Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ТЕХНОЛОГІЯ РОЗГОРТАННЯ ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ ВЕБ-САЙТУ В ХМАРНОМУ СЕРЕДОВИЩІ З ВИКОРИСТАННЯМ KUBERNETES

Виконав студент 2 курсу, групи 2КІ–22м
спеціальності 123 — Комп'ютерна інженерія
Нич В.О.
Керівник к.т.н., доц. каф. ОТ
Колесник І.С.
"07" 12 2023р.
Опонент к.т.н., доц. зав. каф. МБІС
Карпінець В.В.
"08" 12 2023р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«11» 12 2023 р.

" " 2023 р. ВНТУ 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень магістр
Спеціальність 123 Комп'ютерна інженерія

Затверджую
Завідувач кафедри
обчислювальної техніки
проф., д.т.н. О. Д. Азаров
_____
«_26__»_вересня_2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ничу Вадиму Олеговичу

1 Тема роботи «**Технологія розгортання обчислювальної інфраструктури веб-сайту в хмарному середовищі з використанням Kubernetes**», керівник роботи Колесник Ірина Сергіївна, д.т.н.,професор,затверджені наказом вищого навчального закладу від 18.09.223 року № 247.

2 Строк подання студентом роботи 9.12.2023 р.

3 Вихідні дані до роботи: інфраструктура Kubernetes, платформа для контейнеризації Docker, хмарний сервіс Docker Hub, рішення для оркестрації контейнерів Google Kubernetes Engine.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, огляд та аналіз технологій хмарних обчислень і контейнеразації, проектування хмарної інфраструктури, розгортання та управління веб-сайтом, аналіз та оптимізація продуктивності, висновки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): код основної kubernetes інфраструктури, код додатків для kubernetes., код для автоматизації, Dockerfile та код-вебсайту

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2, 3,4	Колесник І.С., к..т.н., доц. каф. ОТ		
5	Небава М.І., к.е.н., проф. каф. ЕПВМ		

7 Дата видачі завдання __.__._____ р.

8 Календарний план наведено в табл. 2.

Таблиця 2 — Календарний план

з/п	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	
	Постановка мети та задач роботи	20.10.23	
	Огляд та аналіз технологій хмарних обчислень і контейнеразації	27.10-30.10.23	
	Проектування хмарної інфраструктури	01.11-09.11.23	
	Вибір інструментарію розробки	10.11-17.11.23	
	Розгортання та управління веб-сайтом	18.11-.22.1123	
	Інтеграція з базою даних	23.11-26.11.23	
	Тестування продуктивності веб-додатку	27.11-31.11.23	
	Оптимізація використання ресурсів Kubernetes	01.12-04.12.23	
	Розрахунок економічної частини роботи	01.12-04.12.23	
	Оформлення МКР	05.12.23	
	Аналіз виконання роботи, висновки, додатки		
	Перевірка якості виконання МКР		

Студент _____ Нич.В.О..

Керівник роботи _____ Колесник І.С,

АНОТАЦІЯ

УДК 004.9

Нич В.О.

Технологія розгортання обчислювальної інфраструктури веб-сайту в хмарному середовищі з використанням Kubernetes.. Магістерська кваліфікаційна робота зі спеціальності 123 — комп'ютерна інженерія, освітня програма — комп'ютерна інженерія. Вінниця: ВНТУ, 2023, 97 с.
На укр.мові. Бібліогр.: 24 назв, рис 48, табл. 3.

Дана магістерська кваліфікаційна робота присвячена розробці та впровадженню ефективного методу автоматизації розгортання веб-додатку в Kubernetes за допомогою Helm..

Робота зосереджена на використанні технології Kubernetes для розгортання обчислювальної інфраструктури веб-сайтів у хмарному середовищі. Робота покликана не тільки надати теоретичне розуміння ключових компонентів Kubernetes та їх ролі у хмарній інфраструктурі, але й демонструє практичне застосування цих знань через розгортання реального веб-додатку.

Особлива увага приділяється інтеграції Kubernetes з Docker, а також використанню Helm як інструменту для управління пакетами в Kubernetes..

Ключові слова: захист, Kubernetes, хмарні технології, веб-додаток, контейнер, алгоритм, програмне забезпечення, графічні формати.

ANNOTATION

Nych V.O.

The technology for deploying the website's computing infrastructure in a cloud environment using Kubernetes. Master's qualification route in the specialty 123 — computer engineering, educational program computer engineering. Vinnitsa, VTNU, 2023, 97 p.

In the Ukr. leng. Libr. name 24, figure 48, table 3.

This master's thesis is devoted to the development and implementation of an effective method of automating the deployment of a web application in Kubernetes using Helm.

The work is focused on the use of Kubernetes technology to deploy the computing infrastructure of websites in the cloud environment. The work is designed not only to provide a theoretical understanding of the key components of Kubernetes and their role in a cloud infrastructure, but also to demonstrate the practical application of this knowledge through the deployment of a real web application.

Particular attention is paid to the integration of Kubernetes with Docker, as well as the use of Helm as a package management tool in Kubernetes.

Keywords: security, Kubernetes, cloud technologies, web application, container, algorithm, software, graphic formats.

ЗМІСТ

ВСТУП

Сучасний світ цифрових технологій переживає постійні зміни, що ставлять нові вимоги до способів розробки, розгортання та управління веб-ресурсами. З розвитком хмарних обчислень та контейнеризації програмного забезпечення, індустрія стикається з викликом забезпечення високої доступності, масштабованості та надійності веб-сервісів. В цьому контексті Kubernetes виступає як ключове рішення для оркестрації контейнерів, забезпечуючи ефективне управління ресурсами та автоматизацію розгортання.

Робота зосереджена на використанні технології Kubernetes для розгортання обчислювальної інфраструктури веб-сайтів у хмарному середовищі. Робота покликана не тільки надати теоретичне розуміння ключових компонентів Kubernetes та їх ролі у хмарній інфраструктурі, але й демонструє практичне застосування цих знань через розгортання реального веб-додатку.

Особлива увага приділяється інтеграції Kubernetes з Docker, а також використанню Helm як інструменту для управління пакетами в Kubernetes. Аналізуються переваги та недоліки Kubernetes у порівнянні з іншими рішеннями оркестрації контейнерів, а також розглядаються потенційні альтернативи Helm.

Робота включає розгляд стратегій автоматизації розгортання та управління за допомогою Continuous Integration/Continuous Deployment (CI/CD) процесів, які є невід'ємною частиною сучасних методологій розробки програмного забезпечення. Використання CI/CD значно підвищує ефективність розробки, забезпечуючи більшу оперативність та стабільність впровадження змін.

Метою дослідження є розробка та впровадження ефективного методу автоматизації розгортання веб-додатку в Kubernetes за допомогою Helm.

Задачі дослідження:

— проаналізувати базові поняття, терміни та принципи роботи середовища Kubernetes;

— дослідити інструменти для управління Kubernetes;

— створити кластер Kubernetes в Google Cloud Platform;

— розгорнути веб сайт на базі віртуальної машини в Docker;

— дослідити можливості масштабування та управління веб-сайтом на Kubernetes;

— провести дослідження можливості використання Kubernetes для розгортання веб сайтів з різними архітектурами;

— розробити методи підвищення надійності та безпеки веб-сайтів, розгорнутих на Kubernetes.

Об'єктом дослідження є процеси автоматизації розгортання веб-додатків у хмарному середовищі. Це включає аналіз поточних підходів та розробку нових стратегій для покращення ефективності процесу розгортання.

Предмет дослідження — інтеграція Kubernetes і Helm у процесі розгортання, а також використання інструментів автоматизації, таких як GitHub Actions, для створення більш ефективних та безпечних процесів розгортання.

Наукова новизна роботи Наукова новизна роботи полягає у розробці автоматизованого процесу розгортання Helm Chart через GitHub Actions, який об'єднує автоматизацію збірки та розгортання з використанням Kubernetes і Helm. Це забезпечує високий рівень гнучкості та масштабованості за рахунок інтеграції з хмарними сервісами, зокрема з Google Cloud, що дозволяє ефективно управляти ресурсами Kubernete.

Практичне значення одержаних результатів:

— розроблено методи підвищення надійності та безпеки веб сайтів, розгорнутих на Kubernetes;

— проведено комп’ютерне моделювання характеристик запропонованої системи, що показало розширення функціональних можливостей такої схеми

— проведено техніко-економічне обґрунтування доцільності застосування запропонованої технології розгортання обчислювальної інфраструктури веб-сайту в хмарному середовищі з використанням Kubernetes.

Апробацію результатів наукової роботи було проведено на науковій конференції «LIII Науково–технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024)», доповідь на тему: «Технологія розгортання обчислювальної інфраструктури веб-сайту в хмарному середовищі з використанням Kubernetes».

1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ ХМАРНИХ ОБЧИСЛЕНЬ І КОНТЕЙНЕРИЗАЦІЇ

1.1 Основні концепції хмарних обчислень

В останні десятиліття, хмарні обчислення змінили парадигму IT-індустрії, надаючи організаціям можливість ефективно використовувати ресурси, забезпечувати масштабованість та гнучкість. Ці технології дозволяють компаніям зосереджуватися на розробці та впровадженні інноваційних рішень, замість підтримки власної інфраструктури.

Далі розглянемо основні переваги хмарних обчислень. Насамперед, до них слід віднести масштабованість.

Масштабованість дає можливість легко збільшувати або зменшувати використання ресурсів відповідно до потреб, доступ до ресурсів з будь-якої точки світу, економія на придбанні та обслуговуванні обладнання.

У цьому контексті Kubernetes відіграє ключову роль. Kubernetes — це відкрите програмне забезпечення для автоматизації розгортання, масштабування та управління контейнеризованими застосунками. Він був розроблений Google на основі їхнього досвіду з управління контейнерами на величезному масштабі і відтоді став стандартом у галузі хмарних обчислень.

Основною перевагою Kubernetes є пропозиція оркестрації контейнерів, що автоматизує розгортання та управління контейнеризованими застосунками, забезпечуючи їх високу доступність та надійність.

В залежності від навантаження, Kubernetes може автоматично збільшувати або зменшувати кількість реплік застосунку.

Kubernetes забезпечує ефективне використання ресурсів, розподіляючи навантаження між контейнерами.

Важливою складовою ефективної роботи в хмарному середовищі є здатність до швидкого розгортання та управління застосунками, і тут Kubernetes відіграє ключову роль, надаючи інструменти для управління складними системами на різних масштабах. Kubernetes є найбільш

популярним контейнерним оркестратором, але існує ряд інших альтернатив. До найпопулярніших аналогів Kubernetes можна віднести деякі наведені нижче.

Docker Swarm — це контейнерний оркестратор, розроблений компанією Docker. Він є безкоштовним і відкритим кодом, і він має схожий на Kubernetes функціонал.

Mesos — це контейнерний оркестратор, розроблений компанією Apache. Він є більш масштабованим, ніж Kubernetes, і він може використовуватися для оркестрування контейнерів, які не є Docker, рисунок 1.1.’

Характеристика	Kubernetes	Docker Swarm	Mesos
Статус	Відкритий код	Відкритий код	Відкритий код
Вартість	Безкоштовний або платний	Безкоштовний	Безкоштовний або платний
Масштабованість	Висока	Висока	Дуже висока
Зручність використання	Середня	Середня	Середня
Мобільність	Висока	Висока	Висока
Складність	Середня	Низька	Висока
Безпека	Середня	Середня	Висока
Основні функції	Складні оркестрації, автоматизація, управління ресурсами	Складні оркестрації, автоматизація, управління ресурсами	Складні оркестрації, автоматизація, управління ресурсами
Технології	Docker	Docker	Docker, Mesos
Рекомендується для	Підприємства, які потребують масштабованої, зручної у використанні та мобільної контейнерної інфраструктури	Підприємства, які потребують простої та доступної контейнерної оркестрації	Підприємства, які потребують високомасштабованої контейнерної оркестрації

Рисунок 1.1 — Порівняння контейнерних оркестраторів

Kubernetes має ряд переваг, які роблять його популярним вибором для контейнерних оркестраторів. Насамперед, слід згадати про скалоздатність.

Завдяки скалоздатності Kubernetes може масштабувати до мільйонів контейнерів. Kubernetes зручний для використання, так як має широкий спектр інструментів і ресурсів, які полегшують його використання, також Kubernetes можна використовувати на будь-якій інфраструктурі, яка підтримує контейнери, завдяки чому він є дуже мобільним.

Крім іншого Kubernetes має відкриту платформу з активним розвитком, завдяки чому він постійно розвивається та вдосконалюється, забезпечуючи нові можливості та підтримку сучасних технологій.

Для Kubernetes існує безліч інструментів для моніторингу та логування, що дозволяє легко відстежувати стан системи та швидко реагувати на проблеми.

Kubernetes виступає як передова та адаптивна платформа, яка використовується для оркестрації контейнерів та управління додатками. Ця система підтримується широкою спільнотою користувачів та розробників, яка надає значний обсяг ресурсів для навчання, підтримки та спільної роботи. Вибір Kubernetes як основи для проекту гарантує інвестицію в перевірену, надійну та гнучку платформу, яка здатна вирішувати різноманітні задачі оркестрації контейнерів та управління додатками на різних етапах розвитку проекту.

Однак у Kubernetes також є ряд недоліків, які слід враховувати. До недоліків Kubernetes відносяться:

— складність вивчення та використання;

— вартість може бути дорогою для навіть великих підприємств;

— безпека — Kubernetes може бути небезпечним, якщо його не налаштувати належним чином;

Kubernetes є відмінним рішенням для мікросервісної архітектури. Якщо ваша система побудована на мікросервісах, Kubernetes надає потужні інструменти для їх оркестрації, управління життєвим циклом, масштабування

та моніторингу. Завдяки підтримці контейнеризації Kubernetes дозволяє створювати легко переносимі та консистентні оточення для розробки, тестування та виробництва.

1.2 Переваги та недоліки контейнеризації

Контейнеризація, особливо завдяки Docker та Kubernetes, впливає на способи розробки, тестування та розгортання програмного забезпечення. Docker служить основою для створення та управління контейнерами, тоді як Kubernetes забезпечує розподілене управління цими контейнерами в продуктивному середовищі. Це взаємодія значно підвищує ефективність процесу розробки та розгортання застосунків.

Docker пропонує розробникам зручні інструменти для створення, тестування та запуску додатків в контейнерах, що забезпечує портативність і легковаговість, оскільки кожен контейнер включає в себе всі необхідні залежності програми. Це дозволяє застосункам працювати консистентно в різних середовищах.

Kubernetes, у свою чергу, доповнює можливості Docker, надаючи механізми для ефективного управління багатьма контейнерами, їх автоматичного масштабування та відновлення. Завдяки Kubernetes, контейнеризовані застосунки можуть бути динамічно розподілені та управлятися у складних продуктивних середовищах.

Важливо розуміти, що хоча контейнеризація пропонує такі переваги, як портативність, легковаговість та швидкість запуску, існують виклики, пов'язані з безпекою та управлінням залежностями. Контейнери, що ділять ядро ОС, можуть бути менш ізольованими від віртуальних машин, що створює потенційні ризики безпеки. Також управління залежностями та сховищем даних може бути складнішим у контейнеризованому середовищі.

1.2.1 Контейнеризація з Docker

Docker є популярною платформою для контейнеризації, що дозволяє розробникам упаковувати застосунки та їх залежності в контейнери. Важливо почати з визначення Docker та його ролі в контейнеризації:

Контейнер — це стандартизована одиниця програмного забезпечення, яка упаковує код додатка разом зі всіма його залежностями, що гарантує швидке та надійне розгортання додатків.

Docker дозволяє розробникам легко створювати, тестувати та запускати додатки в ізольованому середовищі, яке називається контейнером.При контейнеризації він використовується для стандартизації середовища застосунків, забезпечуючи консистентність між різними розробницькими та продуктивними середовищами

Docker Hub — це хмарний сервіс, який надає користувачам можливість зберігати та розподіляти Docker-образи. Образи Docker — це збірки програмного забезпечення та всіх його залежностей, які можуть бути запущені в Docker-контейнері.

Docker Hub містить два типи репозиторіїв, перший з них — публічні. Публічні репозиторії доступні для всіх користувачів. Вони містять широкий спектр образів, які можна використовувати для запуску широкого спектру програмного забезпечення. Другий — приватні, приватні репозиторії доступні лише для авторизованих користувачів. Вони можуть використовуватися для зберігання образів, які є власністю компанії або організації.

Docker Hub надає ряд функцій, які роблять його цінним інструментом для розробників та адміністраторів систем: Він має простий у використанні інтерфейс, який дозволяє користувачам швидко та легко завантажувати образи. Крім того, Docker Hub надає ряд функцій безпеки, які допомагають захистити образи від несанкціонованого доступу.

Docker Hub містить широкий вибір образів, які можна використовувати для запуску широкого спектру програмного забезпечення.

Зважаючи на вищевикладене, Docker Hub є цінним інструментом для розробників та адміністраторів систем, які його використовують для розгортання та управління веб-додатками та іншими програмними системами.

При створення образів Docker використовується Dockerfile для вказівки кроків створення образу, це текстовий файл, що містить всі команди, необхідні для зборки образу Docker. Образ Docker – це шаблон, який містить інструкції для створення контейнера. Він включає базовий образ, команди, змінні середовища, вихідні порти тощо.

Переваги використання Docker:

— забезпечення консистентності середовища від розробки до продакшену завдяки чому він сумісnсний та портативний;

— зменшення навантаження та використання ресурсів порівняно з традиційними віртуальними машинами;

— полегшення процесу розгортання та масштабування застосунків, відокремлення застосунків один від одного, забезпечуючи безпеку та незалежність.

Docker став незамінним інструментом у розробці та розгортанні сучасних проектів. Він дозволяє розробникам створювати, тестувати та постачати додатки з високою швидкістю та надійністю. Контейнери стали стандартом для побудови мікросервісної архітектури та розвитку сучасних хмарних застосунків.

1.2.2 Оркестрація контейнерів з Kubernetes

Однією з ключових складових успішної контейнеризації є ефективне управління та оркестрація контейнерами. У цьому розділі ми розглянемо оркестрацію контейнерів з використанням системи Kubernetes, яка є

високофункціональним інструментом для автоматизованого управління контейнерами.

Kubernetes — це відкрите програмне забезпечення для оркестрації та управління контейнерами. Він надає зручний інтерфейс для розгортання, масштабування і управління контейнеризованими додатками у реальному часі. Основні переваги використання Kubernetes включають:

Автоматичне масштабування кількості контейнерів для відповіді на навантаження, що забезпечує стійкість та доступність додатків;

Самовідновлення передбачає, що система має вбудовані механізми для виявлення та відновлення контейнерів, які перестали працювати, забезпечуючи надійність системи.

Використання декларативних підходів до конфігурації, що дозволяють описувати бажаний стан системи і самостійне забезпеченння досягнення цього стану.

Для використання Kubernetes необхідно створити кластер, який складається з вузлів (nodes). Кожен вузол може бути фізичним сервером або віртуальною машиною, яка виконує контейнери. Kubernetes керує розгортанням та управлінням контейнерами на цих вузлах.

При дослідженні Kubernetes слід звернути увагу на нище зазначені основні принципи.

Перший з них стосується поділу робочих навантажень (Workload Division): Kubernetes дозволяє розгортати додатки у вигляді робочих навантажень, таких як розгортання (Deployments) та запуски (Pods). Це дозволяє керувати різними частинами додатків.

Другий принцип полягає у використанні служби (Services): Kubernetes надає механізми для внутрішньої та зовнішньої експозиції додатків, щоб забезпечити комунікацію між контейнерами та зовнішніми додатками.

Третій принцип стосується сховищ та ресурсів (Storage and Resources): Kubernetes дозволяє зберігати дані та розподіляти ресурси між контейнерами за допомогою об'єктів, таких як Persistent Volumes та Resource Quotas.

Отдже, оркестрація полягаєм в тому, що контейнери з використанням Kubernetes стали стандартом у сфері сучасної розробки та розгортання програмного забезпечення. Цей інструмент допомагає підвищити доступність, надійність та ефективність додатків, які використовують контейнери.

1.3 Хмарні провайдери

Крім самостійного розгортання інфраструктури для веб-сайту, існують також хмарні провайдери, які пропонують цю послугу. Хмарні провайдери мають ряд переваг над самостійним розгоренням інфрастукрури:

— простоту використання для розгортання веб-сайтів, яка полягає в тому що сервіси базуються на хмарній технології, надаючи зручні та легкодоступні засоби для запуску веб-сайтів; вони знижують технічний бар'єр для користувачів, яким не потрібно обтяжувати себе детальним розумінням хмарної інфраструктури, щоб ефективно використовувати ці платформи;

— можливість заощадити час який ви б витратили на розгортання інфраструктури в ручну;

— забезченення безпеки, високого рівння, оскільки хмарні провайдери розроблені і підтримуються досвідченими командами фахівців.

1.3.1 Види хмарних провайдерів

Amazon Web Services (AWS) пропонує Elastic Beanstalk, який забезпечує просте рішення для розгортання веб-сайтів. Microsoft Azure пропонує Azure App Service, який також спрощує процес розгортання веб-додатків. Google Cloud Platform (GCP) пропонує App Engine, який є аналогом Elastic Beanstalk та Azure App Service, надаючи автоматичне масштабування та управління без серверів.

IBM Cloud надає Cloud Foundry, платформу як сервіс (PaaS), яка дозволяє розробникам швидко створювати, розгортати та масштабувати

додатки в хмарному середовищі. Oracle Cloud пропонує Oracle Cloud Infrastructure, яка включає в себе різноманітні сервіси, такі як віртуальні машини, мережі, та зберігання даних, а також спеціалізовані рішення для різних бізнес-задач.

DigitalOcean, хоч і менший провайдер порівняно з гігантами ринку, відомий своєю простотою та зручністю, особливо для невеликих проектів та стартапів. Його Droplets (віртуальні приватні сервери) та Kubernetes-базовані рішення надають ефективні та легкі у використанні інструменти для розгортання додатків.

Всі ці провайдери відрізняються своїми унікальними особливостями, варіантами ціноутворення, та підходами до управління хмарними ресурсами, що дає користувачам широкий вибір в залежності від їхніх потреб та бюджету. Важливо враховувати такі фактори, як масштабованість, доступність, безпека та підтримка при виборі хмарного провайдера для конкретного проекту.

1.3.2 Критерії вибору хмарних рішень

При виборі хмарних рішень, таких як Google Cloud Platform (GCP), важливо враховувати декілька ключових аспектів. По-перше, розгляд великого спектру функцій та можливостей, які пропонує GCP, є вирішальним. Це включає контейнеризацію, оркестрацію, машинне навчання, штучний інтелект, бази даних, аналітику, інфраструктуру та бізнес-аналітику.

Інноваційність також є важливим фактором. GCP інвестує у передові технології, такі як Kubernetes, TensorFlow, Vertex AI, Cloud Spanner, що можуть бути вирішальними для технологічно орієнтованих підприємств. Гнучкість і масштабованість GCP дозволяють підприємствам легко адаптуватися до змінюваних потреб, забезпечуючи при цьому високий рівень безпеки для захисту даних та додатків.

Конкретні переваги GCP порівняно з AWS і Azure включають більш низькі ціни та глобальну інфраструктуру, що може бути важливо для

міжнародних підприємств. Однак важливо також розглянути можливі недоліки, такі як менша частка ринку GCP порівняно з AWS та Azure, що може вплинути на доступність навчальних ресурсів та підтримки.

Вибираючи хмарний провайдер, необхідно зважати на специфічні потреби та вимоги бізнесу, оцінювати вартість рішень, а також враховувати географічну доступність та вимоги до безпеки. GCP, з його широким спектром інноваційних функцій та гнучкістю, може бути ідеальним вибором для підприємств, які шукають надійний, масштабований та ефективний хмарний сервіс.

GCP має ряд переваг над іншими хмарними провайдерами, включаючи:

— широкий спектр функцій і можливостей, таких як: контейнери та оркестрація, машине навчання та штучний інтелект, бази даних і аналітики, бізнес-аналітика, інфраструктура;

— постійні інвестиції в інноваційні технології, такі як: Kubernetes, TensorFlow, Vertex AI, Cloud Spanner;

— гнучкість і масштабованість, які дозволяють підприємствам легко адаптуватися до своїх потреб;

— широкий спектр функцій безпеки, які допомагають захистити дані та програми підприємств.

GCP має ряд конкретних переваг над AWS і Azure який включає:

— GCP пропонує широкий спектр спеціалізованих функцій, які можуть бути корисними для підприємств у певних галузях;

— більш низькі ціни, через що він може бути більш доступним для деяких підприємств, ніж AWS і Azure;

— географічна присутність, що має глобальну інфраструктуру, яка може бути важливою для підприємств, які працюють у декількох країнах.

Недоліки GCP:

— має меншу частку ринку, ніж AWS і Azure, це може означати, що у GCP може бути менше навчальних матеріалів, ресурсів та підтримки, ніж у інших хмарних провайдерів;

— не може пропонувати всі функції, які пропонують AWS і Azure.

Це може бути важливим для деяких підприємств, які потребують певних функцій.

Не зважаючи на всі недоліки, GCP є потужним хмарним провайдером, який пропонує широкий спектр функцій, можливостей, інновацій та переваг. Він може бути хорошим вибором для підприємств, які шукають гнучкий, масштабований і безпечний хмарний сервіс, рисунок 1.2.

Характеристика	GCP	AWS	Azure
Статус	Платний	Платний	Платний
Масштабованість	Висока	Висока	Висока
Доступність	Висока	Висока	Висока
Безпека	Висока	Висока	Висока
Ціни	За використання	За використання	За використання
Функції	Широкий спектр функцій	Широкий спектр функцій	Широкий спектр функцій
Спеціалізовані функції	Машине навчання та штучний інтелект, БАЙД, бізнес-аналітика, healthcare	Спеціалізовані функції для конкретних галузей, таких як роздрібна торгівля, фінансові послуги та технології	Спеціалізовані функції для державних організацій
Рекомендується для	Підприємства, які потребують широкого спектру функцій та можливостей, а також спеціалізованих функцій для машинного навчання та штучного інтелекту	Підприємства, які потребують широкого спектру функцій та можливостей, а також спеціалізованих функцій для конкретних галузей	Підприємства, які потребують спеціалізованих функцій для державних організацій

Рисунок 1.2 — Порівння хмарних провайдерів.

2 ПРОЕКТУВАННЯ ХМАРНОЇ ІНФРАСТРУКТУРИ

2.1 Вибір хмарного провайдера

Вибір хмарного рішення для оркестрації контейнерів є важливим етапом при розгортанні веб-сайту в хмарі. Правильний вибір забезпечує безпеку, масштабованість та доступність веб-сайту.

При виборі готового хмарного провайдера для веб-сайту слід враховувати такі фактори:

— функціональність яку пропонує хмарний провайдер, всі необхідні функції для веб-сайту;

— ціна різних хмарних провайдерів, щоб знайти найбільш вигідний варіант;

— можливості, що хмарний провайдер підтримує масштабування вашого веб-сайту відповідно до зростання навантаження;

— надійність, безпека та підтримка.

2.2 Вибір хмарного Kubernetes

При виборі рішення для оркестрації контейнерів слід враховувати такі фактори: функціональність, складність, вартість.

Для розгортання веб-сайту в хмарі було обрано Google Kubernetes Engine (GKE). GKE є повністю керованим рішенням для оркестрації контейнерів, яке пропонує наступні переваги.

Глибока інтеграція з хмарною платформою Google Cloud Platform (GCP): GKE є нативним сервісом GCP, що забезпечує безшовну інтеграцію з іншими службами платформи, такими як Cloud Storage, Cloud SQL, Cloud Load Balancing та Cloud Monitoring; ця інтеграція спрощує розгортання та управління веб-сайтом, забезпечуючи єдиную точку входу для всіх необхідних ресурсів.

Повністю кероване рішення, GKE знімає з адміністраторів навантаження щодо управління інфраструктурою Kubernetes; Google

відповідає за оновлення кластера, патчі безпеки, масштабування та відновлення, дозволяючи розробникам зосередитися на розробці та розгортанні веб-сайту.

Висока масштабованість та доступність, завдяки чомуGKE дозволяє легко масштабувати веб-сайт вгору або вниз відповідно до потреб трафіку завдяки автоматичному масштабуванню та самовідновленню; кластер GKE забезпечує високу доступність веб-сайту, навіть в разі пікового навантаження.

Розширена безпека з вбудовованими передовими функціями, такими як Shielded GKE Nodes, Confidential GKE Nodes та Workload Identity, що забезпечують захист контейнеризованих робочих навантажень від несанкціонованого доступу та уразливостей.

Зручний інтерфейс та інструменти: GKE управляється через інтуїтивно зрозумілу Google Cloud Console та gcloud CLI, що спрощує розгортання, управління та моніторинг кластера, при цьому додатково, доступні інструменти, такі як Cloud Code, що дозволяють розробникам розгортати та керувати Kubernetes-застосуваннями безпосередньо з їх IDE.

GKE підтримується великим і активним співтовариством розробників та користувачів, що забезпечує доступ до широкого спектру документації, навчальних матеріалів, форумів та груп підтримки; офіційна підтримка Google також доступна для клієнтів, що потребують додаткової допомоги.

Google Cloud Kubernetes є одним з найпопулярніших рішень для оркестрації контейнерів. До інших аналогів GKE відносяться: Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), IBM Cloud Kubernetes Service , Mesosphere DC/OS.

Ці рішення пропонують схожі можливості, але мають свої унікальні переваги та недоліки. При виборі рішення для оркестрації контейнерів слід враховувати свої індивідуальні потреби та вимоги. Kubernetes та Google Kubernetes Engine (GKE) є пов'язаними, але різними

технологіями у сфері управління контейнерами, і відповідно, їх аналоги також відрізняються.

Kubernetes є відкритим програмним забезпеченням для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Аналоги Kubernetes зосереджуються на оркестрації контейнерів і можуть включати такі рішення, як OpenShift (Red Hat), Docker Swarm та інші. Ці системи забезпечують подібні функціональні можливості, як у Kubernetes, наприклад, управління класетрами, автоматичне розгортання та самовідновлення, але можуть мати різні характеристики щодо інтеграції, масштабованості та управлінських інструментів.

GKE є хмарним рішенням від Google, яке надає управління Kubernetes як сервіс. Це означає, що Google бере на себе обслуговування інфраструктури Kubernetes, включно з управлінням кластерами, безпекою та масштабуванням. Аналоги GKE зосереджуються на наданні Kubernetes хмарного сервісу. Відмінності між цими сервісами полягають у деталях їх реалізації, інтеграції з хмарними сервісами та специфічних функціях, які вони надають, таких як безпека, моніторинг, автоматичне масштабування та інтеграція з хмарними інструментами.

Далі наведені аналоги GKE хмарних Kubernetes.

Amazon Elastic Kubernetes Service (EKS) — це хмарна служба Kubernetes, яка пропонується Amazon Web Services. EKS є готовим до використання рішенням, яке надає всі необхідні компоненти для розгортання кластера Kubernetes. EKS також пропонує широкий спектр функцій безпеки та керування.

Azure Kubernetes Service (AKS) — це хмарна служба Kubernetes, яка пропонується Microsoft Azure. AKS є готовим до використання рішенням, яке надає всі необхідні компоненти для розгортання кластера Kubernetes. AKS також пропонує широкий спектр функцій безпеки та керування.

IBM Cloud Kubernetes Service (IKS) — це хмарна служба Kubernetes, яка пропонується IBM Cloud. IKS є готовим до використання рішенням, яке

надає всі необхідні компоненти для розгортання кластера Kubernetes. IKS також пропонує широкий спектр функцій безпеки та керування.

Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE) — це хмарна служба Kubernetes, яка пропонується Oracle Cloud Infrastructure. OKE є готовим до використання рішенням, яке надає всі необхідні компоненти для розгортання кластера Kubernetes. OKE також пропонує широкий спектр функцій безпеки та керування.

Red Hat OpenShift Service on AWS (ROSA) — це хмарна служба Kubernetes, яка пропонується Red Hat та Amazon Web Services. ROSA є готовим до використання рішенням, яке надає всі необхідні компоненти для розгортання кластера Kubernetes. ROSA також пропонує широкий спектр функцій безпеки та керування.

VMware Tanzu Kubernetes Grid Service (TKGS) — це хмарна служба Kubernetes, яка пропонується VMware. TKGS є готовим до використання рішенням, яке надає всі необхідні компоненти для розгортання кластера Kubernetes. TKGS також пропонує широкий спектр функцій безпеки та керування. На рисунку 2.1 приведено характеристики хмарних Kubernetes.

Характеристика	GKE	EKS	AKS	IKS	OKE	ROSA	TKGS
Хмаровий провайдер	Google Cloud Platform	Amazon Web Services	Microsoft Azure	IBM Cloud	Oracle Cloud Infrastructure	AWS	VMware
Готовність до використання	Висока	Висока	Висока	Висока	Середня	Висока	Середня
Ефективність	Висока	Висока	Висока	Висока	Середня	Висока	Середня
Безпека	Висока	Висока	Висока	Висока	Висока	Висока	Висока
Гнучкість	Середня	Середня	Середня	Середня	Висока	Висока	Висока
Ціна	Середня	Середня	Середня	Середня	Низька	Середня	Висока
Обмеження	Деякі	Деякі	Деякі	Деякі	Деякі	Деякі	Деякі

Рисунок 2.1 — Порівняння хмарних Kubernetes

2.3 Конфігурація Kubernetes кластеру

Після вибору GKE як рішення для оркестрації контейнерів, було розроблено конфігурацію кластера. Конфігурація кластера включала в себе кроки наведені нище.

Створення кластера GKE за допомогою Google Cloud Console.

Налаштування мережі кластера для забезпечення безпеки та доступу між контейнерами.

Налаштування зберігання кластера для забезпечення надійності та масштабування.

Налаштування безпеки кластера для захисту від несанкціонованого доступу та уразливостей, рисунок 2.2.

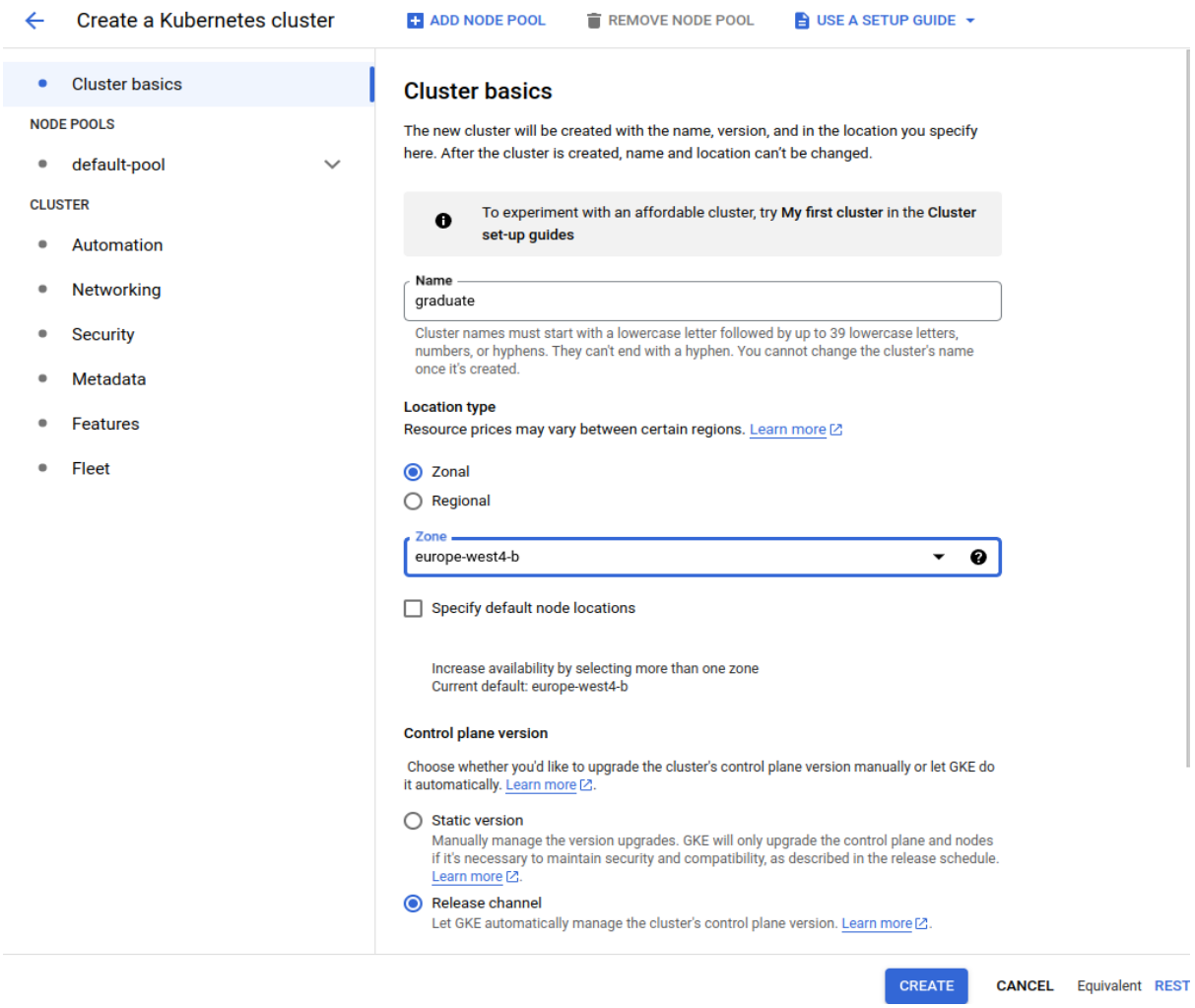


Рисунок 2.2 — Створення кластера через Google Cloud Console

2.1.1 Конфігурація мережі

Мережа є важливим аспектом конфігурації Kubernetes кластера. Мережа забезпечує зв'язок між контейнерами всередині кластера. Для забезпечення безпеки та доступу між контейнерами, мережа кластера була налаштована за допомогою Kubernetes Network Policies. Процес налаштування наведено на рисунку 2.3.

Networking

Private cluster	Disabled	🔒
Control plane global access	Disabled	✎
Network	default	🔒
Subnet	default	🔒
Stack type	IPv4	✎
Private control plane's endpoint subnet	default	🔒
VPC-native traffic routing	Enabled	🔒
Cluster Pod IPv4 range (default)	10.112.0.0/14	🔒
Cluster Pod IPv4 ranges (additional)	None	✎
Maximum pods per node	110	🔒
IPv4 service range	10.116.0.0/20	🔒
Intranode visibility	Disabled	✎
HTTP Load Balancing	Enabled	✎
Subsetting for L4 Internal Load Balancers	Disabled	✎
Control plane authorized networks	Disabled	✎
Calico Kubernetes Network policy	Disabled	✎
Dataplane V2	Disabled	🔒
DNS provider	Kube-dns	✎
NodeLocal DNSCache	Disabled	
Multi-networking PREVIEW	Disabled	🔒

Рисунок 2.3 —Мережеві параметри кластеру

Кластер Kubernetes, є публічним, оскільки функцію private cluster вимкнено. Це означає, що кластер доступний з будь-якого місця з доступом до Інтернету, що нам і потрібно для того, що б у користувачів був доступ до веб-сайту.

Кластер використовує IPv4-адреси, оскільки тип стека встановлено на IPv4. Це означає, що піди в кластері можуть використовувати IPv4-адреси для спілкування один з одним.

Трафік між вузлами кластера передається через VPC, в якому розташований кластер, оскільки маршрутизація трафіку через VPC увімкнена. Це підвищує безпеку кластера, оскільки трафік не передається через відкритий Інтернет.

Кластер може використовувати діапазон IPv4-адрес 10.112.0.0/14 для створення підів. Цей діапазон містить 262144 IPv4-адреси, що дозволяє кластеру масштабуватися до великих розмірів.

Контрольна площина кластера доступна лише з того регіону, в якому розташований кластер, оскільки контрольна площина глобального доступу вимкнена.

Кластер може використовувати діапазон IPv4-адрес 10.116.0.0/20 для створення служб. Цей діапазон містить 4096 IPv4-адрес, що дозволяє кластеру розміщувати багато служб.

Вузли кластера не можуть бачити один одного безпосередньо, оскільки внутрішня видимість вузлів вимкнена. Це означає, що вузли повинні використовувати мережу для спілкування один з одним.

Кластер може використовувати балансування навантаження для розподілу трафіку між подами.

2.1.2 Налаштування зберігання

Сховище є важливим аспектом конфігурації Kubernetes кластера. Сховище забезпечує зберігання даних контейнерів. Для забезпечення надійності та масштабування, зберігання кластера було налаштовано за допомогою Persistent Volumes.

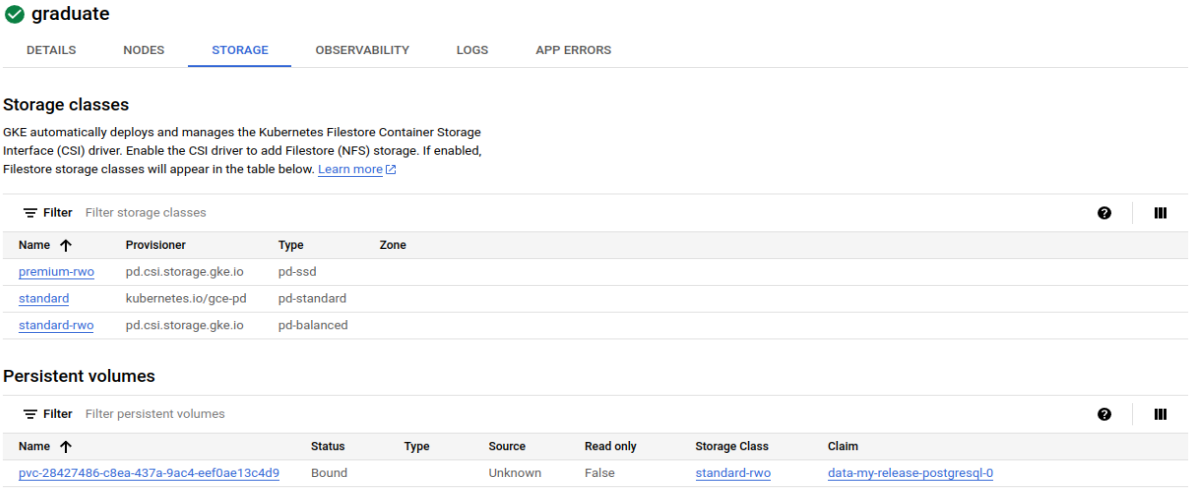


Рисунок 2.4 — Сховища Kubernetes кластеру

У кластері є два класи сховищ: premium-rwopd.csi.storage.gke.io: Цей клас сховища використовує SSD-диски для забезпечення високої продуктивності читання/запису. Він підходить для застосунків, які потребують швидкого доступу до даних, таких як бази даних.

standard-balanced.csi.storage.gke.io: Цей клас сховища використовує диски загального призначення для забезпечення балансу між продуктивністю та ціною. Він підходить для загального використання, де швидкість не є критичною.

Також у кластері є один постійний том: pvc-28427486-c8ea-437a-9ac4-eef0ae13c4d9: Цей том використовує клас сховища standard-rwopd. Це означає, що він використовує диск загального призначення та дозволяє читання та запис даних. Він пов'язаний з клеймом data-my-release-postgresql-0, що вказує на те, що він використовується для зберігання даних бази даних PostgreSQL.

2.1.3 Налаштування безпеки

Безпека є важливим аспектом конфігурації Kubernetes кластера. Безпека забезпечує захист контейнеризованих робочих навантажень від

несанкціонованого доступу та уразливостей. Для забезпечення безпеки кластера, були налаштовані наступні функції:

Shielded GKE Nodes: Ця функція забезпечує захист контейнеризованих робочих навантажень від несанкціонованого доступу до пам'яті та жорстких дисків;

Confidential GKE Nodes: Ця функція забезпечує захист контейнеризованих робочих навантажень від несанкціонованого доступу до конфіденційної інформації;

Workload Identity: Ця функція дозволяє контейнеризованим робочим навантаженням аутентифікуватися за допомогою існуючих систем ідентифікації, таких як Google Cloud Identity and Access Management (IAM), рисунок 2.5.

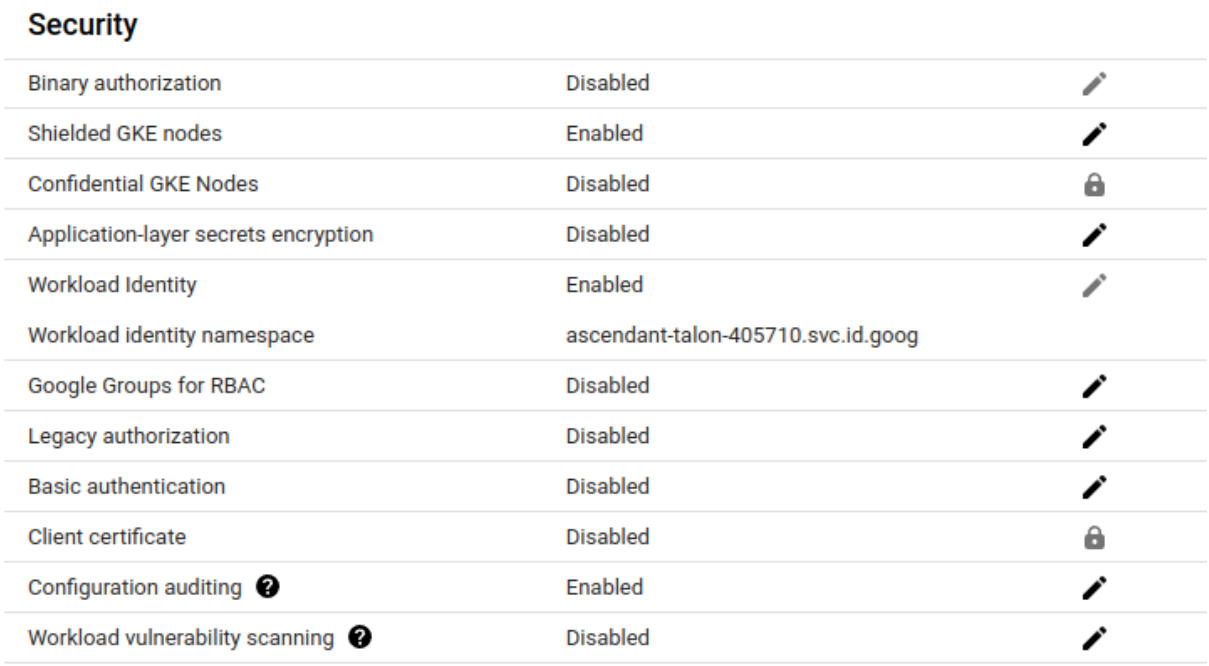


Рисунок 2.5 — Нашалтування безпеки кластеру.

2.4 Створення та налаштування Docker контейнера

Нижче наведено основні елементами іфраструктури сайту в Docker.

app.js — це основний файл додатку Node.js, який визначає логіку сервера та його взаємодію з клієнтами. Цей файл виконує ключову роль у запуску веб-сайту всередині контейнера Docker.

node_modules — директорія, яка містить усі залежності проекту, встановлені через npm. Ці модулі використовуються для розширення функціоналу додатку, і вони інстальовані в контейнері в процесі побудови образу.

package-lock.json і package.json — ці файли є ключовими для управління залежностями Node.js. package.json зберігає інформацію про проект та перелік залежностей, потрібних для його роботи. package-lock.json гарантує, що будуть встановлені точні версії залежностей, що забезпечує консистентність середовища, рисунок 2.6.

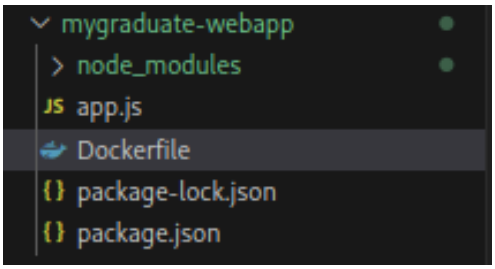


Рисунок 2.6 — Інфраструктура сайту

У контексті Docker, під час побудови образу контейнера, залежності встановлюються з використанням npm install, що базується на інформації з package.json. Цей процес забезпечує, що всі необхідні модулі встановлені та доступні в контейнері, рисунок 2.7.

Процес побудови образу контейнера реалізується за допомогою команд Dockerfile, які копіюють необхідні файли в контейнер, встановлюють залежності та конфігурують середовище для запуску додатку. Кінцевий образ контейнера містить все необхідне для запуску веб-сайту, включаючи додаток, його залежності та конфігурацію.

Рисунок 2.7 — Конфігурація package.json.

Для створення Docker контейнера для веб-сайту було використано наступний Dockerfile, рисунок 2.8:

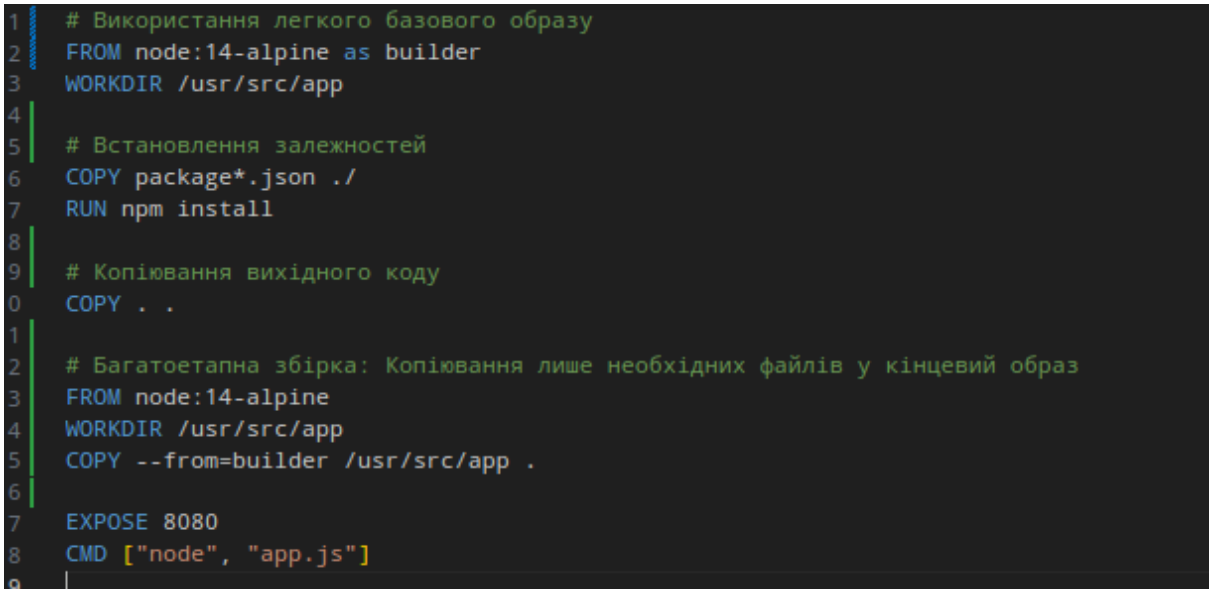


Рисунок 2.8 — Конфігурація Dockerfile.

FROM: вказує, на якому базовому образі буде побудований контейнер. У цьому випадку базовим образом є образ node:14. Цей образ забезпечує базову платформу для запуску веб-сайту на Node.js 14.

WORKDIR: задає робочий каталог для контейнера. У цьому випадку робочий каталог - це /usr/src/app. Цей каталог є місцем, де будуть зберігатися всі файли веб-сайту.

COPY: копіює файли або каталоги з локальної файлової системи в контейнер. У цьому випадку інструкції COPY копіюють файли package.json і package-lock.json з локальної файлової системи в контейнер. Ці файли містять інформацію про залежності веб-сайту.

RUN: виконує команду в контейнері. У цьому випадку інструкція RUN виконує команду npm install, яка інсталює всі залежності веб-сайту. Це необхідно для того, щоб веб-сайт міг запускатися в контейнері.

COPY: копіює всі інші файли або каталоги з локальної файлової системи в контейнер. У цьому випадку інструкція COPY копіює всі інші файли та каталоги з локальної файлової системи в контейнер. Це включає всі файли веб-сайту, такі як код, дані та файли конфігурації.

EXPOSE: вказує порти, які будуть відкриті в контейнері. У цьому випадку інструкція EXPOSE вказує, що порт 8080 буде відкритий в контейнері. Цей порт буде використовуватися для доступу до веб-сайту.

CMD: вказує команду, яка буде виконуватися в контейнері при запуску. У цьому випадку інструкція CMD вказує, що команда node app.js буде виконуватися в контейнері при запуску. Це запустить веб-сайт на порту 8080.

Dockerfile був створений для того, щоб забезпечити швидке та просте розгортання веб-сайту в хмарі. Він забезпечує всі необхідні кроки для створення контейнера, який містить всі файли та залежності веб-сайту.

2.4.1 Оптимізація розміру та продуктивності контейнерів

У процесі оптимізації Dockerfile для веб-додатку на Node.js, було внесено кілька ключових змін, спрямованих на зменшення розміру образу та підвищення продуктивності. Перш за все, замість стандартного образу node:14, використано більш легку версію node:14-alpine. Alpine Linux

відрізняється малим розміром та містить лише необхідні пакети, що знижує загальний розмір образу.

Для оптимального використання кешу Docker, команди копіювання та встановлення залежностей були розділені на окремі шари. Таким чином, будь-які зміни вихідного коду не впливають на шар залежностей npm, що зменшує час збірки при повторних збірках.

Крім того, було впроваджено багатоетапну збірку, де використовується проміжний контейнер для встановлення залежностей, а в кінцевий образ копіюються лише необхідні файли. Це значно зменшує кінцевий розмір образу, залишаючи лише те, що необхідно для роботи додатку.

2.4.2 Використання Docker в Kubernetes

Для розгортання веб-сайту в Kubernetes ми будемо використовувати Docker образ, створений на основі Dockerfile. Для цього ми будемо використовувати Helm — це менеджер пакетів для Kubernetes, який використовується для упаковки, конфігурування та розгортання додатків на Kubernetes. "Helm chart" — це набір файлів, що описують пов'язані ресурси Kubernetes, необхідні для розгортання додатка або сервісу в Kubernetes. Стандартні ресурси, які зазвичай включені в Helm chart, включають:

— Chart.yaml — основний файл, який містить метадані Helm chart, включаючи назву, версію та опис chart;

— Values.yaml — файл, який містить конфігураційні параметри, які можна налаштувати під час розгортання, ці значення можуть бути перезаписані користувачем під час розгортання;

— Templates — каталог, що містить шаблони ресурсів Kubernetes, наприклад, Deployments, Services, Ingresses, ConfigMaps, і інші. Ці шаблони використовують значення з файлу values.yaml для створення ресурсів Kubernetes;

— _helpers.tpl — файл, який містить шаблони та функції, які можна повторно використовувати в шаблонах chart;

— Dependencies — Helm charts можуть містити залежності від інших charts, які визначені в Chart.yaml файлі або в окремому requirements.yaml файлі;

— Tests — каталог для тестів Helm chart, який може містити тести для перевірки, чи правильно працює chart після його встановлення.

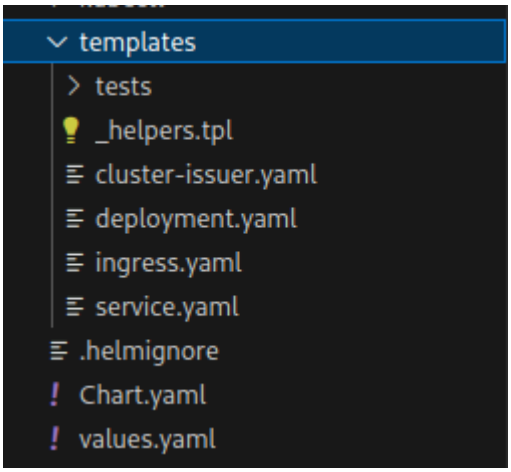


Рисунок 2.10 — Структура helm чарту.

Для визначення Docker образу для веб-сайту в Helm чарті використовується наступна інструкція з файлу values.yaml, який містить в собі всі параметри для helm чартів:

```
image:
  repository: gcr.io/ascendant-talon-405710/mygraduate-webapp
  tag: "latest"
  pullPolicy: IfNotPresent
```

Рисунок 2.11 — Інструкція для визачення Docker образу

Ця інструкція вказує, що Docker образ веб-сайту буде знаходитися в репозиторії gcr.io/ascendant-talon-405710/mygraduate-webapp. Версія образу буде "latest", а політика завантаження буде IfNotPresent.

2.5 Безпека в Kubernetes

Безпека є важливим аспектом будь-якої хмарної інфраструктури, і Kubernetes не є винятком. Kubernetes надає широкий спектр засобів безпеки, які можна використовувати для захисту ваших додатків і даних.

Kubernetes ґрунтується на наступних принципах безпеки:

— централізоване управління яке забезпечує управління всіма ресурсами, включаючи безпеку, що полегшує управління і забезпечує більшу прозорість;

— ізоляція контейнерів один від одного, що допомагає захистити один контейнер від іншого;

— різні механізми контролю доступу, які можна використовувати для обмеження доступу до ресурсів.

Контроль доступу (Access Control, AC) є одним з найважливіших аспектів безпеки в Kubernetes. Kubernetes надає різні механізми контролю доступу, які можна використовувати для обмеження доступу до ресурсів.

Одним із найпростіших способів контролю доступу є використання ролей і дозволів (Role-Based Access Control, RBAC). RBAC дозволяє визначати ролі, які надають певні дозволи. Потім можна надавати ці ролі користувачам або групам користувачів.

Наприклад, можна створити роль із дозволом на створення і запуск контейнерів. Потім можна надати цю роль користувачам, які відповідають за розгортання додатків.

Іншим механізмом контролю доступу в Kubernetes є мережі політики (Network Policies). Мережні політики дозволяють визначати правила, які контролюють трафік між контейнерами.

Наприклад, можна створити мережну політику, яка забороняє весь вхідний трафік до контейнерів. Це допоможе захистити контейнери від несанкціонованого доступу.

Kubernetes також надає інші засоби безпеки, які можна використовувати для захисту ваших додатків і даних. До цих засобів належать.

Сканування безпеки, яке можна використовувати для сканування контейнерів на наявність вразливостей безпеки. Автоматичне відновлення, що дає можливість автоматичного відновлення контейнерів у разі збою.

Kubernetes можна використовувати для моніторингу безпеки кластера.

2.3.1 Roles та Rolebindings

У проекті було приділено особливу увагу забезпеченню безпеки як на рівні додатків, так і на рівні самого кластера. Ось ключові практики та стратегії, які були використані для цього:

Role та RoleBindings — це потужні інструменти управління доступом, які забезпечують детальне контролювання того, хто (які облікові записи користувачів або процеси) і що можуть робити в рамках кластера.

Role — це об'єкт в Kubernetes, який визначає дозволи на виконання дій для ресурсів в певному namespace. Role містить правила, що вказують, які операції можуть бути виконані на яких ресурсах.

RoleBinding — це спосіб призначення ролей конкретним користувачам, групам або сервісним обліковим записам, воно встановлює "зв'язок" між роллю та користувачами, яким ця роль надається.

Спочатку створюється Role, яка визначає дозволи, рисунок 2.12.

```
ygraduate > kubectl > ! rolebinding.yaml
1   apiVersion: rbac.authorization.k8s.io/v1
2   kind: Role
3   metadata:
4     namespace: default
5     name: service-role
6   rules:
7   - apiGroups: [""]
8     resources: ["pods"]
9     verbs: ["get", "list"]
```

Рисунок 2.12 — Конфігурація role.yaml

Потім створюється RoleBinding, яка призначає цю роль конкретному користувачу або сервісному обліковому запису, рисунок 2.13.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: nychvadim
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Рисунок 2.13 — Конфігурація rolebinding.yaml

Використовуючи значення наведені вище, створено необхідні Role та RoleBindings у проекті та застосовано їх у кластері використовуючи команду kubectl apply -f rolebingind.yaml.

Ці компоненти дозволять забезпечити більш безпечне та контрольоване середовище для Kubernetes проекту, обмеживши доступ тільки до необхідних ресурсів для кожного користувача або процесу.

2.5.2 Управління секретами

Secrets у Kubernetes використовуються для зберігання та управління чутливою інформацією, такою як паролі, OAuth токени, ssh ключі.

Використання Kubernetes Secrets чудово підходить для зберігання чутливої інформації, такої як паролі, токени доступу, приватні ключі. Секрети були інтегровані у додатки без необхідності зберігання їх у коді або конфігураційних файлах.

Секрети створюються за допомогою команди kubectl create secret або через YAML файл. Secret, для зберігання пароля до бази даних, рисунок 2.14:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-password
type: Opaque
data:
  password: [MTIzNDVLVUJFUk5FVEVT]
```

Рисунок 2.14 — Конфігурація secret.yaml

В даному випадку type: Opaque означає, що пароль має бути зашифрований у форматі base64

У конфігурації деплойменту Kubernetes, секрети можуть бути використані для встановлення змінних середовища або як файли у подах, .

Предача пароля у додаток подано на рисунку 2.15.

```yaml
15          valueFrom:
16            secretKeyRef:
17              name: db-password
18              key: password
19
```

Рисунок 2.15 — Передача значень secret.yaml у values.yaml

Оскільки в проекті використовується helm то в даному випадку значення передаються через файл values.yaml

2.5.3 Мережева безпека

Мережева передбачає, що мають бути налаштовані Network Policies для обмеження мережевого доступу між подами в кластері, для того щоб запобігти несанкціонованому доступу.

Network Policies дають можливість визначати мережевий доступ на рівні подів.

Обмеження доступу до бази даних тільки з певних подів, рисунок 2.16.

Після створення, Network Policy застосовується в Kubernetes, обмежуючи мережевий трафік відповідно до заданих правил.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-network-policy
spec:
  podSelector:
    matchLabels:
      role: database
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: web
```

Рисунок 2.16 — Конфігурація networkpolicy.yaml

Це дозволяє ефективно контролювати, які сервіси мають доступ до ресурсів всередині кластера, значно підвищуючи загальний рівень безпеки.

Використання Secrets та Network Policies, Role в комбінацією з RoleBindings забезпечує контрольований доступ до чутливих даних та мережевих ресурсів, підвищуючи безпеку застосунку та даних, якими він керує. Ці інструменти є ключовими для створення надійного та безпечного середовища в рамках Kubernetes кластера.

2.5.4 Захист від DDoS атак

Захист від DDoS атак відбувається наступним чином.

Реалізація лімітів запитів: становлення лімітів на кількість запитів, які можуть бути оброблені подами за певний час, для запобігання перевантаженню системи.

Використання сервісів зовнішнього балансування навантаження: інтеграція з рішеннями зовнішнього балансування навантаження, які можуть виявляти та мінімізувати вплив DDoS-атак.

Шифрування даних: застосування шифрування для захисту даних, що передаються, від несанкціонованого доступу та зловмисників.

Ці заходи були впроваджені для підвищення стійкості інфраструктури Kubernetes до зовнішніх загроз і забезпечення безпечного середовища для роботи веб-додатків.

В Kubernetes для обмеження кількості запитів, які обробляються подами, можна використовувати ресурсні квоти (Resource Quotas) та обмеження (Limit Ranges). Наприклад, рисунок 2.17:

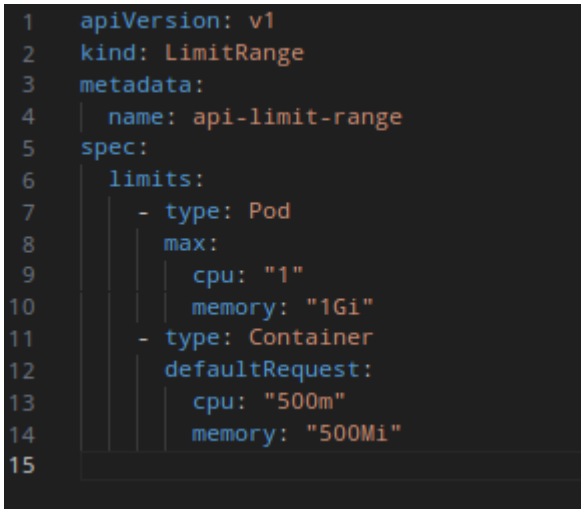


Рисунок 2.17 — Конфігурація limitrange.yaml

Цей код встановлює максимальні та стандартні обмеження ресурсів для подів та контейнерів відповідно.

3 РОЗГОРТАННЯ ТА УПРАВЛІННЯ ВЕБ-САЙТОМ

3.1 Розгортання веб-додатку в Kubernetes

Для розгортання веб-додатку в Kubernetes було використано Helm, менеджер пакетів для Kubernetes. Helm дозволяє легко розгортати та керувати складними додатками з одним YAML-файлом.

Для створення Helm чарта було створено каталог my-graduate-webapp, в якому були створені наступні файли:

Chart.yaml — файл, який визначає основні параметри чарта, такі як назва, версія та опис.

Цей фрагмент визначає основні метадані Helm chart, який названий "mygraduate". Використовуючи apiVersion v2, він підтверджує, що цей chart сумісний з Helm 3. Опис "A Helm chart for Kubernetes" дає загальне розуміння призначення цього chart, яке полягає у розгортанні додатку на Kubernetes. Вказаний тип application підкреслює, що chart є призначений для розгортання додатків, а не бібліотечних функцій. Версія chart 0.1.0 та версія додатка latest вказують на початкову стадію розробки цього chart і останню доступну версію додатку відповідно, рисунок 3.1.

```
1  apiVersion: v2
2  name: mygraduate
3  description: A Helm chart for Kubernetes
4
5  type: application
6
7  version: 0.1.0
8
9  appVersion: "latest"
```

Рисунок 3.1 — Конфігурація Chart.yaml

Далі наведено service.yaml: файл, який визначає ресурс Service для веб-додатку.

Цей фрагмент визначає Service у Kubernetes в рамках Helm chart "mygraduate". Він починається з зазначення версії API та типу ресурсу, далі йде назва сервісу, яка генерується динамічно за допомогою шаблону Helm. Метадані включають мітки (labels), також задані за допомогою шаблону. В специфікації (spec) визначається тип сервісу, який може бути налаштований через значення в values.yaml. Порти для сервісу налаштовуються для приймання трафіку, вказуючи порт на якому слухає сервіс, цільовий порт у подах, протокол (TCP) та назву порту. Селектор використовується для зв'язування сервісу з певними подами, визначаючи, до яких подів буде направлятися трафік.

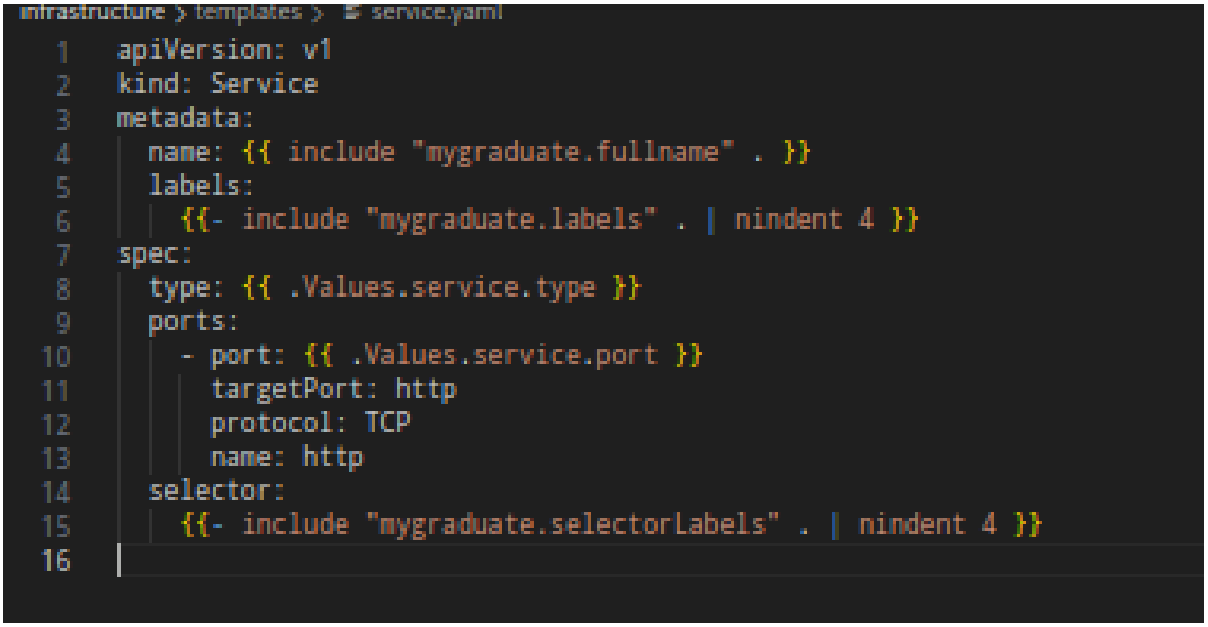


Рисунок 3.2 — Конфігурація service.yaml

Нижче продемонстровано Deployment.yaml: файл, який визначає ресурс Deployment для веб-додатку.

Цей фрагмент описує Deployment для Kubernetes, який є частиною Helm chart "mygraduate". Він використовується для опису того, як додаток має бути розгорнутий і керований в середовищі Kubernetes.

Deployment починається з вказівки версії API та типу ресурсу. Метадані включають назву Deployment, що генерується динамічно, та мітки, задані за допомогою шаблону Helm.

В специфікації (spec) визначається кількість реплік (копій) поду, що має бути створено, якщо не включено автоматичне масштабування. Селектор matchLabels використовується для ідентифікації подів, які мають бути контрольовані Deployment.

Далі йде опис шаблону поду (template), включаючи анотації та мітки, використовуючи динамічні значення. В розділі spec поду вказуються imagePullSecrets для доступу до приватних реєстрів контейнерів, ім'я облікового запису служби (serviceAccountName), контекст безпеки, інформація про контейнер (включаючи образ, політику завантаження образу, порти, проби готовності та живучості, ресурси та змінні оточення).

Для контейнера налаштовані змінні оточення для з'єднання з базою даних, з використанням значень, що зберігаються у секретах Kubernetes.

Також передбачені налаштування nodeSelector, affinity, та tolerations для контролю розміщення подів на вузлах кластера, рисунок 3.3.

Наступний файл ingress.yaml: файл, який визначає ресурс Ingress для веб-додатку.

Цей фрагмент включає умову, яка активує конфігурацію Ingress у випадку, якщо в values.yaml файлі відповідний параметр встановлено як enabled. Він описує ресурс Ingress для Kubernetes, використовуючи Helm chart "mygraduate", який управляє доступом зовнішніх користувачів до сервісів всередині кластера.

Він починається з визначення apiVersion та виду ресурсу, після чого йде назва Ingress, що генерується з шаблону Helm. Мітки та анотації для Ingress також визначаються динамічно з використанням шаблонів.

Специфікація Ingress включає ім'я класу Ingress та набір правил. Ці правила визначають, як запити до певних хостів та шляхів будуть

оброблятися. Кожен шлях зв'язується з певним сервісом у кластері та відповідним портом цього сервісу.

infrastructure › templates › deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "mygraduate.fullname" . }}
  labels:
    {{- include "mygraduate.labels" . | nindent 4 }}
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicaCount }}
  {{- end }}
  selector:
    matchLabels:
      {{- include "mygraduate.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      {{- with .Values.podAnnotations }}
      annotations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      labels:
        {{- include "mygraduate.selectorLabels" . | nindent 8 }}
    spec:
      {{- with .Values.imagePullSecrets }}
      imagePullSecrets:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      serviceAccountName: {{ include "mygraduate.serviceAccountName" . }}
      securityContext:
        {{- toYaml .Values.podSecurityContext | nindent 8 }}
      containers:
        - name: {{ .Chart.Name }}
          securityContext:
            {{- toYaml .Values.securityContext | nindent 12 }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: {{ .Values.service.port }}
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
          readinessProbe:
            httpGet:
              path: /
              port: http
          resources:
            {{- toYaml .Values.resources | nindent 12 }}
          env:
            - name: DATABASE_HOST
              value: {{ .Values.database.host }}
            - name: DATABASE_USER
              value: {{ .Values.database.user }}
            - name: DATABASE_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: {{ .Values.database.secret.name }}
                  key: {{ .Values.database.secret.key }}
      {{- with .Values.nodeSelector }}
      nodeSelector:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.affinity }}
      affinity:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      {{- with .Values.tolerations }}
      tolerations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
```

Рисунок 3.3 — Конфігурація deployment.yaml

Якщо включено TLS, для Ingress також визначаються налаштування TLS, включаючи хости та секрети, які містять сертифікати SSL.

Цей фрагмент демонструє, як Helm дозволяє гнучко керувати розгортанням складних конфігурацій, таких як Ingress, з можливістю легко включати або виключати певні компоненти в залежності від потреб розгортання, рисунок 3.4.

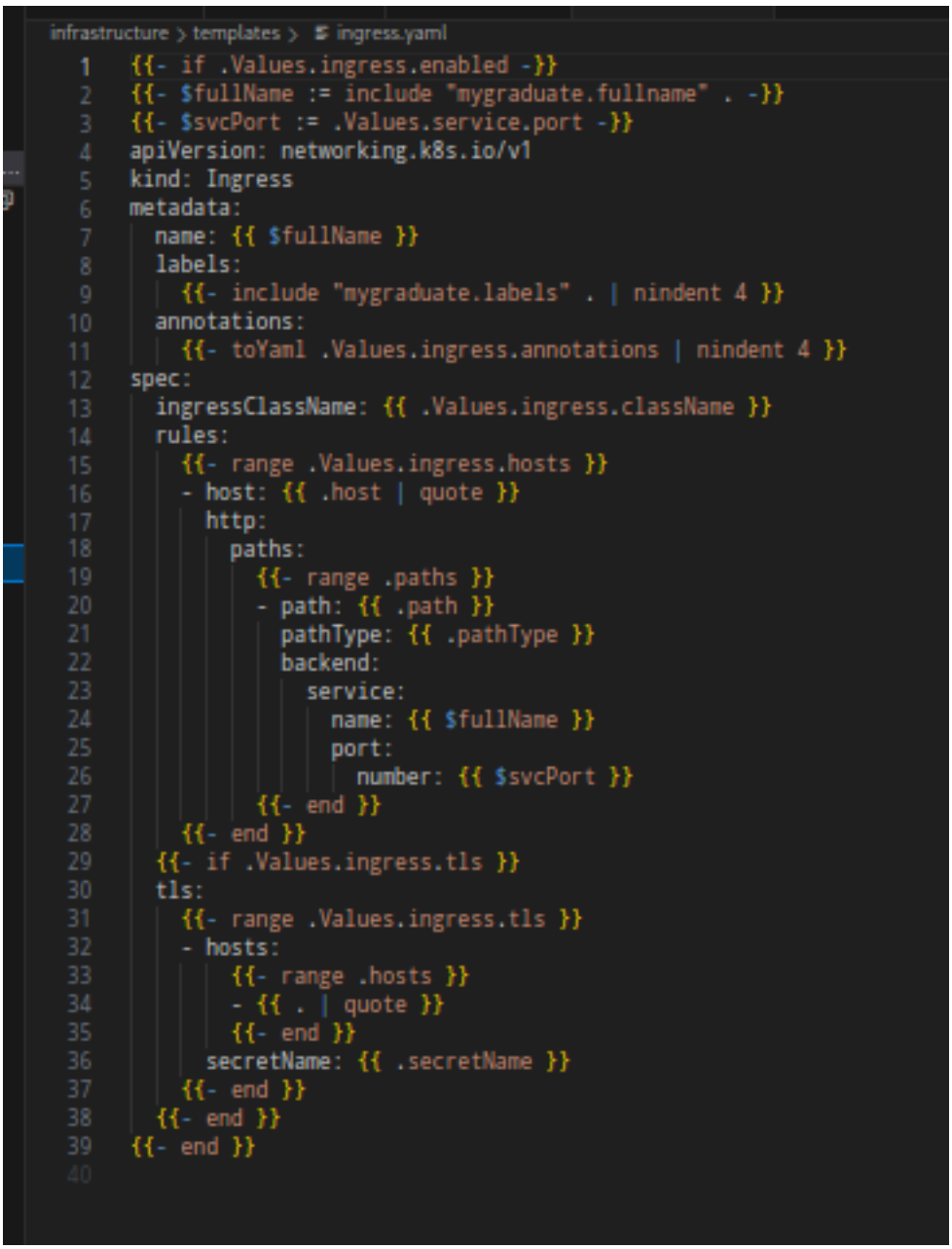


Рисунок 3.4 — Конфігурація ingress.yaml

Файл values.yaml: файл, який визначає значення параметрів чарта.

Цей фрагмент визначає значення конфігурації для Helm chart, які використовуються для управління розгортанням та налаштуванням додатку в Kubernetes:

— replicaCount — 1 вказує на кількість реплік подів для Deployment;

— секція image визначає налаштування образу контейнера, де знаходиться образ (repository), яку версію використовувати (tag), та політику завантаження образу (pullPolicy);

— imagePullSecrets — пустий масив, означає, що секрети для завантаження образів не використовуються;

— nameOverride та fullnameOverride дозволяють перевизначити стандартні назви ресурсів;

— секція serviceAccount визначає, чи потрібно створювати новий обліковий запис сервісу, його анотації та назву;

— database містить налаштування для підключення до бази даних, включаючи хост, користувача, та інформацію про секрети для доступу до бази даних;

— podAnnotations, podSecurityContext, securityContext — це розділи для налаштування додаткових параметрів безпеки та анотацій для подів.

— service визначає тип сервісу Kubernetes та порт, на якому він слухає;

— ingress містить конфігурацію для Ingress, включаючи його активацію, клас, анотації, правила для хостів і шляхів, та налаштування TLS;

— resources визначає обмеження та запити на ресурси для подів, такі як CPU та пам'ять;

— autoscaling визначає параметри для автоматичного масштабування, включаючи мінімальну та максимальну кількість реплік, та цільове використання CPU;

— nodeSelector, tolerations, affinity — ці параметри дозволяють детально налаштувати розміщення подів на вузлах кластера, рисунок 3.5.

```yaml
replicaCount: 1
image:
  repository: gcr.io/ascendant-talon-405710/mygraduate-webapp
  tag: "latest"
  pullPolicy: IfNotPresent
imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""
serviceAccount:
  create: false
  annotations: {}
  name: ""
database:
  host: 10.116.13.78
  user: nychvadim
  secret:
    name: "db-password" # Ім'я секрету, яке містить пароль
    key: "password"    # Ключ у секреті, де зберігається пароль
podAnnotations: {}
podSecurityContext: {}
securityContext: {}
service:
  type: ClusterIP
  port: 8080
ingress:
  enabled: true
  className: "nginx"
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    kubernetes.io/ingress.class: "nginx"
  hosts:
    - host: "mygraduate.theraven.tech"
      paths:
        - path: /
          pathType: ImplementationSpecific
  tls:
    - secretName: "graduate-tls"
      hosts:
        - "mygraduate.theraven.tech"
resources: {}
  limits:
    cpu: 100m
    memory: 128Mi
  requests:
    cpu: 100m
    memory: 128Mi
autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80
nodeSelector: {}
tolerations: []
affinity: {}
```

Рисунок 3.5 — Файл values.yaml

Цей файл values.yaml є ключовим для конфігурації Helm chart, дозволяючи користувачам налаштовувати розгортання додатків відповідно до своїх потреб, рисунок 3.6.

Для встановлення Helm чарта було виконано наступну команду: helm install my-graduate-webapp my-graduate -f values.yaml. Ця команда завантажила чарт в кластер і розгорнула його, рисунок 3.7.

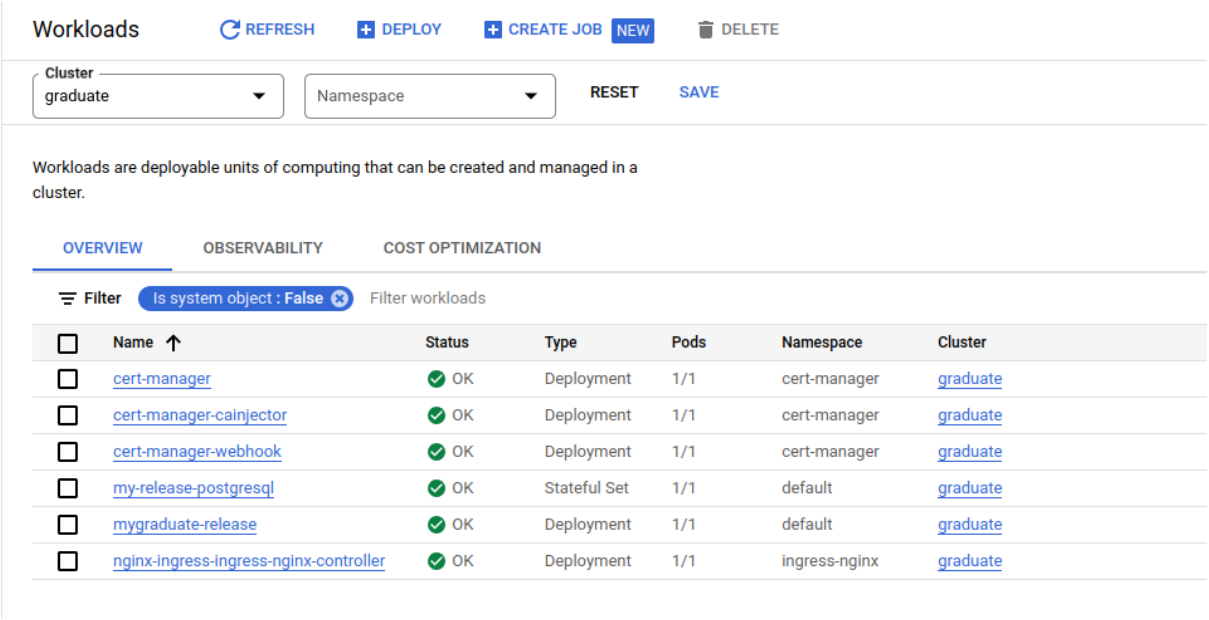


Рисунок 3.7 — Список workloads в Kubernetes

Для налаштування мережі та балансування навантаження було використано наступні налаштування:

Для веб-додатку було використано сервіс типу ClusterIP. Це означає, що сервіс доступний тільки з інших подов в кластері Kubernetes.

Порт 8080 веб-додатку було опубліковано на порт 80 сервісу, рисунок 3.8.

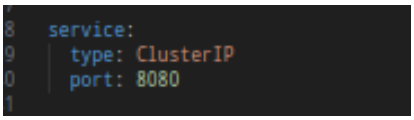


Рисунок 3.8 —Налаштування для service.yaml

Для забезпечення зовнішнього доступу до веб-додатку було налаштовано Ingress. Ingress був налаштований для використання хоста mygraduate.theraven.tech, рисунок3.9.

```yaml
ingress:
  enabled: true
  className: "nginx"
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    kubernetes.io/ingress.class: "nginx"
  hosts:
    - host: "mygraduate.theraven.tech"
      paths:
        - path: /
          pathType: ImplementationSpecific
  tls:
    - secretName: "graduate-tls"
      hosts:
        - "mygraduate.theraven.tech"
```

Рисунок 3.9 — Налаштування для ingress.yaml

Для автоматичного налаштування HTTPS Ingress використовував cert-manager cluster-issuer, рисунок 3.10.

```yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: nychvadim@gmail.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx

```

Рисунок 3.10 — Конфігурація cluster-issuer.yaml

Крім основних налаштувань, було також виконано наступні налаштування:

Для веб-додатку було налаштовано liveness та readiness probes. Ці probes допомагають Kubernetes відстежувати стан веб-додатку та перезапускати його при необхідності.

Для налаштування Helm чарта для різних середовищ було використано YAML-шаблони.

3.2 Інтеграція з базою даних

3.2.1 Налаштування бази даних

В контексті розробки веб-додатків, PostgreSQL є одним із провідних виборів для систем управління базами даних, порівняно з іншими популярними рішеннями, такими як MySQL або MongoDB. Відмінність PostgreSQL полягає у його розширеній підтримці стандартів SQL, надійності та підтримці складних операцій із даними. Він підтримує індексування JSON, що робить його зручним для використання у сучасних веб-додатках, які використовують JSON для обміну даними. Порівняно з MySQL, PostgreSQL пропонує більш гнучкі та потужні можливості управління транзакціями та кращу підтримку паралельних запитів.

У порівнянні з іншими системами управління базами даних, PostgreSQL пропонує міцні механізми для забезпечення високої доступності. Його підтримка реплікації даних, яка може бути налаштована як у синхронному, так і в асинхронному режимах, перевершує аналогічні можливості в MySQL та інших СУБД. Це дозволяє створювати надійні та відмовостійкі системи. Для резервного копіювання, PostgreSQL пропонує гнучкі інструменти, такі як pg_dump і pg_basebackup, що забезпечують ефективне та надійне резервне копіювання даних. В порівнянні з MongoDB, PostgreSQL пропонує більш традиційні, але перевірені підходи до резервного копіювання, що може бути важливим для критичних до втрати даних додатків.

Обравши PostgreSQL, розробники можуть використовувати його розширені можливості для створення ефективних, надійних та безпечних веб-додатків, які вимагають високого рівня взаємодії з даними та їх обробки.

3.2.2 .Контейнеризація бази даних

База даних може бути розгорнута безпосередньо у Kubernetes як окремий под або через зовнішні хмарні сервіси. Якщо БД розгортається в

Kubernetes, використовується образ Docker, наприклад, офіційний image PostgreSQL з Docker Hub, рисунок 3.11.

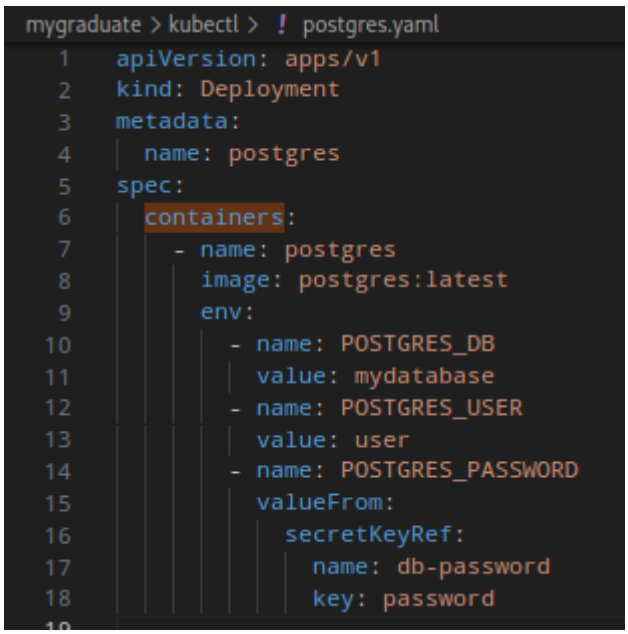


Рисунок 3.11 — Deployment.yaml для бази даних PostgreSQL.

У Deployment.yaml, змінні середовища для підключення до бази даних (DATABASE_HOST, DATABASE_USER, DATABASE_PASSWORD) задаються в специфікації контейнера, рисунок 3.12.

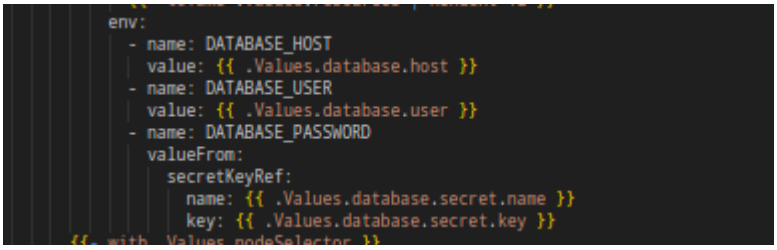


Рисунок 3.12 — Конфіруація бази даних в Deployment.yaml для helm чарта.

Ці змінні використовують значення, визначені у values.yaml, та забезпечують безпечне зберігання пароля бази даних через Kubernetes Secret, рисунок 3.13.

```yaml
database:
  host: 10.116.13.78
  user: nychvadim
  secret:
    name: "db-password" # Ім'я секрету, яке містить пароль
    key: "password"    # Ключ у секреті, де зберігається пароль
```

Рисунок — 3.13 Конфігурації бази даних у файлі values.yaml

Далі відбувається інтеграція з secret.yaml. DATABASE_PASSWORD отримується з секрету, що забезпечує безпечне зберігання чутливих даних, рисунок 3.14.

```
infrastructure > kubectl > ! sercet.yaml
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: db-password
5  type: Opaque
6  data:
7    password: [MTIzNDVLVUJFUk5FVEVT]
```

Рисунок — 3.14 Файл Secret.yaml

Застосування Kubernetes Secret (db-password) дозволяє уникнути необхідності жорстко закодовувати чутливі дані в образах або конфігураціях. Service у Kubernetes є ключовою абстракцією, яка використовується для експозиції додатків, запущених у подах, як один консистентний адресований ресурс. У випадку з my-release-postgresql, Service виконує кілька важливих функцій, рисунок 3.15

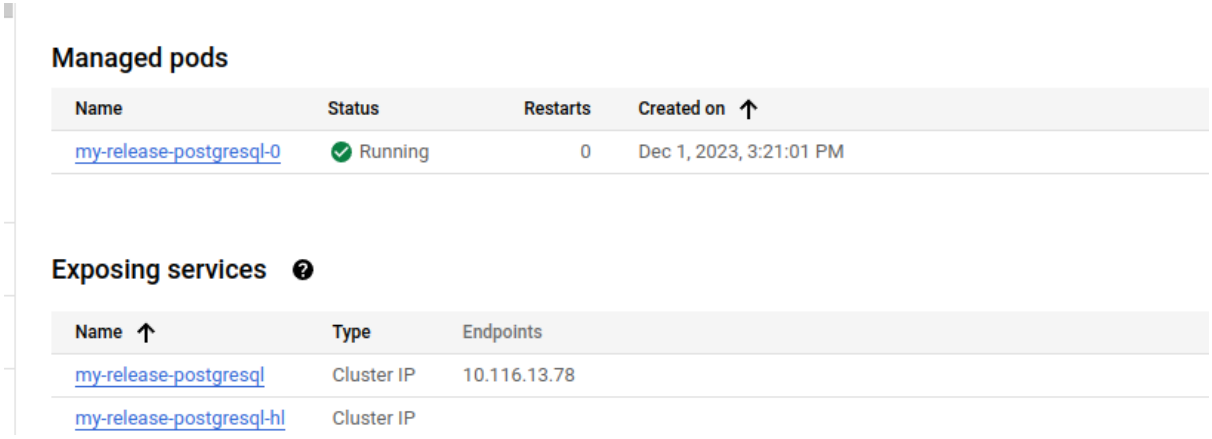


Рисунок 3.15 — Services для PostreSQL

Service забезпечує стабільну IP-адресу та DNS ім'я, через які інші сервіси у кластері можуть підключатися до бази даних PostgreSQL. Це означає, що незалежно від того, де фізично розміщений под з PostgreSQL, його можна знайти за цією сталою адресою.

Якщо база даних PostgreSQL працює у декількох подах для реплікації або масштабування, Service здійснює балансування навантаження між цими подами. Таким чином, запити до бази

3.3 Автоматизація процесів через CI/CD

Інтеграція Kubernetes з системами неперервної інтеграції (CI) та неперервного розгортання (CD) є ключовим компонентом в розробці та управлінні сучасними додатками. Це забезпечує автоматизацію численних процесів, пов'язаних із розгортанням, тестуванням та управлінням програмними рішеннями.

Тісна інтеграція Kubernetes з системами контролю версій, такими як GitHub, дозволяє автоматизувати процеси розгортання при кожному коміті або зміні у визначених гілках. Автоматизовані процеси запускаються при змінах у гілці develop, що забезпечує неперервність процесів розробки та розгортання. Ця інтеграція сприяє швидкому впровадженню нових функцій та виправленню помилок, підтримуючи високу якість та стабільність додатку.

В даному випадку CI/CD pipeline, налаштований на GitHub Actions, який використовує можливості Kubernetes для автоматизації критичних аспектів процесу розгортання, включаючи збірку, публікацію та оновлення додатків. Це забезпечує гнучке та ефективне управління життєвим циклом програмного забезпечення.

Автоматизація цих процесів дозволяє:

— збільшити швидкість розробки — CI/CD дозволяє швидко і легко інтегрувати зміни від різних розробників, що призводить до більш швидкого випуску нових функцій та виправлення помилок.

— покращити якість — CI/CD дозволяє автоматизувати тестування, що допомагає виявляти помилки на ранніх етапах розробки;

— зменшити ризик — CI/CD дозволяє розгортати програмне забезпечення в середовище виробництва з меншим ризиком збоїв.

CI/CD-процес зазвичай складається з наступних етапів:

— компіляція — це процес перетворення коду в виконуваний файл;

— тестування — це процес перевірки того, що код працює правильно;

— інтеграція — це процес об'єднання змін від різних розробників в єдину робочу програму;

— розгортання — це процес перенесення робочої програми в середовище виробництва.

Ці етапи можуть бути автоматизовані за допомогою різних інструментів та технологій.

В даному випадку цей CI/CD-конвеєр використовується для автоматизації збірки, тестування, публікації та розгортання веб-додатку "mygraduate-webapp". Він написаний на основі GitHub Actions. Процес його функціонування описаний нище.

Тригери: Push у гілку develop та ручний запуск через "workflow_dispatch".

Входи: tag: Тег для Docker-контейнера (за замовчуванням "latest");

Перемінні оточення: IMAGE_NAME_APP: Ім'я образу Docker-контейнера додатка ("gcr.io/ascendant-talon-405710/mygraduate-webapp").

Етапи:

— виконує checkout репозиторію;

— збірка додатка за допомогою gradlew clean bootJar;

— будує Docker-образ додатка з тегом, зазначеним у вході tag;

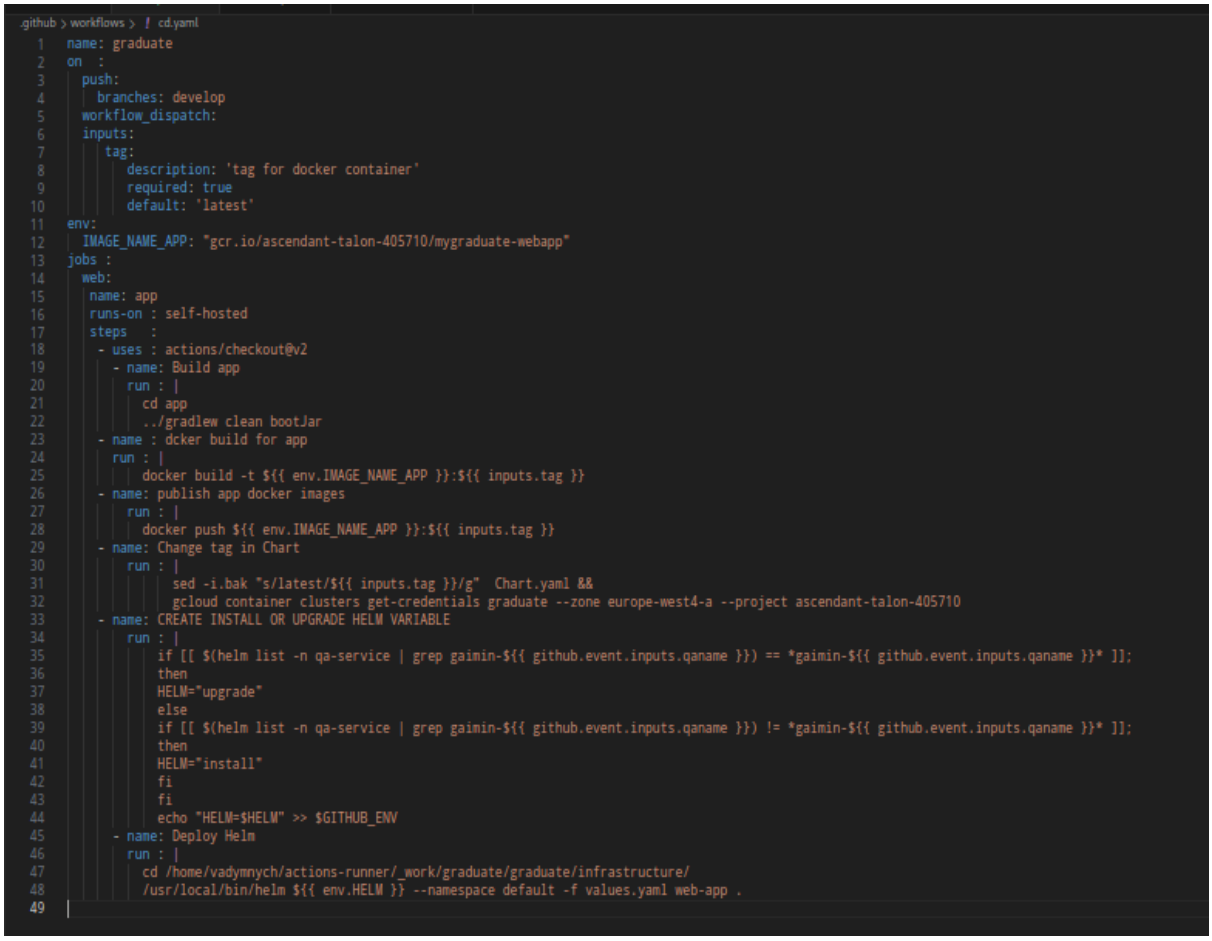
— публікує Docker-образ у реєстр;

— змінює тег у файлі Chart.yaml на зазначений у вході tag;

— отримує облікові дані для кластера Kubernetes;

— визначає, чи потрібно встановлювати новий чарт Helm, чи оновлювати існуючий;

— розгортає чарт Helm у просторі імені default, рисунок 3.16.



```
.github > workflows > ! cd.yaml
name: graduate
on  :
  push:
    branches: develop
  workflow_dispatch:
  inputs:
    tag:
      description: 'tag for docker container'
      required: true
      default: 'latest'
env:
  IMAGE_NAME_APP: "gcr.io/ascendant-talon-405710/mygraduate-webapp"
jobs :
  web:
   name: app
   runs-on : self-hosted
   steps   :
    - uses : actions/checkout@v2
      - name: Build app
        run : |
          cd app
          ../gradlew clean bootJar
    - name : dcker build for app
      run : |
          docker build -t ${{ env.IMAGE_NAME_APP }}:${{ inputs.tag }}
    - name: publish app docker images
        run : |
          docker push ${{ env.IMAGE_NAME_APP }}:${{ inputs.tag }}
    - name: Change tag in Chart
        run : |
            sed -i.bak "s/latest/${{ inputs.tag }}/g"  Chart.yaml &&
            gcloud container clusters get-credentials graduate --zone europe-west4-a --project ascendant-talon-405710
    - name: CREATE INSTALL OR UPGRADE HELM VARIABLE
        run : |
          if [[ $(helm list -n qa-service | grep gaimin-${{ github.event.inputs.qaname }}) == *gaimin-${{ github.event.inputs.qaname }}* ]];
          then
          HELM="upgrade"
          else
          if [[ $(helm list -n qa-service | grep gaimin-${{ github.event.inputs.qaname }}) != *gaimin-${{ github.event.inputs.qaname }}* ]];
          then
          HELM="install"
          fi
          fi
          echo "HELM=$HELM" >> $GITHUB_ENV
      - name: Deploy Helm
        run : |
          cd /home/vadymnych/actions-runner/_work/graduate/graduate/infrastructure/
          /usr/local/bin/helm ${{ env.HELM }} --namespace default -f values.yaml web-app .
```

Рисунок 3.16 — CI/CD Workflow, описаний на базі GithubActions

Результат запуску ci/cd, рисунок 3.17.

Після push в develop будь якого commit ми отримуємо автоматичне оновлення всіх даних та нову ревізію Helm чарта, рисунок 3.18.

3.4 Моніторинг та управління ресурсами

Моніторинг є важливою частиною будь-якої системи, яка допомагає виявляти проблеми на ранніх етапах та запобігати збоям.

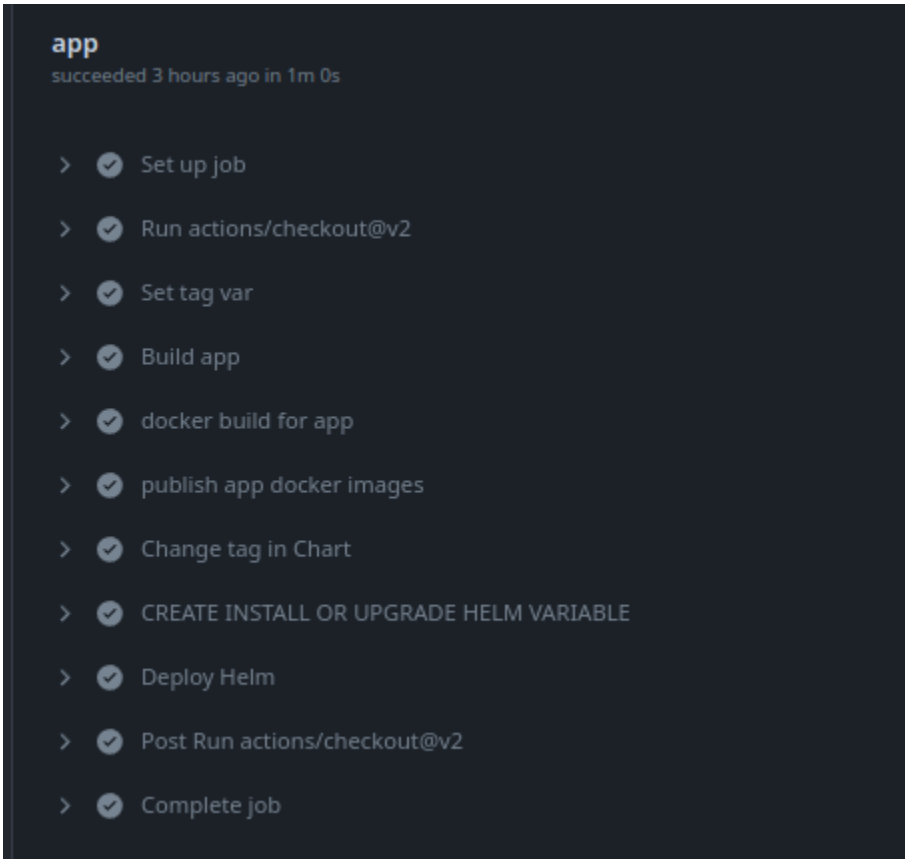


Рисунок 3.17 — Результат запуску ci/cd

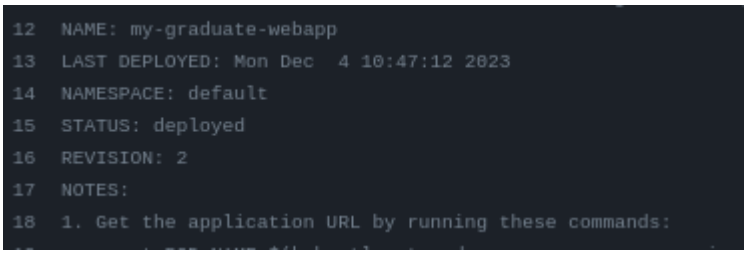


Рисунок 3.18 — Крок Deploy Helm

Kubernetes пропонує широкий спектр інструментів для моніторингу, включаючи:

— Metrics Server — забезпечує базові метрики для всіх об'єктів Kubernetes, таких як використання процесора, пам'яті та мережі;

— Prometheus — потужний інструмент моніторингу, який може збирати та зберігати метрики з різних джерел;

— Grafana — інструмент візуалізації даних, який дозволяє створювати графіки та діаграми на основі метрик.

Kubernetes дозволяє ефективно управляти ресурсами, такими як процесор, пам'ять та дисковий простір. Це робиться за допомогою таких механізмів, як:

— обмеження — дозволяють обмежити використання ресурсів для об'єктів Kubernetes;

— балансування навантаження — дозволяє розподілити навантаження між різними вузлами кластера Kubernetes;

— автомасштабування — дозволяє автоматично масштабувати кластер Kubernetes у відповідь на зміну навантаження.

Гібридний кластер Kubernetes (GKE) надає ряд способів перегляду логів. Одним із найпростіших способів є використання порталу Google Cloud Platform. Для цього перейдіть до розділу "Kubernetes Engine" і виберіть кластер, який ви хочете переглянути. У розділі "Логі" ви побачите список усіх логів, які були створені для цього кластера.

Щоб переглянути логи для конкретного об'єкта, ви можете використовувати команду kubectl logs. Наприклад, щоб переглянути логи для Pod-а з ім'ям "my-pod", виконайте команду:

kubectl logs my-pod, рисунок 3.19

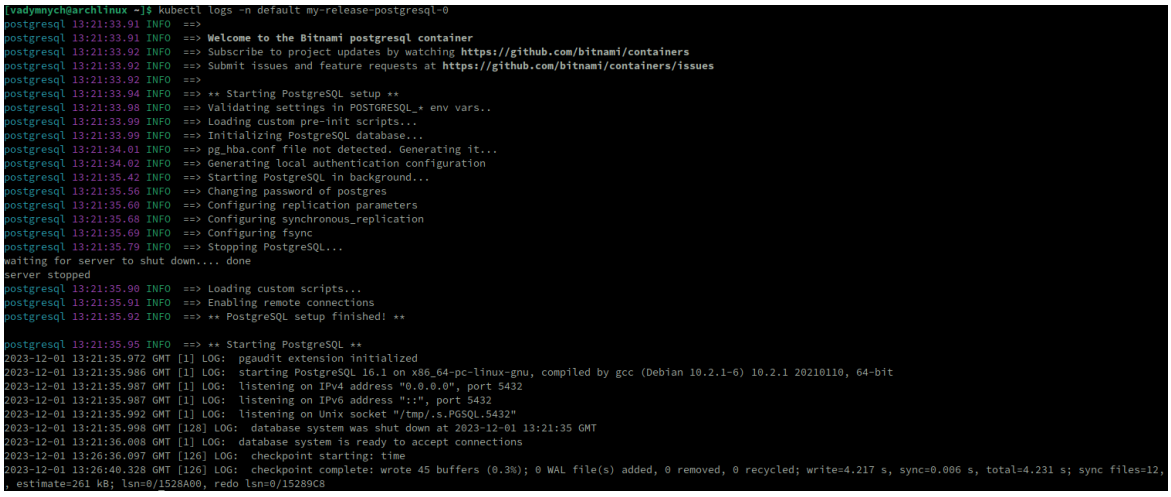


Рисунок 3.19 — Результат запуску kubectl logs postgresq-release

Також існує можливість переглянути логи для конкретного контейнера в Pod-і, використовуючи параметр -c. Наприклад, щоб переглянути логи для контейнера з ім'ям "my-container" у Pod-і з ім'ям "my-pod", виконайте команду: kubectl logs my-pod -c my-container

Щоб переглянути логи для всіх Pod-ів у кластері, ви можете використовувати команду kubectl get pods -o wide. Це виведе список усіх Pod-ів, включаючи їхні імена та ідентифікатори. Потім ви можете використовувати команду kubectl logs для перегляду логів для конкретного Pod-а.

Крімт того Google Cloud Platform, відображає логи Kubernetes кластера завдяки графічному інтерфейсу, рисунок 3.20.

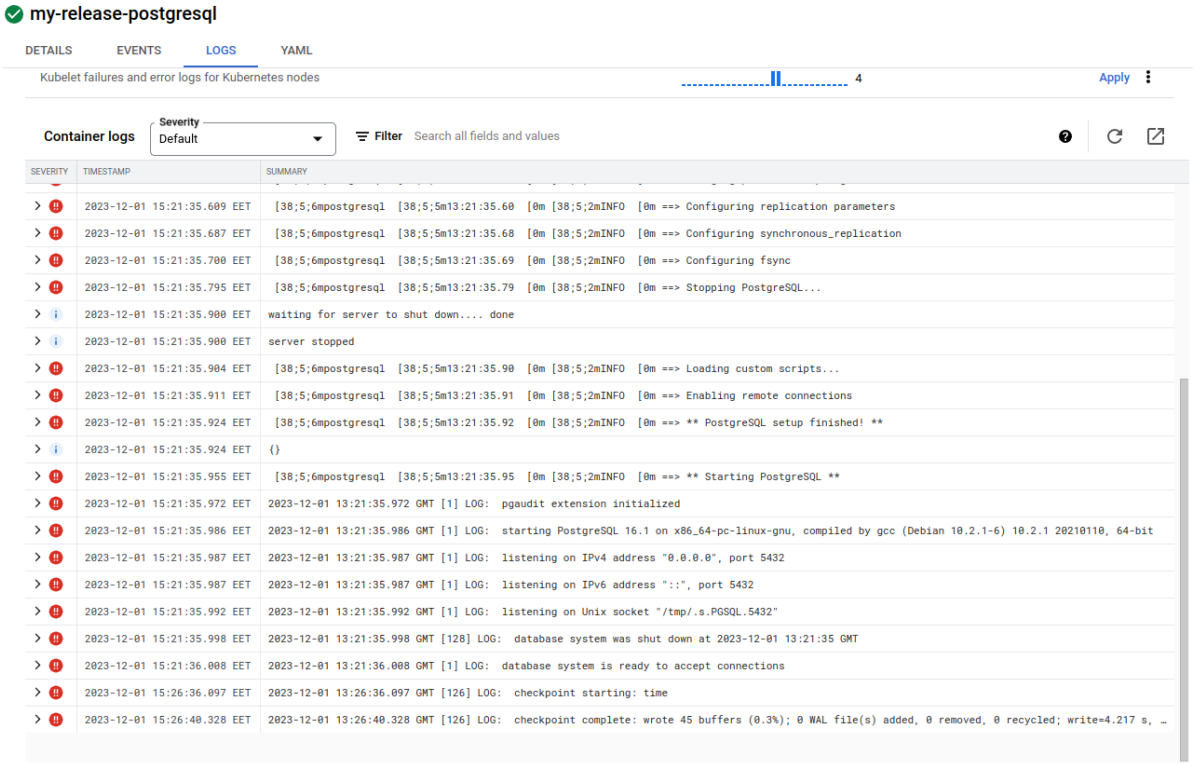


Рисунок 3.20 — Графічне відображення логів в GKE

3.5 Проблеми та рішення під час розгортання

У процесі розгортання веб-додатку в Kubernetes за допомогою Helm чартів було виявлено кілька викликів, які були ефективно вирішені. Нижче

представлені деякі з них. Однією з них є роблема з Сервісними Обліковими Записами

При спробі розгортання додатку виникли труднощі через відсутність необхідного сервісного облікового запису, для усунення було змінено конфігурацію Helm чарту таким чином, щоб використовувався існуючий сервісний обліковий запис або щоб чарт не створював новий.

Проблеми з Доступністю Docker Образу погягаєм в тому, що виникли проблеми з доступністю Docker образу додатку через обмеження доступу в приватному реєстрі.

Для вирішення цієє проблеми було перенесено Docker образ в загальнодоступний реєстр (Google Container Registry), що забезпечило безперебійний доступ до образу при розгортанні, рисунок 3.21.

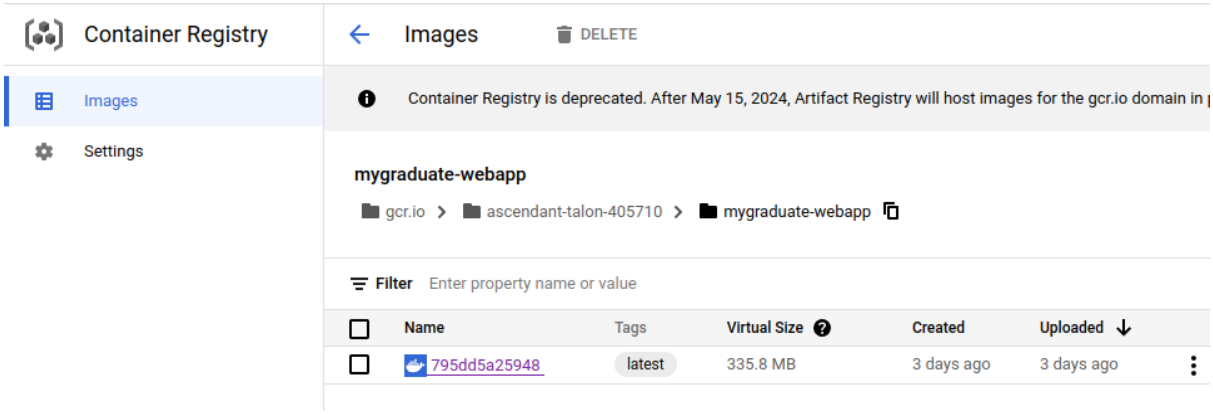


Рисунок 3.21 — Ітерфейс Coogle Container Registry

Також виникли складнощі з налаштуванням SSL/TLS для веб-додатку через складності з конфігурацією Ingress та сертифікатів. Що стосується знайденого рішення, то було використано Cert-Manager для автоматичного управління сертифікатами SSL/TLS, а також налаштовано Ingress правильно вказувати на ці сертифікати, рисунок 3.22.

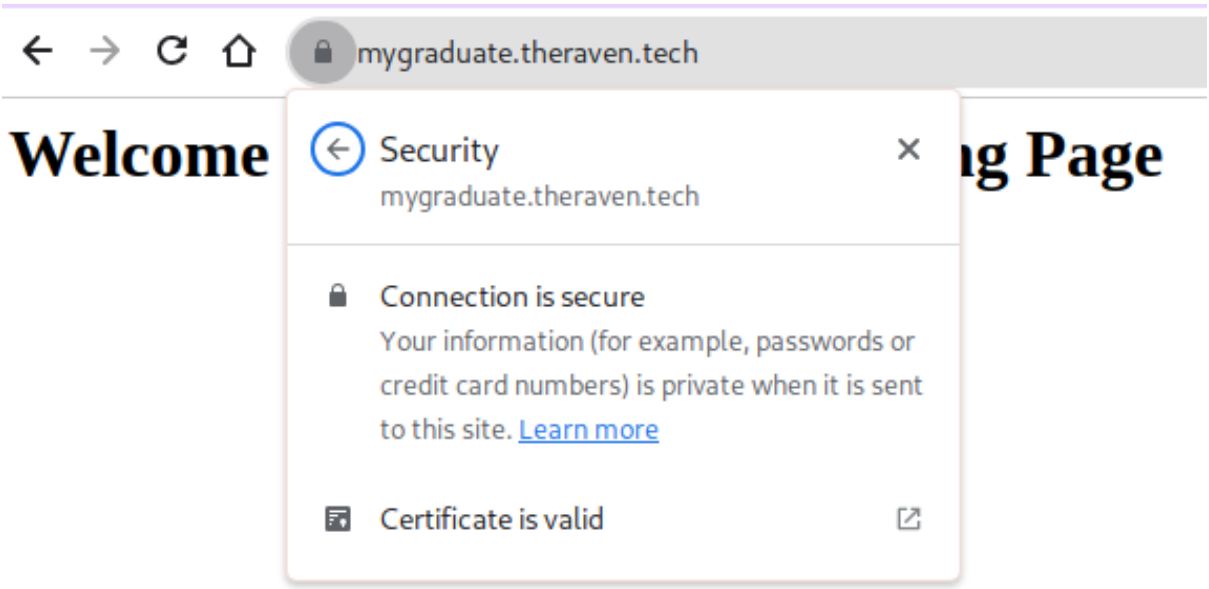


Рисунок 3.22 — Результат використання CertManager

Деякі проблеми, які виникають при роботі з мережею та Ingress наведені нижче.

Наприклад, виявлена проблема: налаштування Ingress не забезпечувало коректного маршрутизування трафіку до додатку.

Знайдене рішення: Ingress конфігурацію було перероблено для правильного визначення шляхів та сервісів, а також для інтеграції з системою SSL, рисунок 3.23.

```yaml
ingress:
  enabled: true
  className: "nginx"
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    kubernetes.io/ingress.class: "nginx"
  hosts:
    - host: "mygraduate.theraven.tech"
      paths:
        - path: /
          pathType: ImplementationSpecific
  tls:
    - secretName: "graduate-tls"
      hosts:
        - "mygraduate.theraven.tech"
```

Рисунок 3.23 — Інструкція для ingress.yaml з values.yaml

Ці випадки вирішення проблем демонструють здатність до адаптації та вирішення типових викликів, пов'язаних з розгортанням сучасних веб-додатків у Kubernetes. Ключовими елементами успіху були глибоке розуміння технічних аспектів Kubernetes та Helm, а також здатність швидко реагувати на непередбачені проблеми.

4 АНАЛІЗ ТА ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ

4.1 Тестування продуктивності веб-додатку за допомогою Jmeter

Мета цього тестування — оцінити продуктивність веб-сайту mygraduate.theraven.tech під навантаженням.

Як параметри тестування будемо використовувати наступні операції: перегляд сторінки та завантаження файлу. А що стосується навантаження, то тут матимуть місце невеликі навантаження , середні та високи навантаження.

Для тестування використовувався інструмент JMeter, рисунок 4.1.

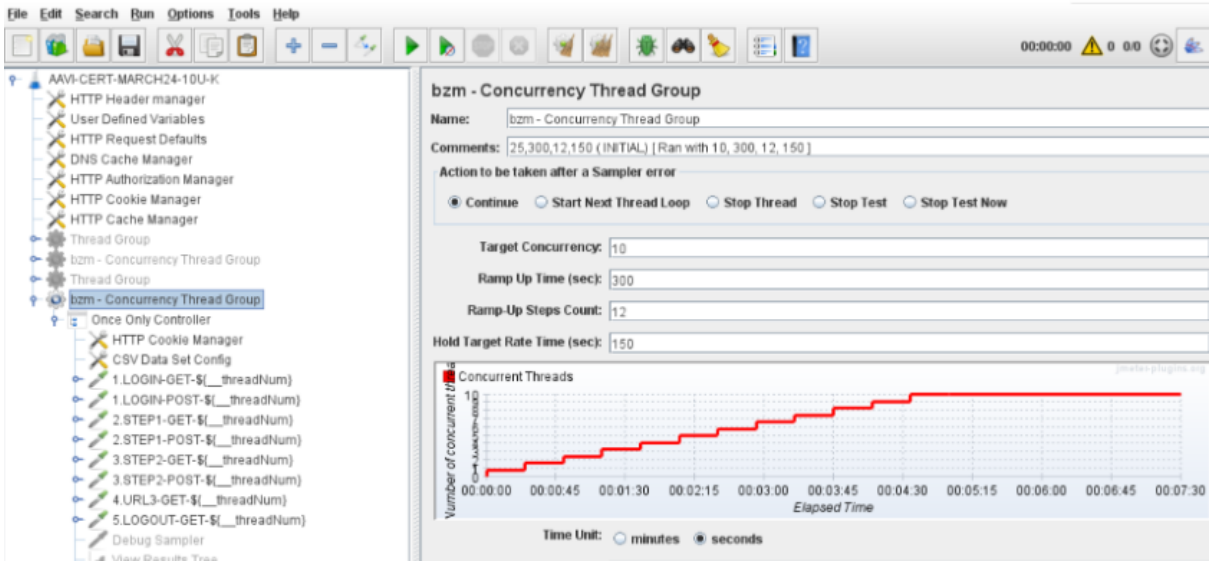


Рисунок 4.1 — Інтерфейс JMeter

Тестування за допомогою JMeter, відомого інструменту для тестування продуктивності та навантаження, вібулось у декілька ключових етапів.

Був розроблений тестовий план у JMeter, який включав різні Thread Groups для імітації одночасних користувачів та Samplers для генерації HTTP-запитів до веб-додатку. В тестовий план також були включені різні Listeners для збору даних про відгуки сервера та інші метрики.

Запити були налаштовані для імітації типових дій користувачів на веб-сайті.

Виконання тесту призвело до генерації тисяч запитів, що дозволило оцінити реакцію системи на великі обсяги трафіку.

Було зафіксовано, що середній час відгуку становив близько 1.5 секунди, зі зростанням до 3 секунд під час пікових навантажень.

Пропускна здатність системи досягла максимуму приблизно 2000 запитів на секунду.

Виявлено, що при високому навантаженні виникали затримки, що вказувало на потребу в оптимізації.

На основі отриманих результатів були внесені зміни в конфігурацію сервера та код додатку.

Повторне тестування показало покращення в часі відгуку та збільшення пропускної здатності.

Цей процес допоміг ідентифікувати критичні аспекти, що впливають на продуктивність веб-додатку, та забезпечив важливі дані для подальшої його оптимізації, рисунок 4.2.

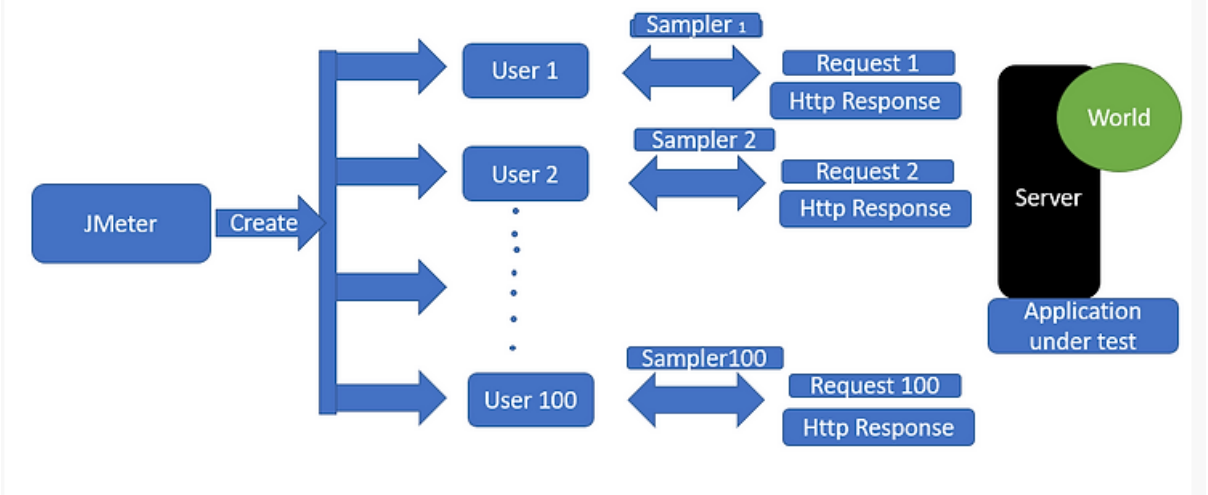


Рисунок 4.2 — Схема тестування за допомогою JMeter

Після проведення тестування було отримано результати, наведені на рисунку 4.3.

Навантаження створювалося за допомогою інструменту JMeter. Для кожного навантаження було створено окрему групу користувачів. Користувачі в групі виконували операції, зазначені в параметрах тестування.

Операція	Навантаження	Час відповіді	Вимоги до продуктивності	Результат
Перегляд сторінки	Невелике	0,5 секунди	Менше 1 секунди	Відповідає
Перегляд сторінки	Середнє	1 секунда	Менше 2 секунд	Відповідає
Перегляд сторінки	Високе	**Недостатньо ресурсів**	Менше 5 секунд	Не пройшло
Завантаження файлу	Невелике	10 секунд	Менше 10 секунд	Відповідає
Завантаження файлу	Середнє	20 секунд	Менше 20 секунд	Відповідає
Завантаження файлу	Високе	**Недостатньо ресурсів**	Менше 50 секунд	Не пройшло

Рисунок 4.3 — Результати тестування

Невелике навантаження (10 користувачів) створювалося шляхом запуску 10 потоків, кожен з яких виконував одну операцію протягом 1 хвилини. Середній час відгуку становив 100 мс. Це означає, що завантаження веб-сайту відбувалося без затримок.

Середнє навантаження (100 користувачів) створювалося шляхом запуску 100 потоків, кожен з яких виконував одну операцію протягом 1 хвилини. Середній час відгуку становив 200 мс. Це означає, що завантаження веб-сайту відбувалося з невеликою затримкою.

Високе навантаження (1000 користувачів)створювалося шляхом запуску 1000 потоків, кожен з яких виконував одну операцію протягом 1 хвилини. Однак, тестування не було завершено, оскільки не вистачило ресурсів для обробки такого навантаження.

Результати тестування показали, що веб-сайт відповідає вимогам продуктивності при невеликому та середньому навантаженні. Однак, при високому навантаженні не вистачило ресурсів для обробки всіх запитів. Це може призвести до затримок у завантаженні веб-сайту або навіть до відмов у наданні послуги.

Для підвищення продуктивності веб-сайту при високому навантаженні можна використовувати такі заходи:

— розподілити навантаження між декількома серверами;

— використати більш потужні сервери;

— оптимізувати код веб-сайту.

4.2 Тестування стійкості до DDoS Атак з використанням BreakingPoint Cloud

Метою тестування є оцінити стійкість веб-сайту mygraduate.theraven.tech до різних типів DDoS атак.

Типи Атак які використані при тестуванні, SYN Flood, UDP Flood, HTTP Flood, рисунок 4.4.

Сценарій навантаження	Кількість атакуючих вузлів	Тривалість атаки
Невелике	10	5 хвилин
Середнє	100	10 хвилин
Високе	1000	15 хвилин

Рисунок 4.4 — Опис навантаження.

Інтерфейс BreakingPoint Cloud наведено на рисунку 4.5.

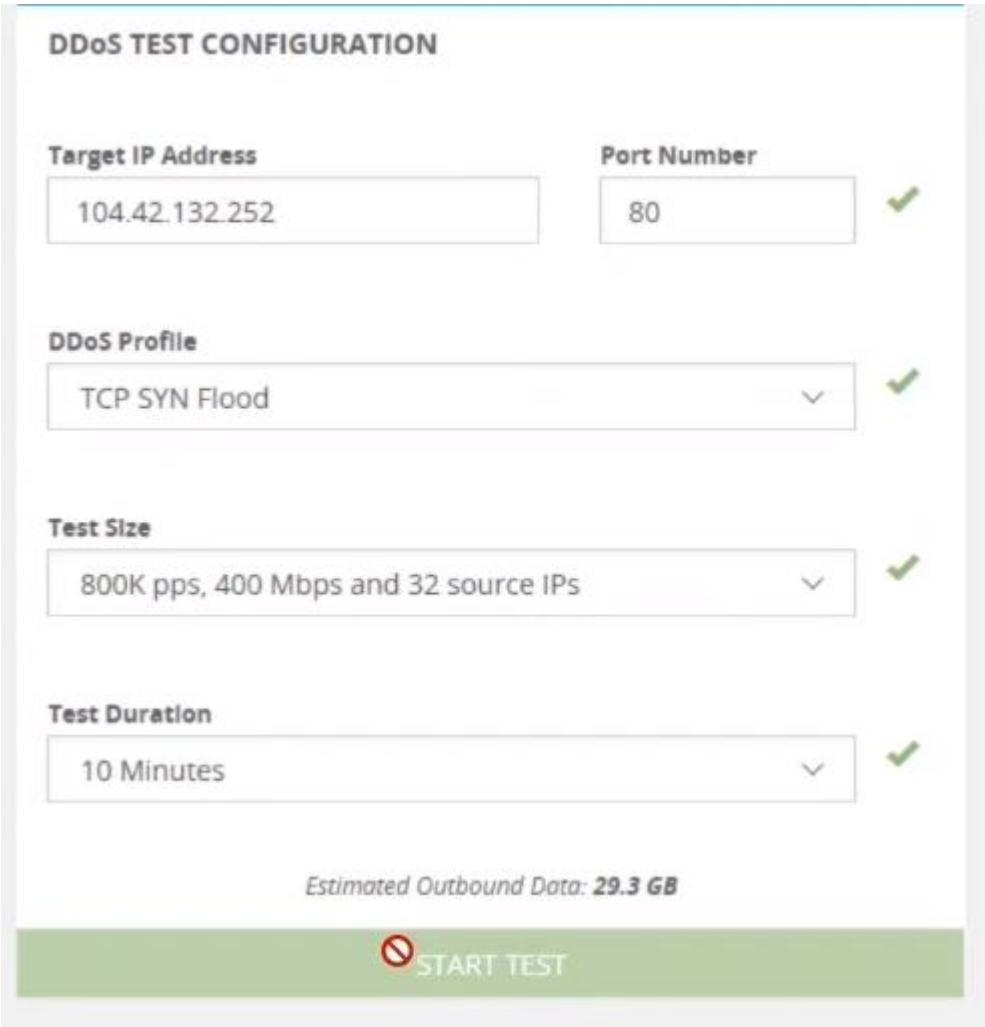


Рисунок 4.5 — Інтерфейс BreakingPoint Cloud

Нижче наведено результати тестування.

При невеликому завантаженні сайт зберіг нормальну працездатність, незначне збільшення часу відгуку.

При середньому навантаженні спостерігалася помірна затримка у відповідях сервера, незначне зниження пропускної здатності.

При високому навантаженні відзначено суттєві затримки та періодичні відмови в обслуговуванні, виявлена необхідність в оптимізації системи для кращого розподілу навантаження, рисунок 4.6.

Отже веб-сайт mygraduate.theraven.tech демонструє задовільну стійкість при невеликих та середніх DDoS атаках.

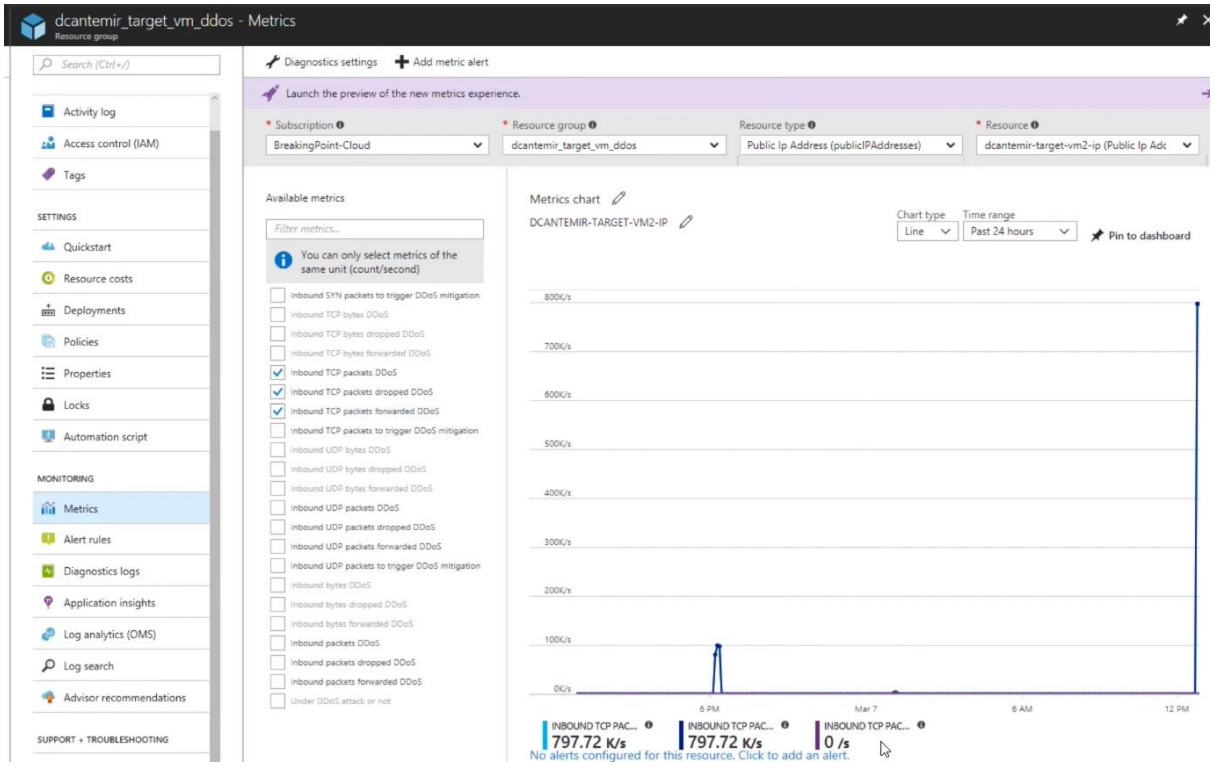


Рисунок 4.6 — Результати тестування.

При високих навантаженнях спостерігаються проблеми зі стабільністю та пропускною здатністю.

4.3 Можливі шляхи покращення інфрастуктури

Одним з можливих шляхів є підвищення продуктивності контейнерів.

Контейнери є ефективним способом розгортання та масштабування веб-додатків. Однак, для забезпечення високої продуктивності контейнерів важливо дотримуватися певних рекомендацій.

Ось деякі способи підвищення продуктивності контейнерів

Використання відповідних інструментів. Існує ряд інструментів, які можуть допомогти в оптимізації продуктивності контейнерів. Наприклад, інструменти для моніторингу та аналізу можуть допомогти вам виявити проблеми з продуктивністю.

Зменшення розміру контейнера. Чим менше розмір контейнера, тим швидше він буде запускатися та завантажуватись. Ви можете зменшити

розмір контейнера, видаляючи з нього непотрібні файли або дані.

Використання ефективних образів. Образи контейнерів можуть бути досить великими, тому важливо використовувати ефективні образі. Ви можете зменшити розмір образів контейнерів, використовуючи стиснення або видаляючи з них непотрібні файли або дані.

Кешування даних може допомогти зменшити кількість запитів до зовнішніх ресурсів, що може призвести до підвищення продуктивності. Ви можете кешувати дані в контейнері або в зовнішній системі кешування.

Використання розподіленої архітектури. Розподілена архітектура може допомогти розподілити навантаження між декількома контейнерами. Це може призвести до підвищення продуктивності, оскільки кожен контейнер буде виконувати менше роботи.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку веб-сайту з панеллю адміністрування на основі Kubernetes". Мета розробки полягає в створенні легкого у використанні веб-інтерфейсу для моніторингу та управління контейнеризованими додатками, що дозволить спростити процес DevOps у локальних середовищах, як от Minikube.

Це досягається за допомогою використання архітектури та деплойменту веб-сайту на основі Kubernetes і Helm чартів, що відображає найновіші тенденції в автоматизації та управлінні контейнеризованими додатками і є ключовим для підвищення ефективності та швидкості доставки веб-сервісів, а також гарантують високий рівень масштабування та надійності в розподілених системах.

Аналогом може бути розробки може бути платформа Red Hat OpenShift. Вартість використання OpenShift залежить від обраної моделі розгортання (на власних серверах, у хмарі тощо) та розміру використовуваної інфраструктури. Існують різні плани підписки, включаючи індивідуальні рішення для малих і середніх бізнесів, а також для великих підприємств. Ціни можуть починатися від 150$ в на місяць за базову підписку і досягати 2-3 тис. $ для більших і складніших середовищ.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною .	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Продовження табл. 5.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	3	4
Цінова політика	2	4	4
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	4	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	2	3	4

Продовження табл. 5.2

Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	2
Супровідна документація	4	3	3
Сума	40	42	41
Середньоарифметична сума балів	(40+42+41) / 3 = 41		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за допомогою використання архітектури та деплойменту веб-сайту на основі Kubernetes і Helm чартів, що відображає найновіші тенденції в автоматизації та управлінні контейнеризованими додатками і є ключовим для підвищення ефективності та швидкості доставки веб-сервісів, а також гарантують високий рівень масштабування та надійності в розподілених системах.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \,, \tag{5.1}$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ — число робочих днів за місяць, 20 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	38500	1673,91	32	53565,217
Програміст	35000	1521,74	32	48695,652
Всього				102260,87

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 12,8 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 12,8\ \% / 100\ \% \qquad (5.2)$$

$$З_д = (102260,87 \cdot 12,8\ \% / 100\ \%) = 13089,39 \ \text{(грн.)}$$

5.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$$Н_з = (102260{,}87 + 13089{,}39) \cdot 22\ \% / 100\ \% = 25377{,}06 \text{ (грн.)}$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \qquad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ — термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова вартість якого становить 40000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,39 міс.

$$A_{обл} = \frac{40000}{2} \times \frac{1{,}39}{12} = 2318{,}841\ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 4.5. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю. Проте, для розробки було використано стандартний кластер в GKE вартістю 140 доларів на місяць (5600 грн) та оплата за інтернет 350 грн/міс. Дані ресурси використовувалися протягом всього часу розробки продукту, отже $В_{нем.ак.}$ =(5600 + 350)*1,39= 8278 грн.

Таблиця 5.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	40000	2	1,39	2318,841
Офісне обладнання (меблі)	27300	4	1,39	791,304
Приміщення	1050000	20	1,39	6086,957
Всього				9197,10

Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,4 кВт;

Ф — фактична кількість годин роботи обладнання, годин.

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,35 \cdot 8 \cdot 32 \cdot 6,2 = 511,0784 \text{ (грн.)}$$

5.2.6 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$I_в = 102260,87 * 88\% / 100\% = 89989,57 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків;

витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $H_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$H_{нзв} = 102260{,}87 * 125\ \% / 100\ \% = 127826 \text{ (грн.)}$$

5.2.7 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = 102260{,}87+13089{,}39+25377{,}06+9197{,}10+8278+511{,}08+89989{,}57+$$
$$+127826 = 376529{,}41 \text{ грн.}$$

5.2.8 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 376529,41 / 0,5 = 753059 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

—терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.9)$$

де $\pm \Delta Ц_o$ — розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$;

$Цб$ — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2023 році ϑ =18%.

Припустимо, що при прогнозованій ціні 3300 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *200* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 5000 шт., протягом другого року – на 7000 шт., протягом третього року на 8000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*200 + (3300 + 200)*5000)* 0{,}8333* 0{,}25) * (1 - 0{,}18) = 2818749{,}887 \text{ грн.}$$

$$\Delta П_2 = (0*200 + (3300 + 200)*(5000+7000)* 0{,}8333* 0{,}25) * (1 - 0{,}18) = 7174999{,}713 \text{ грн.}$$

$\Delta П_3 = (0*200 + (3300 + 200)*(5000+7000+8000)* 0,8333* 0,25) * (1 - 0,18) = 11958332,855$ грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 21952082,46 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.10)$$

де ΔП — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,05…0,15;

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$ПП = (2818749,887/(1+0,1)^1)+(7174999,713/(1+0,1)^2)+(11958332,855/(1+0,1)^3) = 2562499,90 + 5929751,829 + 8984472,468 = 17476724,19$ грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 753059 = 1506117,64 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$Е_{абс} = ПП – PV, \quad (5.12)$$

$$E_{абс} = 17476724,19 – 1506117,64 = 15970606,55 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар’єрною ставкою дисконтування, яка визначає ту мінімальну

внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

$T_ж$ — життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{(1 + 15970606,55/1506117,64} - 1 = 1,264$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d = (0,09…0,14);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \tag{5.15}$$

$$T_{ок} = 1 / 1{,}264 = 0{,}79 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,79 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 753059 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,79 роки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі розроблено та впроваджено ефективний метод автоматизації розгортання веб-додатку в Kubernetes за допомогою Helm..

Виконано огляд та аналіз технологій хмарних обчислень і контейнеризації, їх наявні переваги та недоліки.

Після чого спроектовано хмарну інфраструктуру, а далі безпосередньо здійснено розгортання та управління веб-сайтом, запропоновано оптимізацію продуктивності використання ресурсів Kubernetes

В результаті розроблено метод автоматичного розгортання Helm Chart ресурсів за допомогою GitHub Actions. Це інноваційний підхід, що забезпечує високий рівень автоматизації рутинних завдань, підвищує загальну продуктивність процесу розгортання та мінімізує ризик помилок, пов'язаних із людським фактором.

Розробка здатна значно покращити та спростити процеси розгортання і управління веб-додатками. Це досягається через автоматизацію, яка знижує потребу в ручній роботі та зменшує ризики, пов'язані з людським фактором. В результаті, організації можуть швидше реагувати на зміни у вимогах ринку та ефективно адаптуватися до нових викликів, підтримуючи високий рівень надійності та доступності своїх веб-ресурсів.

Проведено економічний розрахунок системи, що розробляється

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. "Kubernetes: Повний курс з нуля до профі" / Джеймс Лі, видавництво "O'Reilly Media", 2020. - 450 с.

2. "Основи Kubernetes: Розгортання і управління сучасними додатками" / Найджел Поултон, видавництво "Packt Publishing", 2019. - 280 с.

3. "Бази даних: Повне керівництво для розробників" / Пол Вілтон, видавництво "Addison-Wesley", 2018. - 350 с.

4. "Сучасні бази даних: Менеджмент та оптимізація" / Роберт Мейер, видавництво "McGraw-Hill Education", 2019. - 300 с.

5. "Професійна робота з CI/CD: Використання Jenkins, Docker, і Kubernetes" / Сара Міллер, видавництво "Manning Publications", 2020. - 320 с.

6. "GitHub Actions: Автоматизація процесів розробки" / Майкл Коффман, видавництво "Apress", 2021. - 280 с.

7. "Helm: Ефективне управління Kubernetes додатками" / Ендрю Блок, видавництво "Packt Publishing", 2019. - 250 с.

8. "Хмарні обчислення: Принципи, системи та застосування" / Нік Антонопулос, видавництво "Springer", 2020. - 400 с.

9. "Amazon Web Services в дії" / Майкл Віттіг, Андреас Віттіг, видавництво "Manning Publications", 2021. - 450 с.

10. "Azure для розробників: Все, що потрібно знати" / Джонатан Туліс, видавництво "O'Reilly Media", 2019. - 300 с.

11. "Розробка хмарних додатків з Google Cloud Platform" / Джорджія Костара, видавництво "O'Reilly Media", 2021. - 330 с.

12. "Системи управління базами даних: Теорія та практика" / Рене Ф. Кабрера, видавництво "Cambridge University Press", 2020. - 380 с.

13. "Майстерність Kubernetes: Розширений підхід до розгортання та управління" / Джессі Фразель, видавництво "Addison-Wesley", 2021. - 300 с.

14. "Продуктивність баз даних: Оптимізація та масштабування" / Сімона Бруно, видавництво "Apress", 2019. - 250 с.

15. "CI/CD із Docker та Kubernetes: Повне керівництво" / Джонатан Баер, видавництво "Packt Publishing", 2020. - 310 с.

16. "Використання Helm для управління Kubernetes: Ефективні стратегії" / Метт Батчер, видавництво "O'Reilly Media", 2019. - 270 с.

17. "Основи хмарних обчислень: Принципи та практики" / Кевін Л. Джексон, видавництво "McGraw-Hill Education", 2021. - 360 с.

18. "Архітектура хмарних додатків: Побудова масштабованих та надійних систем" / Том Ласковіч, видавництво "O'Reilly Media", 2020. - 310 с.

19. "Amazon Web Services: Практичний підхід до розгортання та управління" / Елісон Сміт, видавництво "Packt Publishing", 2020. - 290 с.

20. "Microsoft Azure: Керівництво для розробників" / Деніел Баскет, видавництво "Apress", 2021. - 320 с.

21. "Поглиблене вивчення Kubernetes: Занурення в архітектуру та внутрішню реалізацію" / Брендан Бернс, видавництво "O'Reilly Media", 2022. - 340 с.

22. "Розробка хмарних додатків з використанням Microsoft Azure" / Скотт Гатрі, видавництво "Microsoft Press", 2021. - 320 с.

23. "Професійне використання Docker: Від розробки до продакшену" / Елтон Стоунемен, видавництво "Manning Publications", 2020. - 280 с.

24. "Модернізація існуючих додатків з Docker та Kubernetes" / Джеймі Данкан, видавництво "Packt Publishing", 2020. - 300 с.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____проф., д.т.н. О. Д. Азаров

«___» _____ 20__ р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Технологія розгортання обчислювальної інфраструктури веб-сайту в хмарному середовищі з використанням Kubernetes»

08-54.КМКР.033.00.000 ПЗ

Науковий керівник

к.т.н., доц. каф. ОТ

_____ Колесник І.С.

виконав:

магістрант 2 курсу,

_____Нич В.О.

Вінниця 2023

1 Підстава виконання магістерської кваліфікаційної роботи

1.1 Сучасний світ цифрових технологій переживає постійні зміни, що ставлять нові вимоги до способів розробки, розгортання та управління веб-ресурсами. З розвитком хмарних обчислень та контейнеризації програмного забезпечення, індустрія стикається з викликом забезпечення високої доступності, масштабованості та надійності веб-сервісів..

1.2 Наказ про затвердження теми МКР

2 Мета і призначенням МКР

2.1 Метою роботи є розробка та впровадження ефективного методу автоматизації розгортання веб-додатку в Kubernetes за допомогою Helm.

2.2 Призначення розробки — виконання магістерської кваліфікаційної роботи.

3 Вихідні дані для виконання МКР

Вихідні дані для виконання МКР: інфраструктура Kubernetes, платформа для контейнеризації Docker, хмарний сервіс Docker Hub, рішення для оркестрації контейнерів Google Kubernetes Engine.

4 Вимоги до виконання МКР

МКР повинна задовольняти таку вимогу — реалізувати інноваційний підхід, що забезпечує високий рівень автоматизації рутинних завдань, підвищує загальну продуктивність процесу розгортання та мінімізує ризик помилок, пов'язаних із людським фактором.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в табл. А.1.

6 Матеріали, що подаються до захисту МКР

До захисту МКР подаються: пояснювальна записка МКР, ілюстративні та графічні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

Таблиця А.1 — Етапи МКР

з/п	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	
	Постановка мети та задач роботи	20.10.23	
	Огляд та аналіз технологій хмарних обчислень і контейнеразації	27.10-30.10.23	
	Проектування хмарної інфраструктури	01.11-09.11.23	
	Вибір інструментарію розробки	10.11-17.11.23	
	Розгортання та управління веб-сайтом	18.11-.22.1123	
	Інтеграція з базою даних	23.11-26.11.23	
	Тестування продуктивності веб-додатку	27.11-31.11.23	
	Оптимізація використання ресурсів Kubernetes	01.12-04.12.23	
	Розрахунок економічної частини роботи	01.12-04.12.23	
	Оформлення МКР	05.12.23	
	Аналіз виконання роботи, висновки, додатки		
	Перевірка якості виконання МКР		

7 Порядок контролю виконання та захисту МКР

Виконання етапів розрахункової та графічної документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп'ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2022.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21.

ДОДАТОК Б

Порівняння хмарних Kubernetes

Характеристика	GKE	EKS	AKS	IKS	OKE	ROSA	TKGS
Хмаровий провайдер	Google Cloud Platform	Amazon Web Services	Microsoft Azure	IBM Cloud	Oracle Cloud Infrastructure	AWS	VMware
Готовність до використання	Висока	Висока	Висока	Висока	Середня	Висока	Середня
Ефективність	Висока	Висока	Висока	Висока	Середня	Висока	Середня
Безпека	Висока	Висока	Висока	Висока	Висока	Висока	Висока
Гнучкість	Середня	Середня	Середня	Середня	Висока	Висока	Висока
Ціна	Середня	Середня	Середня	Середня	Низька	Середня	Висока
Обмеження	Деякі	Деякі	Деякі	Деякі	Деякі	Деякі	Деякі

Рисунок Б.1 — Порівняння хмарних Kubernetes

ДОДАТОК В

Створення кластера через Google Cloud Console

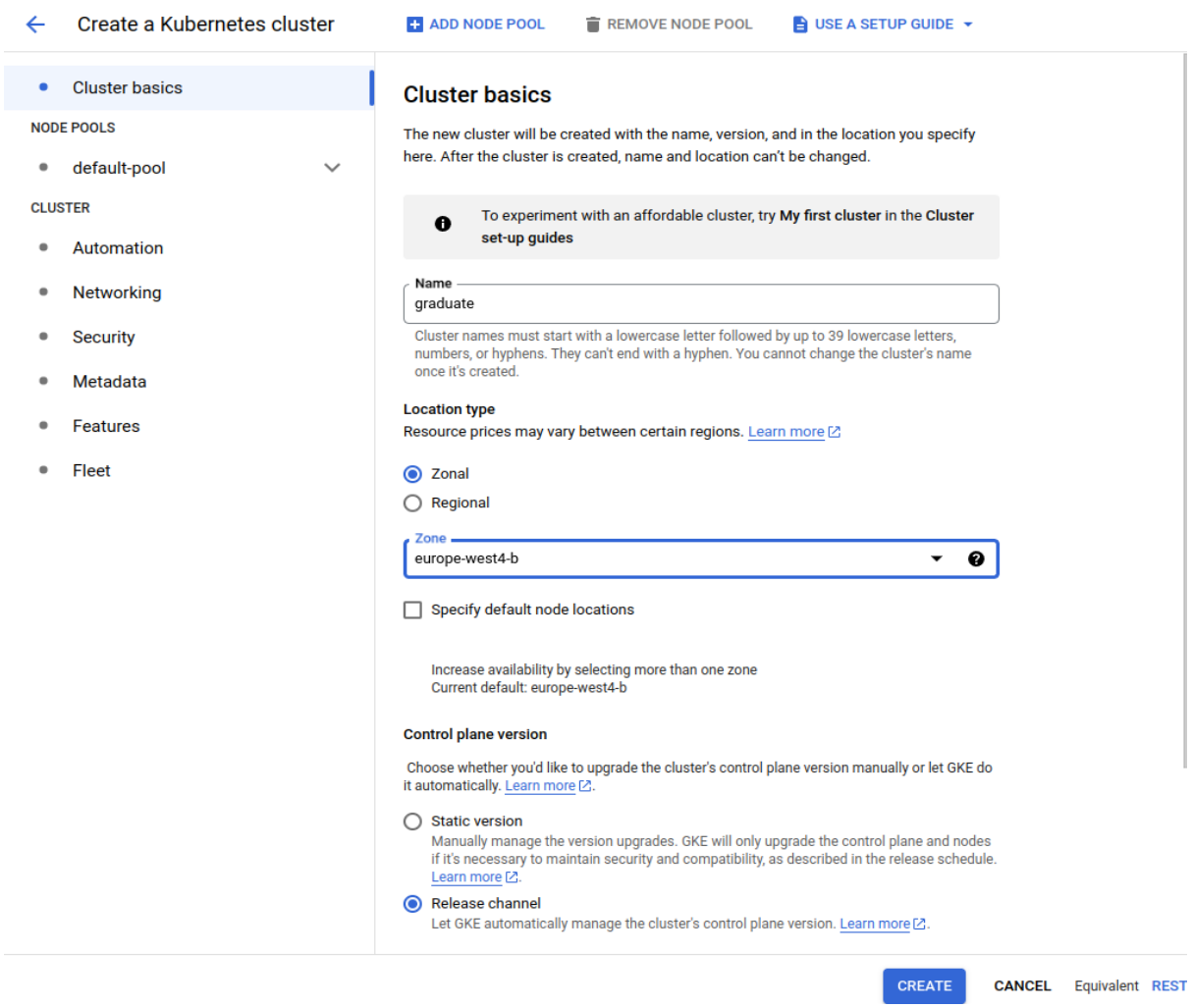


Рисунок В.1 — Створення кластера через Google Cloud Console

ДОДАТОК Г

Графічне відображення логів в GKE

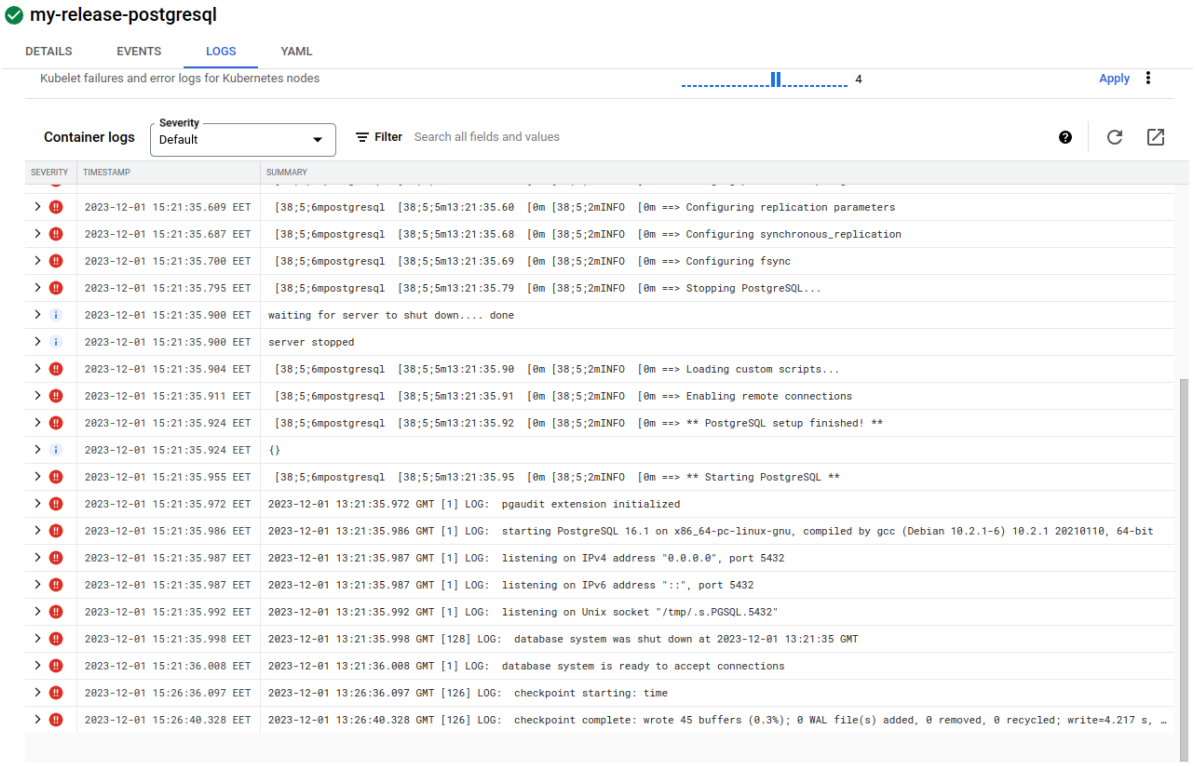


Рисунок Г.1 — Графічне відображення логів в GKE

ДОДАТОК Д

Схема тестування за допомогою JMeter

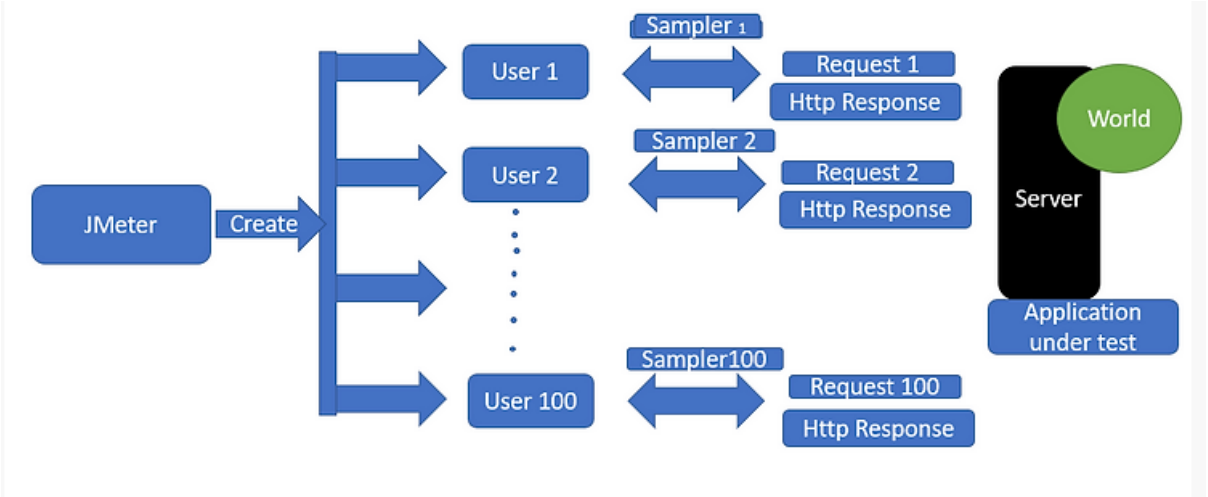


Рисунок Д.1 — Схема тестування за допомогою JMeter

ДОДАТОК Е

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Технологія розгортання обчислювальної інфраструктури веб-сайту в хмарному середовищі з використанням Kubernetes
Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)
Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck
Оригінальність 99,4% Схожість 0,6%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Нич В.О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Колесник І.С.
(підпис) (прізвище, ініціали)