/docker_engine. Після встановлення ви можете використовувати команду docker version, щоб перевірити, чи клієнт і демон (сервер) працюють і спілкуються один з одним[4].

Механізм Docker є основним програмним забезпеченням, яке запускає та керує контейнерами. Ми часто називаємо це просто Docker. Механізм Docker має модульну конструкцію та складається з багатьох невеликих спеціалізованих інструментів. Там, де це можливо, вони ґрунтуються на відкритих стандартах, таких як стандарт OCI. У багатьох відношеннях Docker Engine схожий на автомобільний двигун — обидва модульні та створені шляхом з'єднання багатьох невеликих спеціалізованих частин: API, драйвера виконання, середовища виконання, прокладок (shim) тощо. Основні компоненти, що складають двигун Docker: демон Docker, containerd, runc і різні плагіни, мережеві та сховища. Разом вони створюють і запускають контейнери.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Коли Docker вперше був випущений, його механізм складався з двох основних компонентів: Docker-демона і LXC.

Docker-демон був монолітним двійковим файлом, що включав в себе код для клієнта Docker, API, середовища виконання контейнерів, створення образів та багато іншого. Залежність від LXC становила проблему вже з самого початку.

По-перше, LXC було специфічним для Linux, що створювало проблему для проекту, який прагнув бути багатоплатформеним.

По-друге, залежати від зовнішнього інструменту для такого важливого компонента проекту було великим ризиком, який міг завадити подальшому розвитку.

З плином часу монолітність Docker-демона стала все більш проблематичною: упровадження нововведень стало складнішим, швидкість роботи погіршувалась, що не задовольняло екосистему. Компанія Docker усвідомлювала ці труднощі і зробила великі зусилля, щоб розбити монолітний демон на модулі та перевести його у модульну архітектуру. Мета цих зусиль полягала у видаленні функціональності виконання контейнерів з демона та перетворенні його на невеликі спеціалізовані інструменти.

Як вже було згадано, runc є еталонною реалізацією специфікації OCI container-runtime-spec. Компанія Docker, Inc. активно брала участь у визначенні цієї специфікації та розробці runc. Основна мета runc полягає в запуску контейнерів, і вона виконує це дуже добре та швидко. Оскільки runc є CLI-обгорткою, він фактично є автономним інструментом для виконання контейнерів.

У рамках спроби відокремити функціональність виконання контейнерів від Docker-демона, весь код, пов'язаний з виконанням контейнерів, був видалений з демона та перетворений у новий інструмент під назвою containerd. Його основна мета - керувати операціями життєвого циклу контейнера, такими як запуск, зупинка, пауза, видалення та інші. Як вже зазначалося, вихідний принцип контейнера полягав у його легкості та спрощенні, обмежуючи його

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

функціональність до виконання однієї задачі - операцій життєвого циклу контейнера. Однак з часом його функціональність розширилась, включаючи підтримку образів, томів та мереж.

Однією з причин додавання додаткової функціональності було полегшення використання Docker в інших проектах. Наприклад, проекти, такі як Kubernetes, користуються контейнерами для виконання додаткових дій, таких як надсилання та отримання образів. Таким чином, контейнер може бути включений до таких проектів, як Kubernetes, але тільки для використання необхідних частин, які потрібні вашому проекту.

Найпоширенішим способом запуску контейнерів є використання Docker CLI. Коли команда вводиться в Docker CLI, клієнт Docker перетворює їх на відповідний запит до API. API реалізоване у демоні і може бути доступне через локальний сокет або мережу. У Linux це сокет `/var/run/docker.sock`, а в Windows - `\pipe\docker_engine`. Коли демон отримує команду створити новий контейнер, він звертається до `containerd`. Демон взаємодіє з контейнердом через API у стилі CRUD за допомогою gRPC.

Незважаючи на свою назву, `containerd` насправді не може створювати контейнери. Для цього використовується інструмент `runc`. Він перетворює необхідний Docker-image на OCI-bundle і повідомляє `runc`, щоб він використав його для створення нового контейнера. `runc` взаємодіє з ядром операційної системи, щоб об'єднати всі необхідні конструкції для створення контейнера, такі як `namespaces`, `cgroups` тощо. Процес контейнера запускається як дочірній процес `runc`, і як тільки він запускається, робота `runc` завершується. Таким чином, контейнер успішно запущений.

Docker image (образ) - це пакет, який містить все необхідне для запуску програми, включаючи код програми, залежності та компоненти операційної системи. Якщо у вас є образ Docker програми, ви можете запустити його на будь-

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

якому комп'ютері з встановленим Docker. Ви можете розглядати образи Docker як шаблони віртуальних машин.

Образи складаються з декількох шарів, які поступово накладаються один на одного і представлені як єдиний об'єкт. У середині образу знаходиться ОС та всі файли та залежності, необхідні для запуску програми. Оскільки контейнери повинні бути швидкими та легкими, образи, як правило, повинні бути невеликими.

Ви можете зупинити контейнер і створити з нього новий образ. У зв'язку з цим образи вважаються конструкціями часу збірки, тоді як контейнери - конструкціями часу виконання. Після запуску контейнера з образу ці дві конструкції стають залежними одна від одної, і ви не зможете видалити образ, поки існує активний контейнер, що його використовує. Спроба видалити образ, не зупинивши та не знищивши всі контейнери, які його використовують, призведе до помилки.

Призначенням контейнера є запуск однієї програми або служби. Це означає, що йому потрібен лише код і залежності програми/сервісу, який він запускає - більше йому нічого не потрібно. Це призводить до того, що невеликі образи позбавлені всіх незначних компонентів.

Образ також не містить ядра - всі контейнери, що запущені на хості Docker, мають спільний доступ до ядра хоста. З цих причин іноді говорять, що образи містять "достатню ОС". Офіційний Docker-образ Alpine Linux має розмір близько 5 МБ і є чудовим прикладом того, наскільки малі можуть бути Docker-образи. Образи на базі Windows, як правило, більші за образи на базі Linux через особливості роботи ОС Windows.

Процес отримання образів на хост Docker називається завантаженням (pull). У кожній команді ми повинні вказати, який саме образ потрібно завантажити. Ми зберігаємо образи у централізованих місцях, які називаються реєстрами образів. Це дозволяє легко ділитися ними та отримувати до них доступ.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Найпоширенішим реєстром є Docker Hub. Існують також інші реєстри, включаючи сторонні реєстри та захищені локальні реєстри.

Звернутися до зображень з офіційних сховищ дуже просто: достатньо вказати назву сховища та мітку, розділені двокрапкою (:). Формат команди для завантаження образу з офіційного репозиторію має вигляд:

```
$ docker image pull <репозиторій>:<тег>.
```

Якщо потрібно отримати образи зі сторонніх реєстрів, вам потрібно додати до назви сховища DNS-ім'я реєстру:

```
$ docker image pull mcr.microsoft.com/powershell:latest
```

Один образ може мати скільки завгодно тегів. Це пов'язано з тим, що теги є довільними буквено-цифровими значеннями, які зберігаються як метадані разом зі знімком образу. Образ Docker - це просто набір слабо пов'язаних шарів, доступних лише для читання, кожен з яких складається з одного або декількох файлів. Docker дбає про складання цих шарів і представлення їх як єдиного об'єкта[7].

Усі образи в Docker починаються з базового шару, а по мірі внесення змін і додавання нового вмісту зверху додаються нові шари. Кілька образів можуть спільно використовувати шари, що призводить до економії простору і підвищення продуктивності.

В певному сенсі, сам образ є лише конфігураційним файлом, у якому перелічені шари та деякі метадані. Шари представляють місце, де зберігаються дані (файли, код і т.д.). Кожен шар є повністю незалежним і не нічого не знає про те, що він є частиною більшого зображення.

Коли створюється контейнер, образ використовується як основа. Контейнер - це екземпляр образу під час його виконання. Так само, як ви шаблон віртуальної машини, можна запустити один або кілька контейнерів з одного образу. Основна відмінність між контейнером і ВМ полягає в тому, що

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

контейнери є швидшими і легшими - вони не запускають повноцінну ОС, як ВМ, а використовують ОС/ядро хоста, на якому вони запуснені. Крім того, контейнери часто базуються на мінімалістичних образах, які включають лише необхідне програмне забезпечення та залежності для виконання програми. Контейнери працюють, поки не завершиться програма, яку вони виконують.

Перше, що буде розглянуто тут, це фундаментальні відмінності між контейнером і ВМ. Як контейнерам, так і віртуальним машинам потрібен хост для запуску. Цим може бути все - від ноутбука і порожнього фізичного сервера в дата-центрі до екземпляра в публічному хмарному середовищі. На сьогоднішній день багато хмарних сервісів підтримують запуск контейнерів у безсерверних середовищах, де віртуалізація просто схована.

У моделі ВМ, фізичний сервер запускається і завантажується гіпервізором (пропускаючи BIOS, loader і т.д.). Після завантаження гіпервізор отримує собі всі фізичні ресурси сервера, такі як процесор, оперативна пам'ять, сховище та мережеві інтерфейси. Потім він розділяє ці апаратні ресурси на віртуальні версії, які виглядають як справжні ОС. Потім він упакує їх у програмну конструкцію, яка називається ВМ. Ми беремо ці ВМ і на кожному з них встановлюємо ОС та програми.

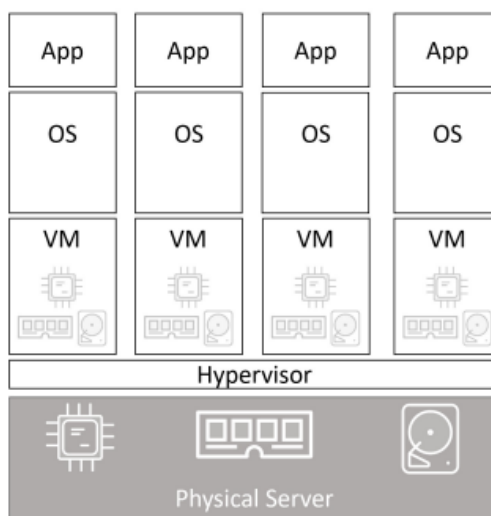


Рисунок 1.2 – Віртуалізація з використання гіпервізора[1]

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

У контейнерній моделі все відбувається трохи інакше. Сервер запускається і завантажується операційна система. У світі Docker це може бути Linux або сучасна версія Windows, яка підтримує контейнерні технології у своєму ядрі. Подібно до моделі віртуальних машин, операційна система приділяє собі всі апаратні ресурси. На верхньому рівні ОС ми встановлюємо контейнерний двигун, такий як Docker. Контейнерний двигун використовує ресурси операційної системи, такі як процесор, файлову систему та мережевий стек, і розділяє їх на ізольовані конструкції, які називаються контейнерами. Кожен контейнер має вигляд, поведінку і відчуття справжньої операційної системи. Усередині кожного контейнера запускаються програми.

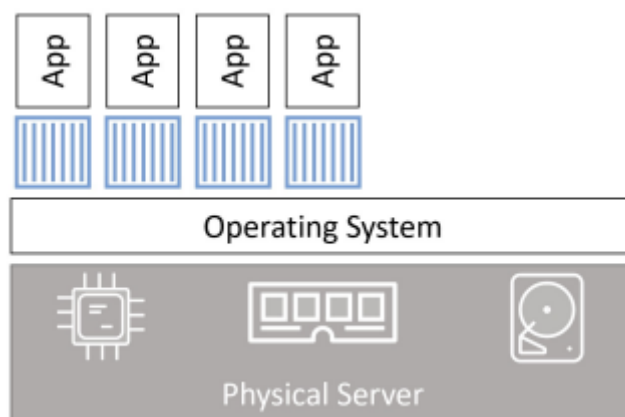


Рисунок 1.3 – Віртуалізація з використання контейнеризації[1]

На високому рівні, гіпервізори виконують апаратну віртуалізацію - вони розбивають фізичні апаратні ресурси на віртуальні версії, які називаються віртуальними машинами. З іншого боку, контейнери виконують віртуалізацію операційної системи - вони розбивають ресурси операційної системи на віртуальні версії, які називаються контейнерами.

Контейнер не може існувати без головного процесу, який є його основою. Це стосується як контейнерів Linux, так і контейнерів Windows - знищення головного процесу в контейнері призведе до знищення контейнера. Ви можете зупиняти, запускати, ставити на паузу і перезапускати контейнер стільки разів, скільки вам потрібно. Однак, якщо ви явно не видалите контейнер, ви ризикуєте

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

втратити дані, які знаходяться всередині контейнера. Навіть якщо ви зберігаєте дані поза контейнером у тому, ці дані залишаться після видалення контейнера.

У більшості контейнерів Linux запускається один процес. Щодо контейнерів Windows, ситуація трохи відрізняється, але вони також запускають один основний процес, і на них застосовуються наступні правила. Для зупинки контейнера Docker надсилає сигнал SIGTERM головному прикладному процесу всередині контейнера (PID 1).

Часто корисно використовувати політику перезапуску для контейнерів, що дозволяє Docker автоматично перезапускати контейнери після певних подій або випадків відмови.

1.3 Огляд Kubernetes

Kubernetes - це оркестратор додатків. Здебільшого він керує масштабованими контейнерними мікросервісними програмами у хмарному середовищі. Оркестратор - це система, що розгортає та керує програмами. Kubernetes може розгортати ваші додатки, динамічно масштабувати їх відповідно до потреб, самовідновлюватися після збоїв, здійснювати оновлення та відкати без простоїв та виконувати багато інших завдань. Краще за все в Kubernetes полягає в тому, що він автоматизує ці процеси[2].

Kubernetes був розроблений компанією Google на основі їхнього багатолітнього досвіду роботи з контейнерами. Він був переданий спільноті як проект з відкритим вихідним кодом і тепер є стандартною галузевою платформою для розгортання та керування хмарними додатками. Kubernetes працює у будь-якому хмарному або локальному середовищі обробки даних, абстрагуючись від конкретної інфраструктури. Це дає можливість створювати гібридні хмари, мігрувати додатки до хмари та з хмари, а також переміщати їх між різними хмарами. Kubernetes має відкритий вихідний код під ліцензією Apache 2.0 і належить до CNCF.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

Kubernetes схожий на будь-який інший кластер - група машин, на яких розміщуються додатки. На мові Kubernetes ці машини називаються "вузлами", і це можуть бути фізичні сервери, віртуальні машини, хмарні екземпляри, Raspberry Pi тощо. Кластер Kubernetes складається з двох основних компонентів: керуючого рівня (control plane) та робочих вузлів.

Керуючий рівень надає доступ до API, має планувальник для розподілу роботи і зберігає дані про кластер та додатки в постійному сховищі. Робочі вузли запускають користувацькі програми. Можна уявляти керуючий рівень як "мозок" кластера, а вузли - як "м'язи". У цій аналогії керуючий рівень відповідає за інтелектуальні функції, такі як планування, автоматичне масштабування та розгортання. Робочі вузли виконують щоденну важку роботу запуску та управління користувацькими додатками[8].

Керуючий вузол у Kubernetes запускає набір системних служб. За лаштунками ці сервіси включають сервер API, кластерне сховище, планувальник та контролери ядра. У найпростіших конфігураціях використовується один керуючий вузол. Однак рекомендується мати 3 або 5 вузлів, і їх розподіляти по різних зонах доступності. Також вважається гарною практикою не запускати користувацькі програми на керуючих вузлах.

Сервер API є центральною точкою в Kubernetes. Вся комунікація між компонентами повинна пройти через сервер API. Він надає RESTful API, до якого надсилається конфігураційні файли YAML через HTTPS. Ці YAML-файли, також відомі як маніфести, які описують бажаний стан програм, включаючи образ контейнера, відкриті порти та кількість реплік Pod.

Pod - це ізольоване середовище для запуску контейнерів. Самі по собі поди насправді не запускають програми - програми завжди запускаються в контейнерах. Pod є просто пісочницею для запуску одного або декількох контейнерів. Якщо ви запускаєте кілька контейнерів в поді, всі вони використовують одне і те ж середовище. Це включає мережевий стек, томи,

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

простір імен IPC, спільну пам'ять тощо. Поди також є мінімальною одиницею планування в Kubernetes[13].

Сховище кластера є єдиною частиною control plane, яка зберігає всю конфігурацію та стан кластера. Наразі для сховища кластера використовується etcd, популярна розподілена база даних. Вона є єдиним джерелом правди для кластера. Для забезпечення високої доступності рекомендується мати 3-5 реплік etcd та налагодити механізми відновлення.

Диспетчер контролерів відповідає за фонові контролери, які контролюють компоненти кластера та реагують на події. Диспетчер контролерів є «контролером контролерів». Кожен контролер відповідає за певну частину функціональності кластера та постійно спостерігає за змінами на сервері API, щоб забезпечити відповідність бажаному стану (desired state).

На високому рівні планувальник відстежує нові робочі завдання на сервері API та призначає їх робочим вузлам. Планувальник реалізує складну логіку, яка фільтрує непридатні вузли, які можуть виконати завдання. Система вибору є складною, але вона обирає вузол з найвищим рейтингом для виконання завдання[10].

Якщо ви запускаєте кластер на хмарній платформі, такій як AWS, Azure, GCP або Linode, у вашому керуючому рівні буде запущений менеджер хмарних контролерів. Його завдання - спростити інтеграцію з хмарними сервісами, такими як екземпляри, балансувальники навантаження та сховище.

Робочі вузли - це вузли, на яких виконуються користувацькі програми. Вони виконують три основні завдання:

1. Спостерігають на сервері API за новими робочими завданнями.
2. Виконують робочі завдання.
3. Звітують про виконану роботу на керуючий рівень через сервер API.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Kubelet є основним агентом Kubernetes і запускається на кожному робочому вузлі. При додаванні вузла до кластера, встановлюється kubelet, який потім реєструє вузол у кластері. Kubelet реєструє характеристики вузла, такі як процесор, пам'ять і сховище, у широкому пулі кластера.

Одне з основних завдань kubelet - стежити за сервером API на предмет нових робочих завдань. Кожного разу, коли він отримує таке завдання, він виконує його і зв'язується з площиною управління.

Kube-proxy працює на кожному вузлі і відповідає за мережеву логіку кластера. Він забезпечує унікальну IP-адресу для кожного вузла і реалізує правила маршрутизації і балансування трафіку в мережі Pod.

Крім площини управління та робочих компонентів, кожен кластер Kubernetes має внутрішню службу DNS, яка є важливою для виявлення сервісів.

Kubernetes базується на декларативній моделі та концепції бажаного стану. Файли маніфестів написані на простому YAML і описують, як повинні виглядати додатки. Це включає образи, кількість реплік, мережеві порти та оновлення. Замість складного скрипта для кожної репліки, Kubernetes просто виконує інструкції маніфесту для досягнення бажаного стану. Основні об'єкти Kubernetes включають наступні складові: Pod, Deployment, Service, Ingress, ConfigMap, Secret.

Deployment - це об'єкт Kubernetes більш високого рівня, який обертається навколо Pod і додає такі функції, як самовідновлення, масштабування, розгортання з нульовим часом простою (zero downtime) та відкат до попередніх версій. За лаштунками Deployment використовує ReplicaSets для виконання більшої частини роботи з подами. Насправді, ReplicaSet, створює, видаляє та керує подами, але Deployment вказує ReplicaSet, що робити.

Service забезпечує стабільну та надійну мережу для програм, розгорнутих на Kubernetes. На зовнішньому боці вони надають стабільне ім'я DNS, яке автоматично реєструється в кластерному DNS, а також стабільну віртуальну IP-

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

адресу. На внутрішній стороні вони балансують трафік між динамічним набором подів, які відповідають селектору міток. Вони також дозволяють вам відкривати елементи вашого додатку для зовнішнього світу, включаючи інтеграцію з хмарними балансувальниками навантаження.

Ingress - це доступ до декількох додатків через одну службу LoadBalancer. Після запуску контролера Ingress створюється та розгортається об'єкт Ingress, які є наборами правил, що регулюють маршрутизацію вхідного трафіку до додатків і служб у кластері Kubernetes. Він підтримує HTTP-маршрутизацію на основі хостів і шляхів.

ConfigMaps і Secrets - це власний спосіб розділення програм і конфігураційних даних у Kubernetes. ConfigMaps використовуються для параметрів конфігурації програми і навіть цілих конфігураційних файлів, тоді як Secrets призначено для конфіденційних даних конфігурації. Обидва типи можна вставити у контейнери під час виконання за допомогою різних конструкцій. Kubernetes не шифрує секрети у сховищі кластера або під час транзиту по мережі.

Окрім згаданих об'єктів, існують інші важливі об'єкти в Kubernetes[12]:

- StatefulSets – це розширення ReplicaSets, яке дозволяє керувати додатками зі збереженням стану.
- DaemonSets забезпечує запуск одного екземпляра пода на кожному вузлі кластера.
- Jobs виконується один раз для виконання певної роботи та зупиняється після завершення.
- CronJobs виконує задачі на основі розкладу Cron.

Kubernetes - це потужна система для автоматизації розгортання, керування та масштабування контейнеризованих додатків. Він надає широкий спектр можливостей, включаючи оркестрацію контейнерів, автоматичне масштабування, балансування навантаження, самовідновлення та управління ресурсами. Завдяки своїй гнучкості та розширюваності, Kubernetes став одним з

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

найпопулярніших інструментів для управління контейнеризованими додатками у хмарних та локальних середовищах. Використання Kubernetes дозволяє спростити розгортання та керування додатками, забезпечуючи більшу ефективність та надійність інфраструктури.

Висновки до першого розділу

В першому розділі були розглянуті теоретичні аспекти контейнеризації, Docker та огляд Kubernetes. Було з'ясовано, що контейнеризація є потужним інструментом для упаковки програмного забезпечення та його залежностей в легкі та переносні контейнери. Docker - це найпопулярніша платформа для розробки, доставки та запуску контейнеризованих додатків. Крім того, був проведений огляд Kubernetes - системи оркестрації контейнерів, яка надає масштабоване та автоматизоване управління контейнерами. Розділ надав загальне уявлення про основні концепції та інструменти, які будуть використовуватися у подальших розділах для побудови високодоступних та масштабованих веб-стеків.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2 ПРОЕКТУВАННЯ ВИСОКОДОСТУПНОГО ВЕБ-СТЕКУ

2.1 Вибір технологій та інструментів

Веб-стек (web stack) - це набір технологій, які використовуються для розробки веб-додатків та веб-сайтів. Він складається з комбінації програмних мов, фреймворків, серверних інструментів та баз даних, які працюють разом для забезпечення функціональності та відображення веб-додатку у браузері користувача.

Зазвичай веб-стек включає такі основні компоненти:

- Клієнтська частина (frontend) - це та частина веб-додатку, яку бачить користувач у своєму браузері. Вона зазвичай реалізована з використанням HTML, CSS та JavaScript, а також може включати фреймворки та бібліотеки, такі як React, Angular або Vue.js.
- Серверна частина (backend) - це та частина веб-додатку, яка виконується на сервері і забезпечує обробку запитів від клієнтської частини. Для розробки серверної частини можуть використовуватись мови програмування, такі як Python, Java, PHP або JavaScript (за допомогою Node.js). Також використовуються фреймворки, такі як Django, Flask, Spring або Express.js, для спрощення розробки та забезпечення швидкості та безпеки.
- База даних - це компонент, який зберігає дані, необхідні для роботи веб-додатку. Зазвичай використовуються реляційні бази даних, такі як MySQL або PostgreSQL, але також можуть використовуватись NoSQL-бази даних, такі як MongoDB або Redis. Бази даних використовуються для зберігання, організації та отримання доступу до інформації, що використовується в веб-додатку.
- База даних: Це компонент, який зберігає дані, необхідні для роботи веб-додатку. Зазвичай використовуються реляційні бази даних, такі як MySQL або PostgreSQL, але також можуть використовуватись

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

NoSQL-бази даних, такі як MongoDB або Redis. Бази даних використовуються для зберігання, організації та отримання доступу до інформації, що використовується в веб-додатку.

Загальний веб-стек може варіюватись залежно від конкретних потреб та вимог проекту, а також від вподобань розробників та доступних ресурсів.

Ось декілька найпопулярніших веб-стеків, які широко використовуються у сучасній веб-розробці:

- LAMP стоїть за Linux (операційна система), Apache (веб-сервер), MySQL (база даних) і PHP (мова програмування). Цей стек є одним з найпоширеніших і використовується для розробки динамічних веб-сайтів і додатків.
- MEAN складається з MongoDB (NoSQL-база даних), Express.js (фреймворк для розробки серверної частини), Angular (фреймворк для розробки клієнтської частини) і Node.js (виконавче середовище для серверного JavaScript). Цей стек дозволяє розробляти повноцінні JavaScript-додатки від клієнтської до серверної частини.
- Ruby on Rails використовує Ruby (мова програмування) і Ruby on Rails (фреймворк). Ruby on Rails відомий своєю простотою, швидкістю розробки і конвенціями над конфігурацією. Він часто використовується для створення веб-додатків з акцентом на продуктивність розробки.
- MERN включає MongoDB, Express.js, React (бібліотека для розробки інтерфейсу користувача) і Node.js. Цей стек також побудований на JavaScript і дозволяє швидко розробку односторінкових додатків з динамічним інтерфейсом.
- Django - це високорівневий фреймворк на Python для розробки веб-додатків. Він забезпечує швидкість, ефективну роботу з базами

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

даних і вбудовану адміністративну панель. Django дозволяє розробляти складні веб-додатки швидко і ефективно.

Це лише кілька з найпопулярніших веб-стеків, існує ще багато інших варіантів та комбінацій технологій. Вибір конкретного веб-стеку залежить від ваших потреб, вимог проекту, рівня володіння мовами програмування та вподобань розробника.

Для розробки даної дипломної роботи було обрано стек PERN. PERN складається з наступних компонентів:

1. PostgreSQL як реляційна база даних, PostgreSQL використовується для зберігання структурованих даних. Вона є потужним і надійним рішенням з великим набором функцій, яке забезпечує можливості для розширення і масштабування додатків.
2. Express.js - це легкий і гнучкий фреймворк для розробки серверної частини додатків на Node.js. Express.js дозволяє швидко створювати API, обробляти маршрутизацію, робити запити до бази даних і багато іншого.
3. React - це потужна JavaScript-бібліотека для розробки користувацького інтерфейсу. За допомогою React можна створювати ефективні і перевикористовувані компоненти, які роблять взаємодію з користувачем більш динамічною та ефективною.
4. Виконавче середовище Node.js дозволяє виконувати JavaScript на серверній стороні. Його неблокуюча архітектура дозволяє ефективно обробляти багато запитів одночасно, що робить його ідеальним для створення масштабованих веб-додатків.

Вибір стеку PERN для розробки дипломної роботи має свої переваги. PostgreSQL є потужною базою даних з широким набором функцій і підтримкою ACID-властивостей. Express.js надає зручні інструменти для розробки серверної частини з багатofункціональним API. React дозволяє розробляти інтерактивний

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

та ефективний користувацький інтерфейс, а Node.js забезпечує швидку та ефективну обробку запитів на сервері. Цей стек є популярним серед розробників, оскільки він забезпечує швидкість розробки, масштабованість та гнучкість у створенні веб-додатків.

2.2 Архітектура високодоступного веб-стеку

Високодоступна архітектура дозволяє забезпечити веб-додатку високу ступінь доступності та надійності. Це особливо важливо для критичних застосунків, де недоступність може призвести до втрати доходу, клієнтів або довіри користувачів.

Зростання обсягу трафіку та користувачів вимагає масштабованості веб-додатка. Високодоступна архітектура дозволяє ефективно масштабувати і розподіляти навантаження між різними серверами, щоб забезпечити швидку відповідь і забезпечити стабільну роботу навіть при великому навантаженні.

Високодоступна архітектура сприяє досягненню високої продуктивності веб-додатка. Вона дозволяє оптимізувати роботу системи, використовувати кешування, мінімізувати затримки та впливати на швидкість завантаження сторінок.

Високодоступна архітектура передбачає наявність механізмів автоматичного відновлення та резервного копіювання, які дозволяють системі швидко відновлюватись після випадку неполадок або збоїв.

Високодоступна архітектура дозволяє веб-додатку збільшувати свою масштабованість з плином часу і зростанням потреб бізнесу. Вона дозволяє додавати нові компоненти, розширювати інфраструктуру та забезпечує гнучкість для змін в майбутньому.

Загалом, високодоступна архітектура допомагає забезпечити стійкість, продуктивність та надійність веб-додатків, що є критичними аспектами успішної роботи та задоволення потреб користувачів.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

Основні шари веб-додатка включають клієнтський шар, серверний шар і шар бази даних. Детальніший опис кожного з них:

- Клієнтський шар відповідає за взаємодію з користувачем і представлення інформації на стороні клієнта. Це може бути веб-браузер, мобільний додаток або інші інтерфейси. Він забезпечує відображення веб-сторінок, збору даних від користувача, валідацію введених даних та передачу їх на серверний шар для обробки.
- Серверний шар відповідає за бізнес-логіку та обробку запитів веб-додатка. Він отримує запити від клієнтського шару, виконує необхідні обчислення, взаємодіє з базою даних та генерує відповіді, які потім передаються назад до клієнтського шару.
- Шар бази даних відповідає за збереження та управління даними, необхідними для веб-додатка. Він забезпечує доступ до бази даних, виконує запити, зберігає дані та забезпечує їх цілісність та безпеку.

Взаємодія між цими шарами відбувається через стандартизовані протоколи та механізми комунікації, такі як HTTP. Клієнтський шар надсилає запити до серверного шару, який обробляє їх і взаємодіє з шаром бази даних для отримання або збереження необхідної інформації. Результати обробки повертаються назад до клієнтського шару, де вони відображаються для користувача.

Використання шарової архітектури дозволяє розділити веб-додаток на логічно зв'язані компоненти, спрощує розробку, підтримку і розширення системи. Кожен шар відповідає за свої функції і може бути незалежно модифікований без впливу на інші шари, що полегшує розподілену роботу над проектом.

Балансування навантаження є важливою технікою, яка дозволяє розподілити трафік між багатьма серверами, що забезпечує ефективне використання ресурсів, підвищення доступності та швидкодії веб-додатка. Основна ідея полягає в тому, щоб розподілити завантаження між серверами,

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

зменшуючи навантаження на кожен окремий сервер і забезпечуючи більшу стійкість та пропускну здатність системи.

В рамках даної кваліфікаційної роботи буде встановлено load balancer в середовищі GCP, який буде направляти трафік на Ingress. Ingress - це контролер, який дозволяє управляти вхідним трафіком в кластер Kubernetes. Він працює на рівні L7 і дозволяє керувати маршрутизацією та балансуванням трафіку до різних служб в Kubernetes-кластері.

Load balancer в GCP буде налаштований таким чином, щоб направляти вхідний трафік на Ingress контролер. Це дозволить забезпечити розподіл навантаження між різними вузлами кластера Kubernetes.

Інтеграція між load balancer і Ingress дозволить ефективно розподіляти вхідний трафік між різними репліками додатку, забезпечуючи високу доступність і швидкодію. Крім того, це дозволить забезпечити гнучкість в управлінні трафіком та виконання різних політик маршрутизації на рівні L7. Наприклад, ви можете налаштувати маршрутизацію на основі шляху URL або заголовків HTTP.

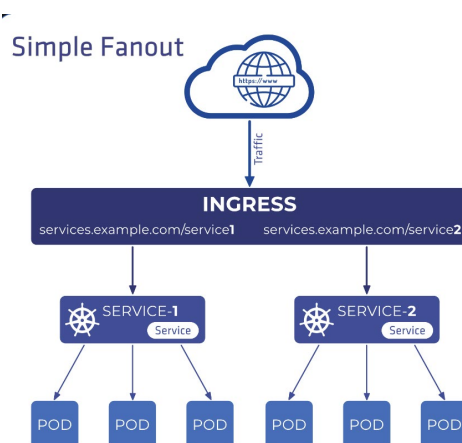


Рисунок 2.1 – Ingress об’єкт

Масштабування є важливим процесом для забезпечення пропускну здатності та високої доступності веб-додатків. Він полягає в збільшенні або зменшенні кількості ресурсів, що використовуються додатком, для відповіді на змінні навантаження.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

Існують два основних підходи до масштабування: вертикальне та горизонтальне.

Вертикальне масштабування передбачає збільшення потужності одного сервера шляхом додавання ресурсів, таких як процесор, оперативна пам'ять або дисковий простір. Наприклад, можна оновлювати сервер, замінивши його процесор на більш швидкий або додавши більше оперативної пам'яті. Вертикальне масштабування підходить для ситуацій, коли потрібно забезпечити більшу потужність одному серверу, але він має фізичні обмеження.

Горизонтальне масштабування передбачає, що додаток розгортається на кількох серверах, які працюють паралельно та розділяють навантаження між собою. При зростанні навантаження додаються нові сервери, а при зниженні - можуть бути вимкнуті. Горизонтальне масштабування забезпечує більшу масштабованість, оскільки дозволяє розподіляти навантаження на більшу кількість серверів, а також забезпечує вищу доступність, оскільки додаток продовжує працювати, навіть якщо один сервер вийде з ладу. Для горизонтального масштабування потрібні механізми для автоматичного розподілу навантаження, такі як load balancer.

Контейнеризація та оркестрація, зокрема за допомогою Kubernetes, дозволяють зручно масштабувати додатки. Kubernetes є одним з популярних інструментів для оркестрації контейнерів. Він дозволяє автоматизувати розгортання, масштабування та управління контейнеризованими додатками. Kubernetes динамічно керує розподілом контейнерів на вузлах, автоматично масштабує додатки відповідно до навантаження та забезпечує високу доступність. Контейнери дозволяють упаковувати додатки та їх залежності відокремлено, створюючи портативні та ізольовані середовища. Це дозволяє легко розгорнути та масштабувати додатки без впливу на решту системи.

Горизонтальне масштабування в Kubernetes передбачає збільшення або зменшення кількості реплік контейнерів, що працюють на вузлах кластера. При

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

зростанні навантаження, Kubernetes автоматично додає нові репліки контейнерів, розподіляючи навантаження між ними. За необхідності, коли навантаження зменшується, Kubernetes може автоматично зменшувати кількість реплік. Горизонтальне масштабування дозволяє розподіляти навантаження на більшу кількість контейнерів і забезпечує високу доступність додатку.

Вертикальне масштабування в Kubernetes полягає в зміні розміру ресурсів, призначених для окремого контейнера. Наприклад, можна збільшити або зменшити обсяг CPU або оперативної пам'яті, виділеної для контейнера. Вертикальне масштабування дозволяє забезпечити більшу потужність одному контейнеру, що може бути корисним у випадку, коли додаток вимагає більше ресурсів для оптимальної роботи.

Моніторинг та логування відіграють надзвичайно важливу роль у високодоступному веб-стеці, оскільки вони забезпечують відстеження метрик продуктивності, виявлення проблем і ефективне їх вирішення. Для цих цілей можна використовувати різні інструменти, такі як Prometheus, Grafana або DataDog. Вони дозволяють збирати, візуалізувати і аналізувати метрики, такі як завантаження сервера, використання ресурсів (CPU, пам'ять), швидкість відповіді та інші параметри продуктивності. Моніторинг метрик допомагає виявити проблеми, знайти "пробої" і зрозуміти, яке навантаження в системі.

Логування є необхідним для відстеження дії веб-стеку і виявлення проблем. Інструменти, такі як ELK Stack (Elasticsearch, Logstash, Kibana) або Splunk, забезпечують збір, аналіз та візуалізацію логів з різних компонентів системи. Це дозволяє виявляти помилки, аналізувати події, відслідковувати діагностичні повідомлення і шукати причини проблем.

Однією з важливих функцій інструментів моніторингу є можливість автоматичного сповіщення, що дозволяє оперативно реагувати на проблеми. Наприклад, вони можуть надсилати сповіщення через електронну пошту, служби

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

сповіщень або месенджери, коли виявляються надмірні навантаження, помилки або відхилення від норми метрик продуктивності.

Використання моніторингу та логування дозволяє оперативно виявляти проблеми, ефективно вирішувати помилки і забезпечувати високу доступність та продуктивність веб-стеку. Ці інструменти допомагають покращити ефективність розробки, вирішувати проблеми і задовольняти потреби користувачів.

2.3 Огляд та вибір інструменту для автоматизації CI/CD процесу

Безперервна інтеграція - це практика розробки програмного забезпечення, яка полягає в постійному інтегруванні коду з різних розробників в спільний репозиторій, а також в автоматичному перевірці цього коду на наявність помилок і конфліктів.

Головна мета безперервної інтеграції - забезпечити регулярну та автоматизовану перевірку інтеграції змін в коді, щоб виявляти помилки та конфлікти якомога раніше. Замість того, щоб очікувати до кінця розробки для об'єднання коду великої кількості розробників, безперервна інтеграція передбачає невеликі та часті інтеграції.

Після злиття змін в репозиторій, спеціальний інструмент або система автоматично запускає процес інтеграції. У цьому процесі код перевіряється на наявність помилок, конфліктів та відхилень від визначених стандартів. Якщо виявляються проблеми, розробник отримує повідомлення з описом проблеми, що дозволяє йому швидко виправити її. Це дозволяє виявити та вирішити проблеми на ранніх етапах розробки, перед тим як вони стануть більш серйозними та складними у виправленні.

Безперервна інтеграція має на меті підвищити якість програмного забезпечення, забезпечити стабільну основу коду та зменшити ризики при інтеграції змін. Вона сприяє покращенню співпраці між розробниками,

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

прискорює процес розробки та сприяє швидкому реагуванню на виявлені проблеми.

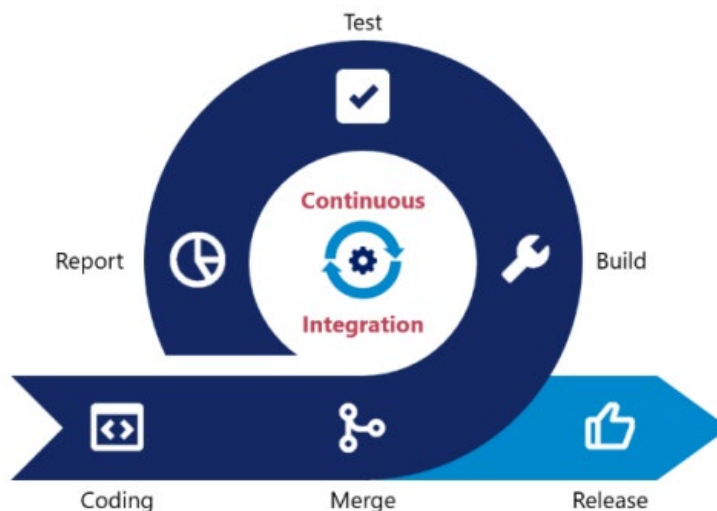


Рисунок 2.2 – Ілюстрація циклу безперервної інтеграції

Безперервна доставка (Continuous Delivery) та безперервне розгортання (Continuous Deployment) є двома практиками в галузі розробки програмного забезпечення, які спрямовані на автоматизацію та прискорення процесу випуску програмного продукту.

Безперервна доставка означає, що розроблене програмне забезпечення готове до випуску в будь-який момент. Це передбачає автоматизовану збірку, тестування та пакування програмного коду, щоб забезпечити стабільність та якість продукту. За допомогою безперервної доставки, розробники можуть швидко та надійно постачати нові функції, зміни та виправлення[18].

Безперервне розгортання піднімає практику безперервної доставки на ще вищий рівень, дозволяючи автоматично розгортати програмне забезпечення в реальних умовах виробництва після проходження всіх тестів і перевірок. Це означає, що зміни, які пройшли процес безперервної доставки, автоматично встановлюються на виробниче середовище без необхідності ручного втручання. Це дозволяє швидко та безпечно впроваджувати нові функції та виправлення у продукт.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

Обидві практики, безперервна доставка та безперервне розгортання, спрямовані на скорочення часу від розробки нового функціоналу до його введення в експлуатацію, покращення якості продукту та забезпечення швидкого зворотного зв'язку між розробниками та користувачами. Вони вимагають використання автоматизованих інструментів та процесів для забезпечення надійності та стабільності розгортання програмного забезпечення.

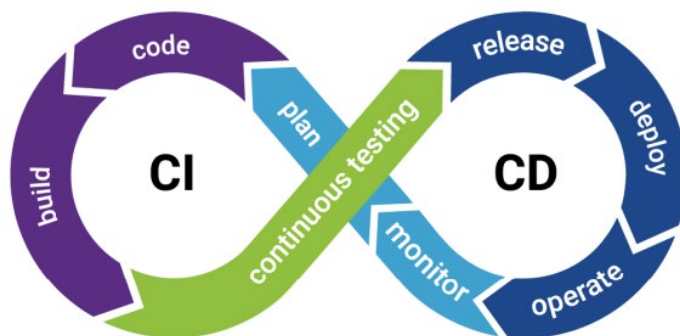


Рисунок 2.3 – Ілюстрація циклу CI/CD

Основні принципи безперервної інтеграції та безперервної доставки (CI/CD) включають наступні елементи:

- Автоматизація процесів розробки та доставки програмного забезпечення. Це охоплює автоматичну збірку, тестування, пакування та розгортання програмного коду. Автоматизовані процеси дозволяють ефективно та надійно виконувати рутинні завдання та скорочувати час до впровадження змін у продукт.
- Розробники регулярно зливають свої зміни у спільний репозиторій. Це сприяє швидкому виявленню можливих конфліктів і помилок, а також підвищує якість коду. Рання інтеграція допомагає зменшити ризики та забезпечити стабільність проекту.
- Частиною CI/CD є запуск автоматизованих тестів під час процесу інтеграції та доставки. Це включає модульні тести, інтеграційні тести, функціональні тести та інші форми перевірок, щоб переконатися, що зміни не порушують функціональність системи. Автоматичні тести

допомагають виявляти помилки на ранніх етапах та підтримувати високу якість коду.

- CI/CD сприяє частим релізам програмного забезпечення. Завдяки автоматизованим процесам та низькому ризику впровадження змін, розробники можуть випускати нові версії швидше та ефективніше.
- CI/CD забезпечує швидкий зворотний зв'язок між розробниками та користувачами. Завдяки швидкому циклу розробки та доставки, можна отримувати зворотній зв'язок щодо нових функцій, виправлень та змін у продукті. Це допомагає розробникам адаптуватися до потреб користувачів та надавати продукт, який відповідає їх очікуванням.

В розробці програмного забезпечення застосовуються різні інструменти для автоматизації тестування і доставки коду проектів кінцевим користувачам:

1. CircleCI

CircleCI - це хмарний сервіс CI/CD, який надає розробникам широкі можливості для автоматизації процесу розробки та доставки програмного забезпечення. Основні можливості CircleCI включають:

- Легка налаштування та інтеграція з репозиторіями Git.
- Автоматична збірка, тестування та розгортання змін.
- Можливість паралельної обробки для прискорення процесу CI/CD.
- Інтеграція з іншими інструментами та сервісами, такими як GitHub, Bitbucket, Slack та інші.

2. Azure Pipelines

Azure Pipelines - це інструмент CI/CD, який є частиною платформи Azure DevOps від Microsoft. Він надає широкий набір функціональних можливостей для автоматизації процесу розробки та доставки. Основні можливості Azure Pipelines включають:

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

- Інтеграція з різними сервісами Azure, такими як Azure Repos, Azure Boards, Azure Artifacts та інші.
- Широкі можливості налаштування та складних сценаріїв CI/CD.
- Використання власних агентів або хмарних агентів Azure для виконання задач CI/CD.

3. GitLab CI

GitLab CI - це вбудований інструмент CI/CD, який надається разом з платформою GitLab. Він забезпечує розробникам широкі можливості для автоматизації процесу збірки, тестування та розгортання програмного забезпечення. Основні можливості GitLab CI включають:

- Проста конфігурація через файл `.gitlab-ci.yml`.
- Інтеграція з GitLab, що дозволяє автоматично запускати процеси CI/CD при злитті змін або створенні пул-реквесту.
- Підтримка контейнерів для створення ініційного середовища для виконання завдань.
- Використання власних агентів або хмарних агентів для виконання задач CI/CD

4. Jenkins

Jenkins - це один з найпопулярніших відкритих інструментів CI/CD. Він надає широкі можливості для автоматизації процесу розробки та доставки програмного забезпечення. Основні можливості Jenkins включають:

- Широкий вибір плагінів та розширень для розширення функціональності.
- Розподілені збірки та обробка паралельних завдань.
- Гнучкість у налаштуванні та конфігурації процесу CI/CD.

Ці інструменти CI/CD надають розробникам потужні інструменти для автоматизації розробки та доставки програмного забезпечення. Кожен з них має

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

свої особливості та можливості, які можуть відповідати певним вимогам проекту. У даній кваліфікаційній роботі буде використовуватися GitLab CI. GitLab CI є частиною інтегрованої платформи GitLab і надає широкий набір можливостей для автоматизації процесу CI/CD, включаючи збірку, тестування та розгортання програмного забезпечення. Він має зручний конфігураційний файл, що дозволяє визначати кроки та правила для автоматичного виконання певних дій при зміні коду.

Висновки до розділу 2

В другому розділі було проведено проектування високодоступного веб-стеку. Вибір технологій та інструментів є важливим кроком у побудові стеку, і було розглянуто різні аспекти, такі як мови програмування, фреймворки, бази даних та інші компоненти. Далі, була розроблена архітектура високодоступного веб-стеку, включаючи розподілення навантаження, балансування навантаження, масштабування та резервне копіювання. Крім того, було проведено огляд інструментів для автоматизації CI/CD процесу, зокрема вибіру та опису основних можливостей інструменту GitLab CI. Цей розділ надав основні принципи та стратегії для побудови високодоступного веб-стеку та вибору відповідних інструментів для автоматизації розробки та доставки програмного забезпечення.

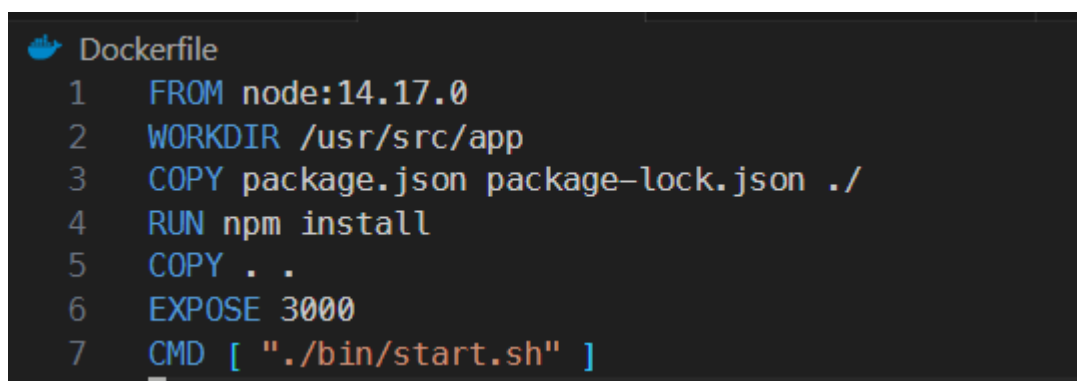
		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

РОЗДІЛ 3 НАЛАШТУВАННЯ CI/CD ДЛЯ РОЗГОРТАННЯ ПРОЕКТУ В KUBERNETES КЛАСТЕРІ ЗА ДОПОМОГОЮ HELM

3.1 Створення Docker-файлів

Створення Docker-файлів є важливим кроком у процесі розробки контейнеризованих додатків. Docker-файл є текстовим файлом, який містить інструкції для побудови образу Docker. Цей образ включає всі необхідні компоненти, бібліотеки та залежності для виконання вашого додатку в ізольованому середовищі.

Docker-файл для backend додатку має наступний вигляд:



```
Dockerfile
1 FROM node:14.17.0
2 WORKDIR /usr/src/app
3 COPY package.json package-lock.json ./
4 RUN npm install
5 COPY . .
6 EXPOSE 3000
7 CMD [ "./bin/start.sh" ]
```

Рисунок 3.1 – Docker-файл backend додатка

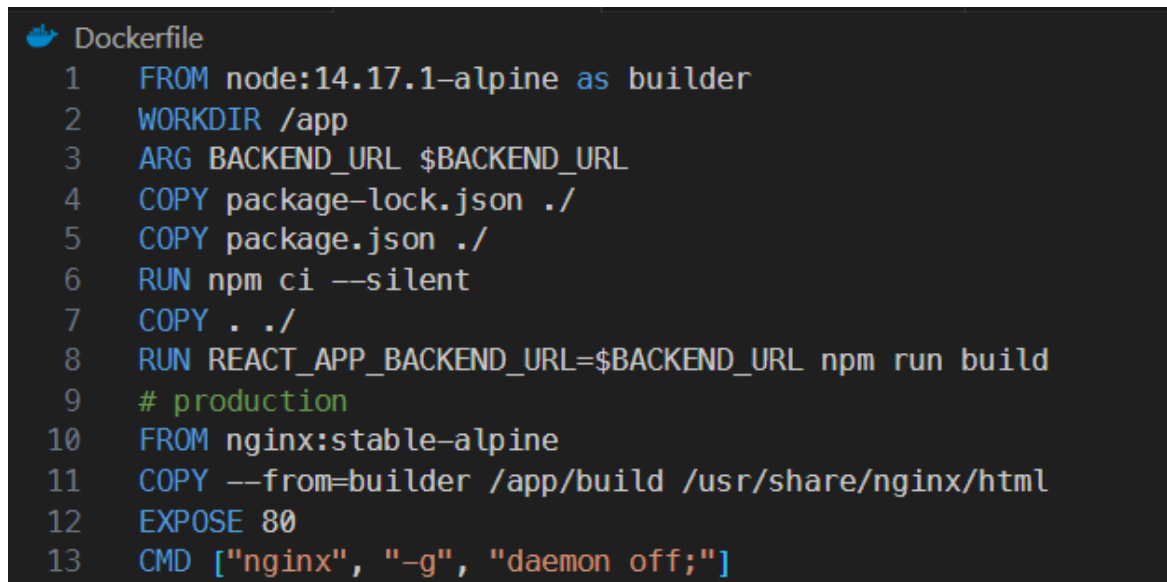
Для побудови Docker-файлу для frontend додатку буде використовуватися multistage збірка. Коли розмова йде про Docker-образи, то "більше" означає "гірше". Великі образи Docker - це повільні образи. Вони ускладнюють роботу і можуть мати більше потенційних вразливостей, що може відкрити шлях до атаки. Головна мета - будувати образи лише з необхідним для запуску вашого додатку. І саме тут на допомогу приходять багатоетапні збірки. Вони дозволяють оптимізувати процес збірки без ускладнень.

Основна ідея багатоетапних збірок полягає в тому, що Docker-файл містить кілька інструкцій FROM. Кожна така інструкція визначає новий етап збірки, який може легко копіювати артефакти з попередніх етапів. Такий підхід дозволяє

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

створити компактний та оптимізований Docker-образ, який містить лише необхідні компоненти для роботи вашого додатку. Це зменшує розмір образу, сприяє швидкому розгортанню та покращує безпеку.

Docker-файл для frontend додатку має наступний вигляд:



```

1 FROM node:14.17.1-alpine as builder
2 WORKDIR /app
3 ARG BACKEND_URL $BACKEND_URL
4 COPY package-lock.json ./
5 COPY package.json ./
6 RUN npm ci --silent
7 COPY . ./
8 RUN REACT_APP_BACKEND_URL=$BACKEND_URL npm run build
9 # production
10 FROM nginx:stable-alpine
11 COPY --from=builder /app/build /usr/share/nginx/html
12 EXPOSE 80
13 CMD ["nginx", "-g", "daemon off;"]

```

Рисунок 3.2 – Docker-файл frontend додатка

Багатоетапні збірки - це потужний інструмент, який допомагає забезпечити ефективну та безпечну роботу з Docker-образами.

3.2 Використання Helm для керування розгортанням та управлінням релізами

Helm - це інструмент управління пакетами для Kubernetes, який дозволяє легко керувати розгортанням додатків та управляти релізами. Використання Helm спрощує процес розгортання, оновлення та керування додатками у Kubernetes-кластері.

Основна концепція Helm базується на використанні "чартів". Чарт - це пакет, який містить всі необхідні компоненти для розгортання додатку у Kubernetes. Він включає в себе опис ресурсів Kubernetes, які необхідно створити (наприклад, поди, служби, конфігураційні файли тощо), а також значення параметрів, які можуть бути налаштовані при розгортанні.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Використання Helm дозволяє зручно встановлювати, оновлювати та видаляти чарти за допомогою відповідних команд. Наприклад, команда *helm install* встановлює новий реліз чарту, *helm upgrade* оновлює існуючий реліз до нової версії, а *helm uninstall* видаляє реліз. Ці команди автоматично керують процесом створення, конфігурації та встановлення ресурсів Kubernetes, що входять до чарту.

Крім того, Helm надає можливість керувати значеннями параметрів чарту через файл *values.yaml* або передавати їх безпосередньо при виклику команд. Це дозволяє налаштувати параметри розгортання, такі як розмір ресурсів, налаштування бази даних, URL-адреси служб тощо.

З використанням Helm також можна керувати версіями релізів. Кожне розгортання додатку вважається релізом, і Helm зберігає історію релізів, що дозволяє легко відновлювати попередні версії або відкочувати до попереднього стану.

Встановлення helm на Linux ОС:

```
(mystic-producer-387705)$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
(mystic-producer-387705)$ chmod 700 get_helm.sh
(mystic-producer-387705)$ ./get_helm.sh
```

Рисунок 3.3 – Встановлення Helm

Іншим важливим аспектом організації Kubernetes-проектів є винесення конфігураційних файлів опису Kubernetes у окремий репозиторій. Це забезпечує чітку роздільність між джерелами коду та конфігурацією і дозволяє краще керувати конфігурацією додатків у різних середовищах, таких як розробка, тестування та продакшн.

Для конфігураційного репозиторію було написано власний CI/CD конвеєр, який запускає linter, пакує чарт в архів та завантажує в Gitlab репозиторій:

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

```

stages:
  - lint
  - publish
  - upload

default:
  image:
    name: "alpine/helm:3.6.3"
    entrypoint: ["/bin/sh", "-c"]

lint:
  stage: lint
  script:
    - helm lint charts/app

package:
  stage: publish
  script:
    - helm package charts/app -d pkgs/
  artifacts:
    paths:
      - pkgs/*.tgz
    expire_in: 1 week

upload:
  stage: upload
  image: curlimages/curl:latest
  script:
    - >
      for d in pkgs/*.tgz; do
        curl --request POST --user gitlab-ci-token:$CI_JOB_TOKEN --form "chart=@$d" "${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/packages/helm/api/
      done

```

У підсумку, використання Helm для керування розгортанням та управлінням релізами дозволяє спростити процес розгортання додатків у Kubernetes, забезпечуючи повторюваність, керованість та швидкість управління. Це розширює можливості розгортання додатків у Kubernetes-середовищі та сприяє підвищенню продуктивності розробників та адміністраторів систем.

Після того як код буде завантажено на віддалений репозиторій, запуститься pipeline та завантажить нову версію чарту в реєстр пакетів Gitlab.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

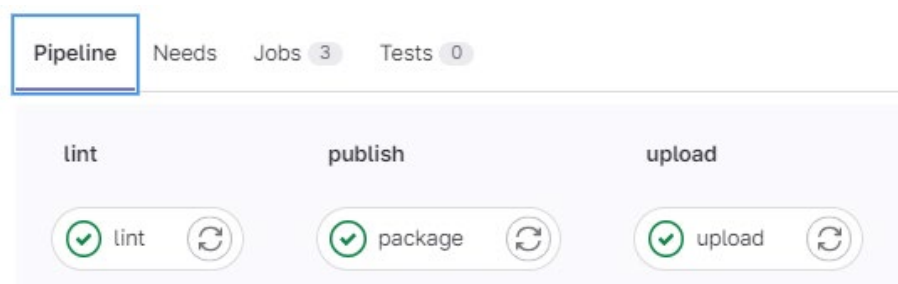


Рисунок 3.4 – Запуск pipeline для конфігураційного репозиторія

Завантажений helm чарт:



Рисунок 3.5 – Публікація helm чарта в реєстр пакетів на Gitlab

3.3 Налаштування CI/CD пайплайну для розгортання веб-стеку в Kubernetes кластері

Налаштування CI/CD пайплайну для розгортання веб-стеку в Kubernetes кластері може включати в себе практику використання Review app. Review app є важливою частиною цього процесу, оскільки дозволяє перевірити і оцінити зміни коду перед їхнім впровадженням в основне середовище Kubernetes.

Review app - це термін, що використовується для опису практики створення тимчасових ізольованих середовищ для перегляду та перевірки коду перед об'єднанням його з основною гілкою репозиторію або внесенням змін до виробничого середовища. Review app - це окремий екземпляр програми або середовище, яке розгортається автоматично на основі останнього стану коду з гілки або під час merge request, який відкрився для перегляду. Це дозволяє розробникам, тестувальникам та іншим зацікавленим сторонам перевірити та протестувати зміни в реальному середовищі перед їхнім впровадженням в основне середовище.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

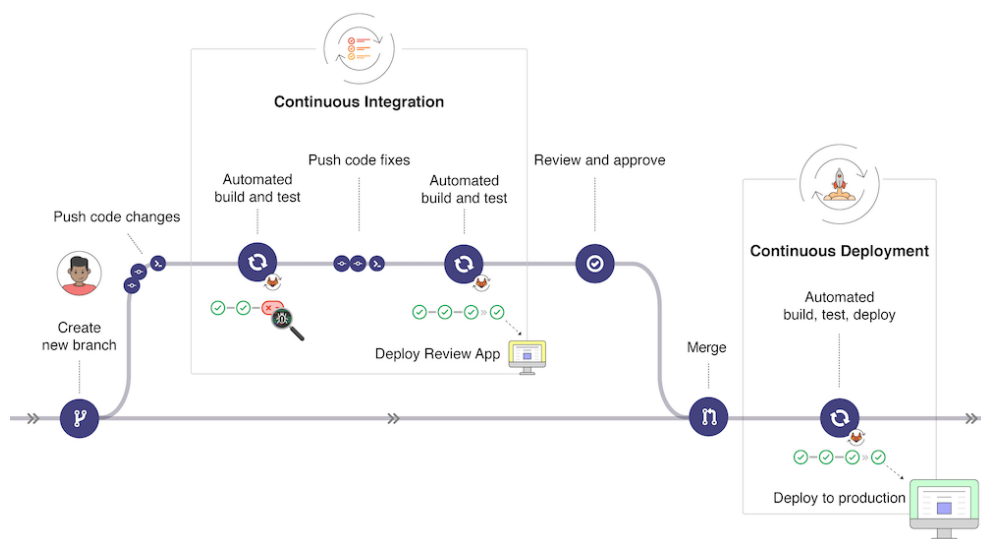


Рисунок 3.6 – Ілюстрація Review app

Для підготовки розгортання веб-стеку в Kubernetes кластері, такого як GCP, спочатку необхідно мати наявність самого кластера Kubernetes. Для прикладу, було створено Kubernetes кластер у сервісі керування контейнерами GKE на платформі GCP. GKE надає легкий спосіб розгортання та керування кластером Kubernetes.

Kubernetes clusters

+ CREATE

+ DEPLOY

REFRESH

OPERATIONS

HELP ASSISTANT

OVERVIEW

OBSERVABILITY

COST OPTIMIZATION

Filter

Enter property name or value

Sort

☐ Status	Name ↑	Location	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
☐	cluster-for-ci-cd	us-central1-c					—

Рисунок 3.7 – Створений Kubernetes кластер

Після створення кластера Kubernetes в GCP, наступним кроком є підключення його до GitLab CI для автоматизації процесу розгортання та управління додатками. Для цього необхідно створити файл у конфігураційному репозиторії GitLab, який вказує, яким групам дозволено отримувати доступ до цього репозиторію.

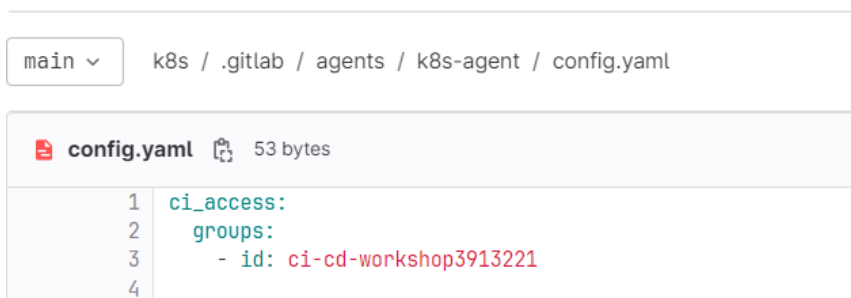


Рисунок 3.8 – Конфігураційний файл

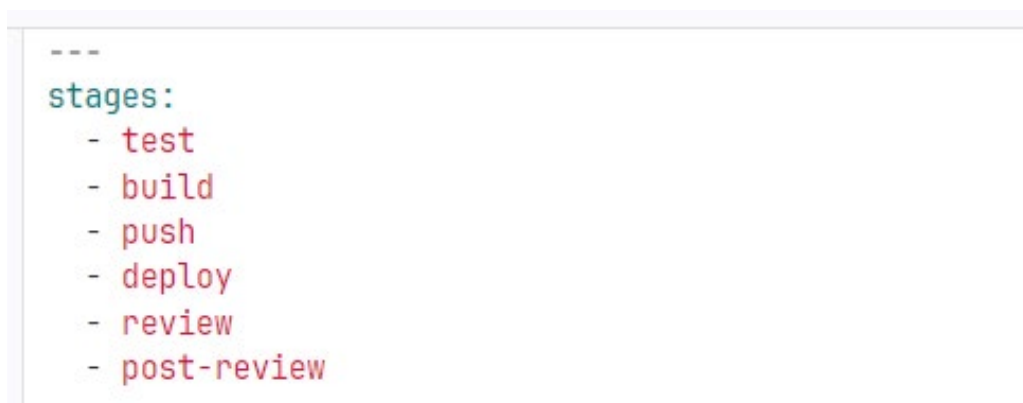
Після створення кластера Kubernetes в GCP, наступним кроком є встановлення GitLab агента в репозиторій за допомогою Helm. Цей агент дозволить забезпечити інтеграцію з GitLab та забезпечити автоматизовану збірку та розгортання.

Після успішного встановлення GitLab агента з використанням Helm, з'явиться повідомлення про успішне підключення. Це підтвердить, що агент успішно встановлено і налаштовано, готовий до виконання завдань CI/CD у репозиторії.

Name	Connection status	Last contact	Version	Agent ID	Configuration
k8s-agent	Connected	1 minute ago	16.0.1	60825	.gitlab/agents/k8s-agent

Рисунок 3.9 – Підключення Kubernetes кластера

Після успішного базового налаштування Kubernetes кластера потрібно налаштувати CI/CD конвеєра для backend додатку. Для початку потрібно розбити конвеєр на відповідні етапи:



Наступним кроком буде задати глобальні змінні для всього конвеєра:

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

```
variables:
  REVIEW_BASE_DOMAIN: "$APP_HOST.nip.io"
  DOMAIN: "$APP_HOST.nip.io"
  LIVE_DOMAIN: "production-$APP_HOST.nip.io"
  KUBE_NAMESPACE: ci-cd-workshop
```

Далі потрібно створити сховані(hidden) job від яких будуть наслідуватися інші job.

```
.node:
  image: node:14.17.0-alpine
  cache:
    key:
      files:
        - package-lock.json
      paths:
        - .npm/
  before_script:
    - npm ci --cache .npm --prefer-offline

.docker:
  image: docker:20
  services:
    - docker:dind
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY

.docker_auth:
  before_script:
    - docker login -u $CI_DEPLOY_USER -p $CI_DEPLOY_PASSWORD $CI_REGISTRY
```

.node - використовує образ node:14.17.0-alpine, який містить Node.js версії 14.17.0 на базі Alpine Linux. Кешування використовується для прискорення процесу встановлення залежностей у проєкті. Ключ кешування визначається за допомогою package-lock.json, а кешовані файли зберігаються у .npm/. Передвиконавчий скрипт встановлює залежності проєкту за допомогою команди npm ci.

.docker використовується для виконання операцій, пов'язаних з Docker. Використовуються служба docker:dind для доступу до сервісів Docker Engine всередині контейнера. Секція before_script аутентифікує GitLab CI з Docker Registry для здійснення публікації образів.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

`.docker_auth` використовується лише для аутентифікації GitLab CI з Docker Registry.

Наступник кроком опишемо завдання `lint`, для перевірки стилю та форматування коду. Завдання буде запускатися за певних умов, визначених у секції `rules`:

```
lint:
  extends:
    - .node
  stage: test
  script:
    - npm install
    - npm run lint
    - npm run check:format
  rules:
    - if: $CI_COMMIT_BRANCH == 'main'
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_PIPELINE_SOURCE == "pipeline"
```

`build` - завдання, яке відповідає за побудову Docker образу. Використовує образ Docker заснований на `.docker`. У скрипті виконуються наступні дії: витягування останньої версії образу з реєстру, побудова нового образу з використанням кешування, надання тегів нового образу за допомогою значень змінних середовища, та завантаження нового образу до реєстру.

```
build:
  needs: ["lint"]
  stage: build
  extends:
    - .docker
  script:
    - docker pull $CI_REGISTRY_IMAGE:latest || true
    - >
      docker build
      --pull
      --cache-from $CI_REGISTRY_IMAGE:latest
      --tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
      --tag $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
    - .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
  rules:
    - if: $CI_COMMIT_BRANCH == 'main'
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_PIPELINE_SOURCE == "pipeline"
```

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

push_latest - job, який відповідає за публікацію оновленої версії образу Docker з тегом "latest" в реєстр.

```
push_latest:
  stage: push
  extends:
    - .docker
  variables:
    GIT_STRATEGY: none
  script:
    - docker pull $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
    - docker tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA $CI_REGISTRY_IMAGE:latest
    - docker push $CI_REGISTRY_IMAGE:latest
  only:
    - main
```

.deploy - job, який відповідає за розгортання додатку на Kubernetes. Цей job належить до стадії "deploy".

```
deploy:
  stage: deploy
  image:
    name: dtzar/helm-kubectl:3.5.3
    entrypoint: ['']
  variables:
    GIT_STRATEGY: none
  script:
    - kubectl config use-context ci-cd-workshop3913221/k8s:k8s-agent
    - |
      helm repo add --username gitlab-ci-token \
        --password $CI_JOB_TOKEN \
        ci-cd-workshop \
        ${CI_API_V4_URL}/projects/46276369/packages/helm/stable
    - helm repo list
    - helm repo update
    - |
      helm upgrade \
        --install \
        --set node.image.tag="$CI_COMMIT_REF_NAME" \
        --set web.image.tag="$CI_COMMIT_REF_NAME" \
        --set ciVars.domain="$DOMAIN" \
        --set ciVars.CI_ENVIRONMENT_SLUG="$CI_ENVIRONMENT_SLUG" \
        --set ciVars.CI_PROJECT_PATH_SLUG="$CI_PROJECT_PATH_SLUG" \
        --set ciVars.CI_COMMIT_REF_SLUG="$CI_COMMIT_REF_SLUG" \
        production \
        ci-cd-workshop/app
  environment:
    name: production
    url: http://$LIVE_DOMAIN/
    kubernetes:
      namespace: "$KUBE_NAMESPACE"
  rules:
    - if: $CI_COMMIT_BRANCH == 'main'
```

На стадії "review" виконується створення нової версії додатку для кожного merge request . Це включає налаштування контексту kubectl, додавання репозиторію Helm, оновлення репозиторіїв Helm та оновлення або встановлення релізу Helm з використанням вказаних параметрів. Додатково, перевіряється

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

наявність гілки у GitLab, і якщо її немає, створюється нова гілка. У результаті розгортається окреме середовище з відповідним URL та налаштуваннями Kubernetes. Опис create_review_app job виглядає наступним чином:

```
create_review_app:
  stage: review
  needs: ["build"]
  image:
    name: dtzar/helm-kubectl:3.5.3
    entrypoint: ['']
  variables:
    BRANCH_EXISTS_URL: 'https://gitlab.com/api/v4/projects/$FRONTEND_PROJECT_ID/repository/branches/$CI_COMMIT_REF_NAME'
    CREATE_BRANCH_URL: 'https://gitlab.com/api/v4/projects/$FRONTEND_PROJECT_ID/repository/branches?branch=$CI_COMMIT_REF_NAME&ref=$CI_DEFAULT_BRANCH'
    GIT_STRATEGY: none
  script:
    - apk add --update curl
    - kubectl config use-context ci-cd-workshop3913221/k8s:k8s-agent
    - |
      helm repo add --username gitlab-ci-token \
        --password $CI_JOB_TOKEN \
        ci-cd-workshop \
        ${CI_API_V4_URL}/projects/46276369/packages/helm/stable
      helm repo update
      helm upgrade \
        --install \
        --set node.image.tag="$CI_COMMIT_REF_NAME" \
        --set web.image.tag="$CI_COMMIT_REF_NAME" \
        --set ciVars.domain="$DOMAIN" \
        --set ciVars.CI_ENVIRONMENT_SLUG="$CI_ENVIRONMENT_SLUG" \
        --set ciVars.CI_PROJECT_PATH_SLUG="$CI_PROJECT_PATH_SLUG" \
        --set ciVars.CI_COMMIT_REF_SLUG="$CI_COMMIT_REF_SLUG" \
        $CI_COMMIT_REF_NAME \
        ci-cd-workshop/app
    - 'status_code=$(curl -I --header "PRIVATE-TOKEN: $PAT_TOKEN" --write-out "%{http_code}" --silent --output /dev/null "$BRANCH_EXISTS_URL")'
    - echo $status_code
    - '[[ "$status_code" -ne 204 ]] && status_code=$(curl -X POST --header "PRIVATE-TOKEN: $PAT_TOKEN" --write-out "%{http_code}" --silent --output /dev/null "$CREATE_BRANCH_URL")'
    - echo OK

environment:
  name: review/$CI_COMMIT_REF_NAME
  url: http://$CI_COMMIT_REF_NAME-$REVIEW_BASE_DOMAIN/
  on_stop: stop_review
  kubernetes:
    namespace: "$KUBE_NAMESPACE"
  rules:
    - if: '$CI_MERGE_REQUEST_TITLE =~ /SKIP REVIEW/'
      when: never
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_PIPELINE_SOURCE == "pipeline"'
```

На наступному кроці, на стадії "post-review" відбувається побудова review app для фронтенду. Залежність від цієї задачі (create_review_app) гарантує, що review для основного додатку вже була успішно створена. Дана задача використовує тригер, щоб запустити CI/CD процес для проекту "ci-cd-workshop3913221/frontend" на гілці, яка відповідає даній.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

create_review_frontend_app:
  stage: post-review
  needs: ["create_review_app"]
  variables:
    GIT_STRATEGY: none
  trigger:
    project: "ci-cd-workshop3913221/frontend"
    branch: $CI_COMMIT_REF_NAME
    strategy: depend
  rules:
    - if: '$CI_MERGE_REQUEST_TITLE =~ /SKIP REVIEW/'
      when: never
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"

```

На стадії "stop_review" виконується зупинка review app для даної гілки. "stop_review" видаляє review app за допомогою команди *helm uninstall* з використанням імені review app.

```

stop_review:
  stage: review
  image:
    name: dtzar/helm-kubectl:3.5.3
    entrypoint: ['']
  variables:
    GIT_STRATEGY: none
  environment:
    name: review/$CI_COMMIT_REF_NAME
    action: stop
  script:
    - kubectl config use-context ci-cd-workshop3913221/k8s:k8s-agent
    - helm uninstall $CI_COMMIT_REF_NAME
  rules:
    - if: '$CI_MERGE_REQUEST_TITLE =~ /SKIP REVIEW/'
      when: never
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      when: manual

```

Pipeline для frontend додатку є схожим і має багато спільних частин з попереднім пайплайном. Тому можна винести деякі частини в окремі шаблони або розширення, які можна використовувати знову.

3.4 Перевірка працездатності

Для перевірки роботи production

```

~ at gke_mystic-producer-387705-us-central1-c_cluster-for-ci-cd ()
+ helm list
NAME          NAMESPACE    REVISION    UPDATED                               STATUS    CHART          APP VERSION
nginx-ingress default       1           2023-05-24 20:17:31.714991144 +0300 EEST    deployed  ingress-nginx-4.6.1  1.7.1
production    default       4           2023-05-29 10:50:46.47235304 +0000 UTC    deployed  app-0.2.0          0.1.1

```

Рисунок 3.10 – Результат виконання команди *helm list*

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Було перевірено налаштування Ingress. Ingress є механізмом управління зовнішнім доступом до наших додатків, і перевірка його налаштувань дозволила переконатись, що зовнішній доступ до додатків належним чином налаштований.

```
+ kubectl describe ingress production-app
Name:                production-app
Labels:              app.gitlab.com/app=ci-cd-workshop3913221-backend
                    app.gitlab.com/env=production
                    app.kubernetes.io/instance=production
                    app.kubernetes.io/managed-by=Helm
                    app.kubernetes.io/name=app
                    app.kubernetes.io/version=0.1.1
                    helm.sh/chart=app-0.2.0
Namespace:          default
Address:             34.132.8.115
Ingress Class:       <none>
Default backend:     <default>
Rules:
  Host                Path    Backends
  ----                -
  main.34-132-8-115.nip.io
                    /      web-production-app-svc:80 (10.8.0.15:80,10.8.1.17:80,10.8.2.20:80)
                    /api  node-production-app-svc:80 (10.8.0.15:3000,10.8.1.17:3000,10.8.2.20:3000)
Annotations:
  kubernetes.io/ingress.class: nginx
  meta.helm.sh/release-name: production
  meta.helm.sh/release-namespace: default
  nginx.ingress.kubernetes.io/ssl-redirect: false
Events:
  <none>
```

Рисунок 3.11 – Результат команди `kubectl describe ingress <name>`

На наступному кроці було перевірено наявність та стан розгорнутих deployment об'єктів. Deployment є ключов об'єктом в управлінні розгортанням додатків в кластері Kubernetes.

```
~ kubectl get deployments.apps
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-ingress-ingress-nginx-controller  1/1     1             1           4d21h
node-production-app                   1/1     1             1           4d4h
postgres-production-app               1/1     1             1           4d4h
web-production-app                    1/1     1             1           4d4h
```

Рисунок 3.12 – Результат команди `kubectl get deployments.apps`

Для перевірки роботи самого додатку, можна перейти за посиланням на вкладці "Environment" в GitLab. На цій вкладці можна знайти інформацію про поточні середовища, де розгорнуті наші додатки, а також їхні стани та деталі.

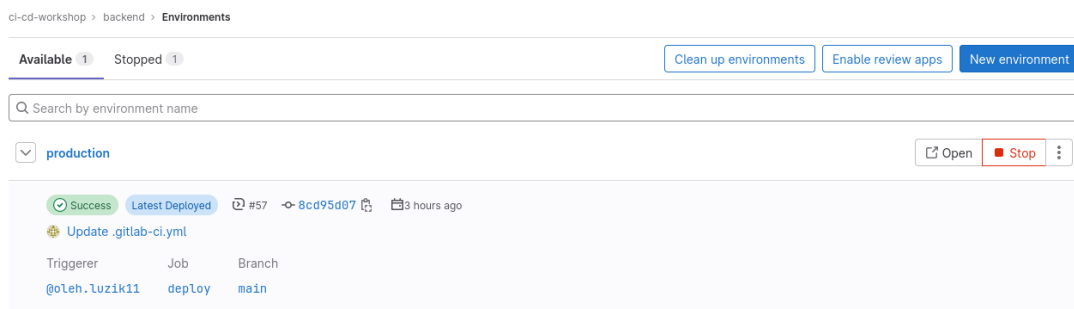


Рисунок 3.13 – Вкладка environment у Gitlab

Після переходу за посиланням, було успішно протестовано роботу веб-стеку на працездатність:

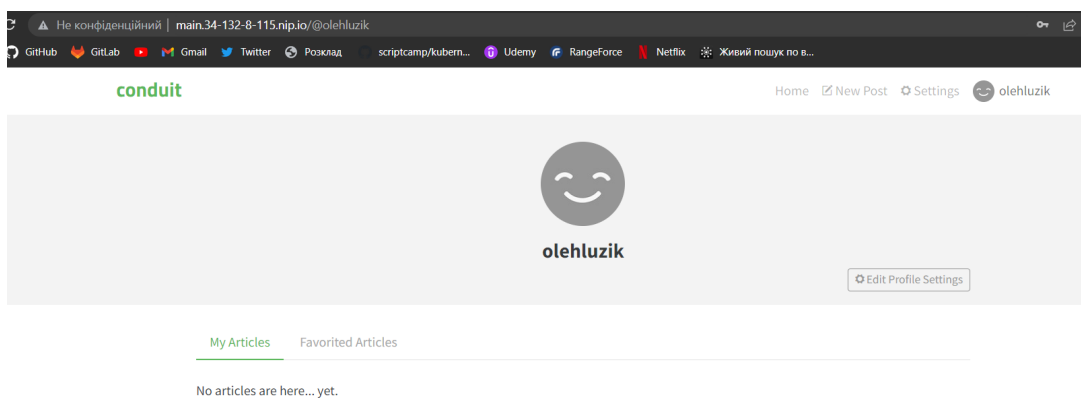


Рисунок 3.14 – Перевірка додатку

Тепер потрібно створити нову гілку та внести в неї зміни. Створити merge request до гілки main. Для цього було створено гілку testing та додано наступні зміни:



Рисунок 3.15 – Внесення змін у гілку testing

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

Після цього потрібно створити merge request у гілку main:

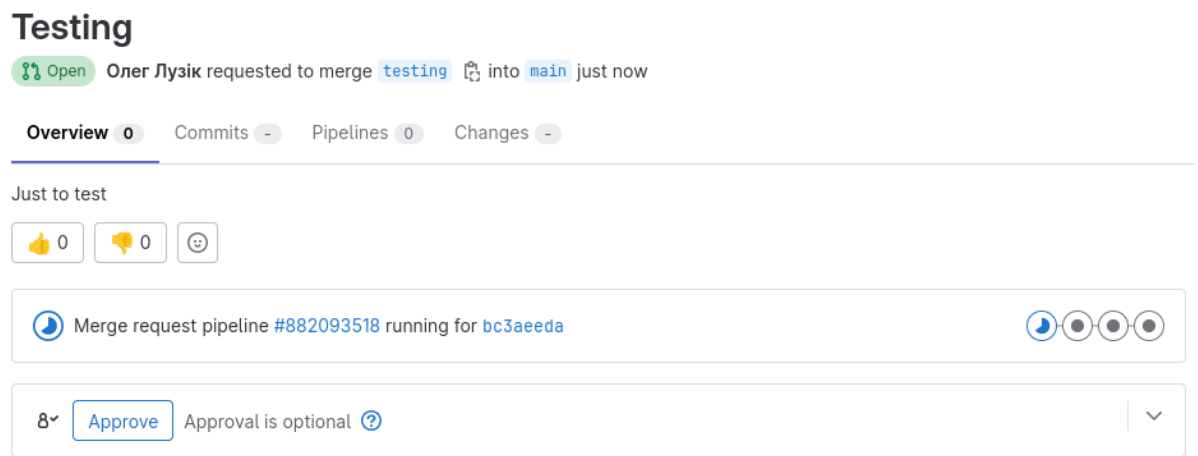


Рисунок 3.16 – Створення merge request

Після створення merge request в GitLab запуститься пайплайн, який автоматично створить review app.

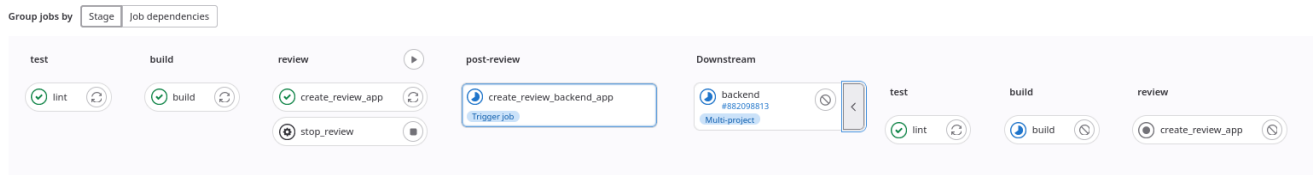


Рисунок 3.17 – Запуск Review App

Для перевірки роботи самого додатку, можна перейти за посиланням на вкладці "Environment" в GitLab.

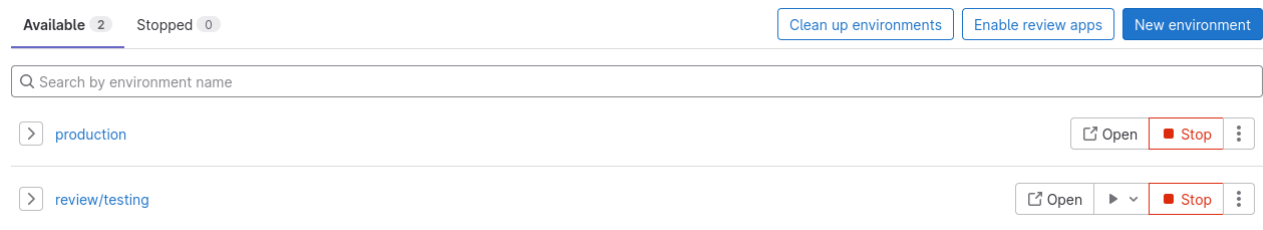


Рисунок 3.18 – Вкладка environment у Gitlab

Перевірки тестового середовища за допомогою команди “helm list” :

```
~ at gke_mystic-producer-387705_us-central1-c_cluster-for-ci-cd ()
+ helm list
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
nginx-ingress	default	1	2023-05-24 20:17:31.714991144 +0300 EEST	deployed	ingress-nginx-4.6.1	1.7.1
production	default	4	2023-05-29 10:50:46.47235304 +0000 UTC	deployed	app-0.2.0	0.1.1
testing	default	2	2023-05-29 15:03:27.814461654 +0000 UTC	deployed	app-0.2.0	0.1.1

Рисунок 3.19 – Результат команди helm list

Перевірка новостворених deployments:

```
~ at gke_mystic-producer-387705_us-central1-c_cluster-for-ci-cd ()
+ kubectl get deployments.apps
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-ingress-ingress-nginx-controller	1/1	1	1	4d21h
node-production-app	1/1	1	1	4d4h
node-testing-app	1/1	1	1	4m41s
postgres-production-app	1/1	1	1	4d4h
postgres-testing-app	1/1	1	1	4m41s
web-production-app	1/1	1	1	4d4h
web-testing-app	1/1	1	1	4m41s

Рисунок 3.20 – Результат команди kubectl get deployments

Перевірка Ingress:

```
~ at gke_mystic-producer-387705_us-central1-c_cluster-for-ci-cd ()
+ kubectl describe ingress testing-app
```

Name:	testing-app
Labels:	app.gitlab.com/app=ci-cd-workshop3913221-backend app.gitlab.com/env=review-testing-gzd91k app.kubernetes.io/instance=testing app.kubernetes.io/managed-by=Helm app.kubernetes.io/name=app app.kubernetes.io/version=0.1.1 helm.sh/chart=app-0.2.0
Namespace:	default
Address:	34.132.8.115
Ingress Class:	<none>
Default backend:	<default>
Rules:	
Host	Path Backends
testing.34-132-8-115.nip.io	/ web-testing-app-svc:80 (10.8.1.25:80,10.8.1.26:80,10.8.2.25:80) /api node-testing-app-svc:80 (10.8.1.25:3000,10.8.1.26:3000,10.8.2.25:3000)
Annotations:	kubernetes.io/ingress.class: nginx meta.helm.sh/release-name: testing meta.helm.sh/release-namespace: default nginx.ingress.kubernetes.io/ssl-redirect: false
Events:	
Type Reason Age From Message	
Normal Sync 2m19s (x3 over 5m34s) nginx-ingress-controller Scheduled for sync	

Рисунок 3.21 – Результат команди

Тестування новоствореного додатку на працездатність:

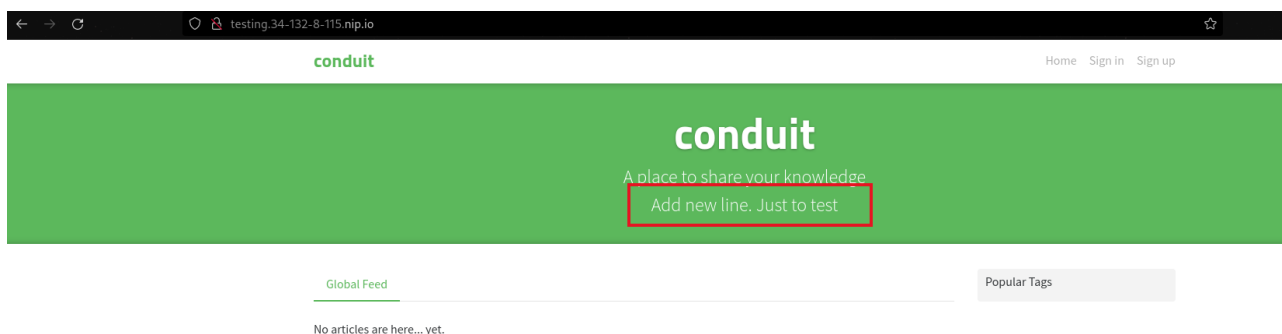


Рисунок 3.22 – Результат команди

Перевірка на зупинку review app. Під час натискання кнопки stop запускається manual job (stop_review) для зупинки review app:

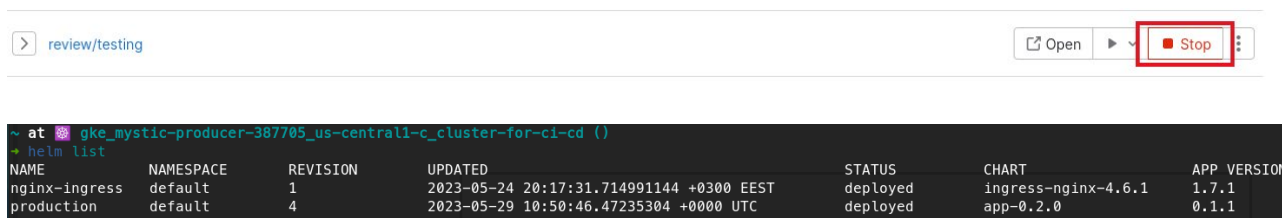


Рисунок 3.23 – Вивід команди helm list

Додаток був успішно протестований шляхом запуску пайплайну та створення review app для перевірки внесених змін. Цей процес дозволяє перевірити правильність роботи додатку перед злиттям з основною гілкою. Результати тестування підтверджують, що додаток працює належним чином у відокремленому середовищі, що сприяє забезпеченню якості та надійності розгорнутої версії.

Висновки до розділу 3

У третьому розділі було проведено налаштування CI/CD для розгортання проекту в Kubernetes кластері за допомогою Helm. Спочатку було розглянуто створення Docker файлів, які використовуються для пакування додатку у контейнери. Далі, було досліджено використання Helm, який є інструментом керування розгортанням та управлінням релізами у Kubernetes. Було описано процес налаштування CI/CD пайплайну для розгортання веб-стеку в Kubernetes

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

кластері, включаючи автоматизацію збірки, пакування та розгортання за допомогою Helm. Нарешті, проведено перевірку працездатності системи після розгортання. Цей розділ надав основні кроки та рекомендації для налаштування CI/CD процесу для розгортання проекту в Kubernetes кластері з використанням Helm.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

В даній кваліфікаційній роботі було розглянуто основні аспекти проектування та налаштування високодоступного веб-стеку з використанням контейнеризації та оркестрації Kubernetes. У першому розділі було надано теоретичні відомості про контейнеризацію, Docker та Kubernetes. Другий розділ присвячений проектуванню високодоступного веб-стеку, включаючи вибір технологій та інструментів, архітектуру системи та вибір інструменту для автоматизації CI/CD процесу. У третьому розділі було детально описано налаштування CI/CD для розгортання проекту в Kubernetes кластері з використанням Helm.

Застосування контейнеризації та оркестрації Kubernetes сприяє покращенню швидкості розгортання, масштабованості та надійності веб-стеку. Використання інструментів CI/CD дозволяє автоматизувати процес розробки, тестування та розгортання програмного забезпечення, забезпечуючи швидкі та безперервні поставки змін. Конкретно для даної кваліфікаційної роботи, для автоматизації CI/CD процесу було використано GitLab CI, що дозволило забезпечити ефективну роботу з кодом, створення контейнерів, налаштування Helm та автоматизувати розгортання проекту в Kubernetes кластері. Використання цих технологій та інструментів сприяє покращенню розробки та розгортання веб-додатків, забезпечує високу доступність, масштабованість та швидкість розгортання.

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. IBM. Containers vs. Virtual Machines. What is the Difference? [Електронний ресурс] – 2023 – Режим доступу: <https://www.ibm.com/cloud/blog/containers-vs-vms>
2. Kubernetes Docs. What is Kubernetes? [Електронний ресурс] – 2021 – Режим доступу: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes>
3. DockerHub. Stress containerized application [Електронний ресурс] – 2015 – Режим доступу: <https://hub.docker.com/r/progrium/stress>
4. Kubernetes Docs. Kubernetes Components [Електронний ресурс] – 2023 – Режим доступу: <https://kubernetes.io/docs/concepts/overview/components>
5. Docker Docs. Docker architecture [Електронний ресурс] – 2021 – Режим доступу: <https://docs.docker.com/get-started/overview/#docker-architecture>
6. Red Hat. What is orchestration? [Електронний ресурс] – 2020 – Режим доступу: <https://www.redhat.com/en/topics/automation/what-is-orchestration>
7. Kubernetes Docs. Run a Stateless Application Using a Deployment [Електронний ресурс] – 2021 – Режим доступу: <https://kubernetes.io/docs/tasks/run-application/run-stateless-application-deployment/>
8. Docker Docs. Docker architecture [Електронний ресурс] – 2021 – Режим доступу: <https://docs.docker.com/get-started/overview/#docker-architecture>
9. Amazon, Microsoft Or Google: Which Is The Best Play On Surging Cloud Infrastructure Demand? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.forbes.com/sites/greatspeculations/2020/05/14/amazon-microsoft-or-google-which-is-the-best-play-on-surging-cloud-infrastructure-demand/>
10. Comparing Kubernetes Services on AWS vs. Azure vs. GCP [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sumologic.com/blog/kubernetes-aws-azure-gcp/>

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

11. Developers bring their ideas to life with Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://www.docker.com/why-docker>
12. Kubernetes blog [Електронний ресурс] – Режим доступу до ресурсу: <https://cloudacademy.com/blog/category/kubernetes/>
13. Kubernetes vs. Docker: A Primer [Електронний ресурс] – Режим доступу до ресурсу: <https://containerjournal.com/topics/containerecosystems/kubernetes-vs-docker-a-primer/>
14. Why Google Cloud [Електронний ресурс] – Режим доступу до ресурсу: <https://cloud.google.com/why-google-cloud>
15. Мікросервісна архітектура [Електронний ресурс] – Режим доступу до ресурсу: - <https://medium.com/@IvanZmerzlyi/microservicesarchitecture-461687045b3d>.
16. 20 Best CI/CD Tools 2021 [Електронний ресурс] – Режим доступу до ресурсу: <https://fulcrum.rocks/blog/best-ci-cd-tools/>
17. What is Continuous Integration and Why it is Important? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.simplilearn.com/tutorials/devopstutorial/continuous-integration>
18. CI/CD [Електронний ресурс] – Режим доступу до ресурсу: <https://itglobal.com/ruru/company/glossary/ci-cd/>
19. What is CI/CD [Електронний ресурс] – Режим доступу до ресурсу: <https://www.redhat.com/en/topics/devops/what-is-ci-cd>
20. How Continuous Integration Works [Електронний ресурс] – Режим доступу до ресурсу: <https://www.appdynamics.com/blog/engineering/continuous-integrationworks-big-benefit-no-one-talks/>

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

Додаток А – Конфігураційні файли production додатку розгорнутого за допомогою Helm

apiVersion: v1

kind: ConfigMap

metadata:

name: postgres-config-production-app

labels:

helm.sh/chart: app-0.2.0

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

app.kubernetes.io/version: "0.1.1"

app.kubernetes.io/managed-by: Helm

app.gitlab.com/env: "production"

app.gitlab.com/app: "ci-cd-workshop3913221-backend"

data:

POSTGRES_DB: realworld

POSTGRES_USER: realworld

POSTGRES_PASSWORD: realworld

Source: app/templates/pv-claim.yaml

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: postgres-pv-claim-production-app

labels:

helm.sh/chart: app-0.2.0

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

```
app.kubernetes.io/version: "0.1.1"
app.kubernetes.io/managed-by: Helm
app.gitlab.com/env: "production"
app.gitlab.com/app: "ci-cd-workshop3913221-backend"
```

spec:

```
storageClassName: standard
```

```
accessModes:
```

```
- ReadWriteOnce
```

```
resources:
```

```
requests:
```

```
storage: 1Gi
```

```
# Source: app/templates/service/node.yaml
```

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
name: node-production-app-svc
```

```
labels:
```

```
helm.sh/chart: app-0.2.0
```

```
app.kubernetes.io/name: app
```

```
app.kubernetes.io/instance: production
```

```
app.kubernetes.io/version: "0.1.1"
```

```
app.kubernetes.io/managed-by: Helm
```

```
app.gitlab.com/env: "production"
```

```
app.gitlab.com/app: "ci-cd-workshop3913221-backend"
```

spec:

```
type: ClusterIP
```

```
ports:
```

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

- port: 80

targetPort: 3000

protocol: TCP

selector:

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

Source: app/templates/service/postgres.yaml

apiVersion: v1

kind: Service

metadata:

name: postgres-production-app-svc

labels:

helm.sh/chart: app-0.2.0

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

app.kubernetes.io/version: "0.1.1"

app.kubernetes.io/managed-by: Helm

app.gitlab.com/env: "production"

app.gitlab.com/app: "ci-cd-workshop3913221-backend"

spec:

type: ClusterIP

ports:

- port: 5432

targetPort: 5432

selector:

type: database

app.kubernetes.io/name: app

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    app.kubernetes.io/instance: production
---
# Source: app/templates/service/web.yaml
apiVersion: v1
kind: Service
metadata:
  name: web-production-app-svc
  labels:
    helm.sh/chart: app-0.2.0
    app.kubernetes.io/name: app
    app.kubernetes.io/instance: production
    app.kubernetes.io/version: "0.1.1"
    app.kubernetes.io/managed-by: Helm
    app.gitlab.com/env: "production"
    app.gitlab.com/app: "ci-cd-workshop3913221-backend"
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    app.kubernetes.io/name: app
    app.kubernetes.io/instance: production
---
# Source: app/templates/deployment/node.yaml
apiVersion: apps/v1
kind: Deployment

```

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

metadata:

name: node-production-app

labels:

helm.sh/chart: app-0.2.0

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

app.kubernetes.io/version: "0.1.1"

app.kubernetes.io/managed-by: Helm

app.gitlab.com/env: "production"

app.gitlab.com/app: "ci-cd-workshop3913221-backend"

spec:

replicas: 1

selector:

matchLabels:

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

template:

metadata:

labels:

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

spec:

imagePullSecrets:

- name: registry-gitlab-key

securityContext:

{}

containers:

- name: node

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

```

securityContext:
  readOnlyRootFilesystem: false
image: "registry.gitlab.com/ci-cd-workshop3913221/backend:main"
imagePullPolicy: Always
ports:
  - containerPort: 3000
env:
  - name: DB_HOST
    value: postgres-production-app-svc
  - name: "DB_PASSWORD"
    value: "realworld"
  - name: "DB_NAME"
    value: "realworld"
  - name: "DB_USER"
    value: "realworld"
  - name: "SECRET"
    value: "secret"
  - name: "NODE_ENV"
    value: "production"
resources:
  {}

```

Source: app/templates/deployment/postgres.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: postgres-production-app

labels:

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

```
helm.sh/chart: app-0.2.0
app.kubernetes.io/name: app
app.kubernetes.io/instance: production
app.kubernetes.io/version: "0.1.1"
app.kubernetes.io/managed-by: Helm
app.gitlab.com/env: "production"
app.gitlab.com/app: "ci-cd-workshop3913221-backend"
```

spec:

replicas: 1

selector:

matchLabels:

type: database

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

template:

metadata:

labels:

type: database

app.kubernetes.io/name: app

app.kubernetes.io/instance: production

spec:

imagePullSecrets:

- name: registry-gitlab-key

securityContext:

{}

volumes:

- name: postgres-volume-production-app

persistentVolumeClaim:

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    claimName: postgres-pv-claim-production-app
containers:
  - name: postgres
    image: "postgres:15.3-alpine"
    imagePullPolicy: IfNotPresent
    ports:
      - containerPort: 5432
    envFrom:
      - configMapRef:
          name: postgres-config-production-app
    volumeMounts:
      - mountPath: /var/lib/postgresql/data
        name: postgres-volume-production-app
        subPath: postgres
---
# Source: app/templates/deployment/web.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-production-app
  labels:
    helm.sh/chart: app-0.2.0
    app.kubernetes.io/name: app
    app.kubernetes.io/instance: production
    app.kubernetes.io/version: "0.1.1"
    app.kubernetes.io/managed-by: Helm
    app.gitlab.com/env: "production"
    app.gitlab.com/app: "ci-cd-workshop3913221-backend"

```

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: app
      app.kubernetes.io/instance: production
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app
        app.kubernetes.io/instance: production
    spec:
      imagePullSecrets:
        - name: registry-gitlab-key
      securityContext:
        {}
      containers:
        - name: web
          securityContext:
            readOnlyRootFilesystem: false
          image: "registry.gitlab.com/ci-cd-workshop3913221/frontend:main"
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          resources:
            {}
---
# Source: app/templates/ingress.yaml

```

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: production-app
  labels:
    helm.sh/chart: app-0.2.0
    app.kubernetes.io/name: app
    app.kubernetes.io/instance: production
    app.kubernetes.io/version: "0.1.1"
    app.kubernetes.io/managed-by: Helm
    app.gitlab.com/env: "production"
    app.gitlab.com/app: "ci-cd-workshop3913221-backend"
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
    - host: "main.34-132-8-115.nip.io"
      http:
        paths:
          - path: "/"
            pathType: Prefix
            backend:
              service:
                name: web-production-app-svc
                port:
                  number: 80
          - path: "/api"

```

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

pathType: Prefix

backend:

service:

name: node-production-app-svc

port:

number: 80

Додаток В – Конфігураційний файл gitlab-ci.yml для інфраструктурного
репозиторія

stages:

- lint
- publish
- upload

default:

image:

name: "alpine/helm:3.6.3"

entrypoint: ["/bin/sh", "-c"]

lint:

stage: lint

script:

- helm lint charts/app

package:

stage: publish

script:

- helm package charts/app -d pkgs/

artifacts:

paths:

- pkgs/*.tgz

expire_in: 1 week

upload:

stage: upload

image: curlimages/curl:latest

script:

- >

for d in pkgs/*.tgz; do

curl --request POST --user gitlab-ci-token:\$CI_JOB_TOKEN --form

"chart=@\$d"

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

```
"${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/packages/helm/api/stable/charts"
```

```
done
```

Додаток С – Конфігураційний docker-файл frontend додатку

```
FROM node:14.17.1-alpine as builder
```

```
WORKDIR /app
```

```
ENV PATH /app/node_modules/.bin:$PATH
```

```
ARG BACKEND_URL $BACKEND_URL
```

```
COPY package-lock.json ./
```

```
COPY package.json ./
```

```
RUN npm ci --silent
```

```
COPY ../
```

```
RUN REACT_APP_BACKEND_URL=$BACKEND_URL npm run build
```

```
# production
```

```
FROM nginx:stable-alpine
```

```
COPY --from=builder /app/build /usr/share/nginx/html
```

```
EXPOSE 80
```

```
CMD ["nginx", "-g", "daemon off;"]
```

Додаток D – Конфігураційний docker-файл backend додатку

```
FROM node:14.17.0
```

```
RUN npm install -g npm@latest
```

```
WORKDIR /usr/src/app
```

```
COPY package.json package-lock.json ./
```

```
RUN npm install
```

```
COPY ..
```

```
EXPOSE 3000
```

```
CMD [ "./bin/start.sh" ]
```

		Лузік О. С.			КІК.КР.Б - 123 - 11 - ПЗ	Арк.
		Фальковський І. Г.				
Змн.	Арк.	№ докум.	Підпис	Дата		73