НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

_____ / Голуб Б.Л. /
(підпис) (ПІБ)

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«ІНТЕГРАЦІЯ КОНТЕЙНЕРИЗАЦІЇ З KUBERNETES У ХМАРНІЙ ПЛАТФОРМІ»

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
_____д.е.н., професор_____ _____ _____ _____Руденський Р.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____ _____ _Кафтанников Олексій Юрійович___
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Мудревський Андрій Олександрович_____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

___к.т.н., доцент_____ _____Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Мудревський Андрій Олександрович

Спеціальність 122 – «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «ІНТЕГРАЦІЯ КОНТЕЙНЕРИЗАЦІЇ З KUBERNETES У ХМАРНІЙ ПЛАТФОРМІ» затверджена наказом ректора НУБіП України від 16.12.2024 № 2246с

2. Термін подання завершеної роботи на кафедру_____
(рік, місяць, число)

3. Вихідні дані до роботи: комплексний аналіз технологій хмарних обчислень і контейнеризації, що дозволило визначити їхні ключові переваги та виклики

4. Перелік питань, що розглядаються:
   1 АНАЛІЗ ТЕХНОЛОГІЙ ХМАРНИХ ОБЧИСЛЕНЬ І КОНТЕЙНЕРИЗАЦІЇ
   2 ПРОЕКТУВАННЯ ХМАРНОЇ ІНФРАСТРУКТУРИ
   3. **РОЗГОРТАННЯ ТА АДМІНІСТРУВАННЯ ВЕБ-САЙТУ**

Дата видачі завдання "___"_____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ / Кафтанников О.Ю /
(підпис) (прізвище та ініціали)

КИЇВ – 2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Отримання завдання	21 лютого 2025	
2	Аналіз предметної області	березень 2025	
3	Моделювання предметної області	березень 2025	
4	Проєктування програмної системи	квітень 2025	
5	Розгортання та експлуатація програмного забезпечення	квітень- травень 2025	
6	Економічне дослідження розробки та експлуатації програмної системи	квітень- травень 2025	
7	Оформлення записки	травень 2025	
8	Перевірка на плагіат	2 червня 2025	
9	Проходження нормо контроля	29 травня – 4 червня 2025	
10	Проходження передзахисту	5-7 червня 2025	
11	Захист роботи	14-16 червня 2025	

Студент _____ Мудревський А.О

(підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

Кафтанников О.Ю

_____ _____ _____

(науковий ступінь та вчене звання) (підпис) (ПІБ)

ЗМІСТ

ВСТУП

Актуальність теми. Цифрові технології постійно розвиваються, що вимагає нових підходів до розробки, розгортання та управління веб-ресурсами. Зі зростанням популярності хмарних обчислень і контейнеризації програмного забезпечення, галузь стикається з викликами, пов'язаними із забезпеченням високої доступності, масштабованості та надійності веб-сервісів. У цьому контексті Kubernetes відіграє ключову роль як система оркестрації контейнерів, яка автоматизує розгортання та ефективно розподіляє ресурси.

Дослідження зосереджене на застосуванні Kubernetes для розгортання обчислювальної інфраструктури веб-сайтів у хмарному середовищі. Воно не лише пояснює принципи роботи Kubernetes та його ключові компоненти, а й демонструє практичне використання технології через розгортання реального веб-додатку.

Особливий акцент робиться на інтеграції Kubernetes з Docker та використанні Helm як засобу керування пакетами всередині Kubernetes. Аналізуються переваги та недоліки Kubernetes порівняно з альтернативними рішеннями оркестрації контейнерів, а також можливості заміни Helm іншими інструментами.

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

Для досягнення даної мети, були сформульовані такі ***завдання***:

1. Дослідити основні принципи роботи Kubernetes.

2. Розглянути інструменти управління Kubernetes.

3. Налаштувати кластер Kubernetes у Google Cloud Platform.

4. Розгорнути веб-сайт на основі віртуальної машини в Docker.

5. Вивчити можливості масштабування та управління веб-сайтами в Kubernetes.

6. Оцінити застосування Kubernetes для розгортання веб-сайтів із різною архітектурою.

7. Розробити стратегії підвищення надійності та безпеки веб-сайтів, розгорнутих на Kubernetes.

Об'єктом дослідження є автоматизація розгортання веб-додатків у хмарному середовищі, що включає аналіз існуючих методів та розробку нових підходів для оптимізації цього процесу.

Предметом дослідження є інтеграція Kubernetes і Helm у процес автоматизованого розгортання, а також застосування таких інструментів, як GitHub Actions, для вдосконалення ефективності та безпеки розгортання.

Наукова новизна дослідження полягає у створенні автоматизованого процесу розгортання Helm Chart через GitHub Actions, що об'єднує автоматизацію збірки та розгортання з використанням Kubernetes і Helm. Це рішення забезпечує високу гнучкість та масштабованість, інтегруючись із хмарними сервісами, зокрема Google Cloud, для ефективного управління ресурсами Kubernetes.

Практична цінність отриманих результатів:

- розробка методів підвищення надійності та безпеки веб-сайтів на платформі Kubernetes;

- проведення комп'ютерного моделювання характеристик запропонованої системи, що демонструє її розширені функціональні можливості;

- техніко-економічне обґрунтування доцільності впровадження технології розгортання веб-інфраструктури в хмарному середовищі на основі Kubernetes.

Структура даної роботи складається зі вступ, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі буде розглянуто основні концепції хмарних обчислень, їх переваги та недоліки, а також готові рішення, які використовуються в промисловості. Другий розділ присвячений

проектуванню хмарної інфраструктури, вибору хмарного провайдера та налаштуванню Kubernetes для управління контейнерами. Нарешті, у третьому розділі буде описано процес розгортання веб-додатку в Kubernetes, інтеграцію з базою даних та вирішення можливих проблем, що виникають під час цього процесу. У висновках будуть узагальнені результати даної роботи.

Робота містить 35 рисунків, 24 використаних джерел. Загальний обсяг – 55 сторінок.

РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ХМАРНИХ ОБЧИСЛЕНЬ І КОНТЕЙНЕРИЗАЦІЇ

1.1. Основи хмарних обчислень

За останні десятиліття хмарні обчислення докорінно змінили IT-індустрію, дозволяючи організаціям ефективно керувати ресурсами, забезпечувати масштабованість і гнучкість. Завдяки цим технологіям компанії можуть зосередитися на розробці та впровадженні інновацій, замість витрат на підтримку власної інфраструктури.

Однією з ключових переваг хмарних обчислень є масштабованість, яка дає змогу швидко коригувати використання ресурсів відповідно до поточних потреб, забезпечуючи доступ до сервісів з будь-якої точки світу та економію на придбанні й обслуговуванні обладнання.

У цьому контексті Kubernetes відіграє важливу роль. Це програмне забезпечення з відкритим кодом, розроблене Google, що автоматизує розгортання, масштабування й управління контейнеризованими застосунками. Базуючись на багаторічному досвіді Google у сфері контейнерних технологій, Kubernetes став промисловим стандартом у галузі хмарних обчислень [3].

Головною перевагою Kubernetes є його здатність оркеструвати контейнери, автоматизуючи розгортання та управління застосунками, що забезпечує їхню високу доступність і надійність. Залежно від навантаження, система може автоматично змінювати кількість реплік застосунку, оптимально розподіляючи ресурси між контейнерами.

Ефективне функціонування в хмарному середовищі передбачає швидке розгортання та зручне управління застосунками, і саме тут Kubernetes пропонує потужні інструменти для управління складними системами будь-якого масштабу. Попри свою популярність, існують альтернативні рішення для оркестрації контейнерів, серед яких [2]:

- Docker Swarm – контейнерний оркестратор, розроблений компанією Docker. Він є безкоштовним, має відкритий код і пропонує функціональність, подібну до Kubernetes.
- Apache Mesos – оркестратор контейнерів, що відзначається високою масштабованістю. Він може керувати не лише Docker-контейнерами, а й іншими типами середовищ, забезпечуючи широкий спектр можливостей.

Таким чином, вибір між Kubernetes та його аналогами залежить від конкретних потреб організації, масштабу її інфраструктури та технічних вимог до оркестрації контейнеризованих застосунків (рис. 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

Рис. 1.1. Порівняння контейнерних оркестраторів

Kubernetes володіє низкою переваг, які зробили його одним із найпопулярніших рішень для оркестрації контейнерів. Однією з ключових особливостей є його масштабованість, що дозволяє управляти мільйонами контейнерів, забезпечуючи ефективний розподіл ресурсів відповідно до поточних потреб.

Ще одна важлива перевага Kubernetes – його універсальність. Завдяки підтримці широкого спектра інструментів і ресурсів, ця платформа спрощує процеси розгортання, управління та моніторингу контейнеризованих застосунків. Крім того, Kubernetes може працювати на будь-якій інфраструктурі, що підтримує контейнери, що робить його надзвичайно гнучким та мобільним.

Окрім функціональності, Kubernetes відзначається активним розвитком як проєкт з відкритим кодом. Завдяки внескам великої спільноти розробників, він постійно оновлюється та вдосконалюється, пропонуючи нові можливості та інтеграцію з сучасними технологіями.

Для Kubernetes існує широкий набір інструментів для моніторингу та логування, що дозволяє ефективно відстежувати стан системи та оперативно реагувати на виникаючі проблеми. Універсальність цієї платформи сприяє її широкому поширенню, а підтримка спільноти забезпечує доступ до великої кількості навчальних матеріалів, технічної документації та консультацій.

Вибір Kubernetes як основи для проєкту гарантує інвестування у перевірене, надійне та гнучке рішення, яке здатне адаптуватися до різноманітних потреб оркестрації контейнерів та управління застосунками на всіх етапах розвитку.

Попри численні переваги, Kubernetes має і певні недоліки, які варто враховувати [6]:

- Складність освоєння та налаштування – його функціональність вимагає глибоких знань та досвіду для ефективного використання.
- Вартість – впровадження та підтримка Kubernetes може бути затратним навіть для великих підприємств.

- Безпека – неналежне налаштування може призвести до вразливостей, що потребують уважного контролю та управління.

Kubernetes є ідеальним рішенням для мікросервісної архітектури. Якщо ваша система побудована на принципах мікросервісів, Kubernetes надає потужні інструменти для їхньої оркестрації, управління життєвим циклом, масштабування та моніторингу. Завдяки контейнеризації він забезпечує створення гнучкого та стабільного середовища для розробки, тестування та експлуатації застосунків.

1.2. Контейнеризація: переваги та виклики

Контейнеризація, особливо з використанням Docker та Kubernetes, суттєво впливає на процеси розробки, тестування та розгортання програмного забезпечення. Docker виступає базовою технологією для створення та управління контейнерами, а Kubernetes забезпечує розподілене керування ними в продуктивному середовищі. Їх взаємодія значно покращує ефективність роботи з застосунками.

Docker надає розробникам зручні інструменти для створення, тестування та запуску застосунків у контейнерах, що забезпечує високу портативність та компактність. Кожен контейнер містить усі необхідні залежності, що гарантує стабільну роботу застосунків у різних середовищах [11].

Kubernetes розширює можливості Docker, пропонуючи механізми автоматичного управління контейнерами, їх масштабування та відновлення. Використовуючи Kubernetes, можна динамічно розподіляти та адмініструвати контейнеризовані застосунки навіть у складних інфраструктурах.

Попри переваги контейнеризації, такі як мобільність, компактність та швидкість запуску, існують і певні виклики. Зокрема, питання безпеки та управління залежностями потребують особливої уваги. Контейнери, що використовують спільне ядро операційної системи, можуть бути менш ізольованими порівняно з віртуальними

машинами, що створює потенційні ризики. Крім того, керування залежностями та зберігання даних у контейнеризованому середовищі можуть виявитися складними.

Успішне впровадження контейнеризації вимагає зваженого підходу, що враховує як її переваги, так і можливі ризики.

Docker – це широко використовувана платформа для контейнеризації, яка дає змогу розробникам пакувати застосунки разом із їхніми залежностями в ізольовані контейнери. Спершу варто визначити роль Docker у цьому процесі:

Контейнер – це уніфікована одиниця програмного забезпечення, що містить код застосунку та всі необхідні залежності, забезпечуючи стабільне та швидке розгортання.

Docker спрощує створення, тестування та запуск застосунків у спеціально ізольованому середовищі. При контейнеризації він гарантує стандартизацію середовища виконання, що забезпечує узгодженість між різними етапами розробки та виробничими середовищами. Завдяки цьому розробники отримують гнучкість та надійність у керуванні контейнеризованими застосунками.

Docker Hub – це хмарний сервіс, що забезпечує зберігання та розповсюдження Docker-образів. Образи Docker являють собою зібрані версії програмного забезпечення разом із усіма його залежностями, що дозволяє запускати їх у контейнерному середовищі. Сервіс підтримує два типи репозиторіїв [1]:

- Публічні репозиторії – доступні всім користувачам, містять широкий вибір образів, придатних для розгортання різноманітного програмного забезпечення.
- Приватні репозиторії – обмежені для авторизованих користувачів і використовуються для зберігання образів, що належать конкретним компаніям або організаціям.

Docker Hub пропонує низку корисних можливостей, які роблять його ефективним інструментом для розробників та адміністраторів: зручний інтерфейс, що

дозволяє швидко завантажувати та керувати образами та інструменти безпеки, які забезпечують захист даних від несанкціонованого доступу.

Ця платформа відіграє важливу роль у розгортанні та управлінні контейнеризованими застосунками, надаючи гнучкі та надійні механізми для роботи з Docker-образами.

Docker Hub містить широкий асортимент образів, що можуть використовуватися для запуску різноманітного програмного забезпечення.

У цьому контексті Docker Hub є важливим інструментом для розробників та адміністраторів, які використовують його для ефективного розгортання та управління веб-додатками й іншими програмними системами.

Для створення Docker-образів використовується Dockerfile – текстовий файл, що містить набір команд, необхідних для побудови образу. Docker-образ слугує шаблоном, який визначає параметри контейнера, включаючи базовий образ, команди, змінні середовища та порти. Основні переваги використання Docker [9]:

- Стабільність середовища: забезпечує узгодженість між розробкою та продакшеном, що підвищує його сумісність та портативність.
- Ефективне використання ресурсів: споживає менше обчислювальних потужностей порівняно з традиційними віртуальними машинами.
- Просте розгортання та масштабування: дозволяє легко керувати застосунками, ізолюючи їх один від одного для безпечної та автономної роботи.

Завдяки своїй гнучкості Docker став ключовим елементом у розробці та розгортанні сучасних проєктів. Він забезпечує швидке створення, тестування та постачання застосунків, а контейнеризація відіграє ключову роль у реалізації мікросервісної архітектури та розвитку хмарних рішень.

Ефективне управління та оркестрація контейнерів є ключовими чинниками успішної контейнеризації. У цьому розділі розглядається оркестрація за допомогою

Kubernetes – потужного інструменту для автоматизованого керування контейнерними середовищами.

Kubernetes – це програмне забезпечення з відкритим кодом, яке забезпечує оркестрацію та адміністрування контейнерів. Він надає інтуїтивний інтерфейс для розгортання, масштабування та управління контейнеризованими застосунками в режимі реального часу. Основні переваги використання Kubernetes включають [12]:

- Автоматичне масштабування контейнерів відповідно до навантаження, що підвищує стабільність і доступність додатків.
- Самовідновлення, завдяки якому система здатна виявляти та перезапускати контейнеризовані сервіси, що вийшли з ладу, забезпечуючи надійність середовища.
- Декларативне налаштування, що дозволяє описати бажаний стан системи, а Kubernetes самостійно підтримуватиме його.

Щоб використовувати Kubernetes, необхідно створити кластер, який складається з вузлів (nodes). Вузли можуть бути фізичними серверами або віртуальними машинами, що запускають контейнеризовані застосунки. Kubernetes управляє їх розгортанням та функціонуванням.

При дослідженні Kubernetes слід приділити увагу його основним принципам та механізмам, що дозволяють ефективно оркеструвати контейнеризоване середовище.

Перший принцип пов’язаний із розподілом робочих навантажень (Workload Division): Kubernetes дозволяє розгортати додатки у вигляді окремих робочих компонентів, таких як розгортання (Deployments) та групи контейнерів (Pods), що забезпечує гнучке управління різними частинами за стосунків [7].

Другий принцип стосується служб (Services): Kubernetes надає механізми для організації внутрішньої та зовнішньої доступності додатків, що дозволяє забезпечити ефективну комунікацію між контейнерами та інтеграцію із зовнішніми сервісами [7].

Третій принцип охоплює управління сховищем та ресурсами (Storage and Resources): Kubernetes забезпечує можливість зберігання даних і раціонального розподілу ресурсів між контейнерами за допомогою таких об'єктів, як Persistent Volumes та Resource Quotas [7].

Таким чином, оркестрація контейнерів із використанням Kubernetes стала стандартом у сучасній розробці та розгортанні програмного забезпечення. Ця платформа значно підвищує доступність, надійність та ефективність контейнеризованих застосунків, забезпечуючи їхню стабільну роботу в масштабованих середовищах.

1.3. Огляд сучасних хмарних рішень

Окрім розгортання інфраструктури власними силами, існують хмарні провайдери, що пропонують готові рішення для веб-сайтів. Використання таких платформ має низку переваг у порівнянні з самостійною реалізацією [15]:

1) Зручність використання: сервіси базуються на хмарних технологіях, надаючи інтуїтивно зрозумілі інструменти для розгортання веб-сайтів. Це значно знижує технічний бар'єр, дозволяючи користувачам запускати веб-додатки без глибокого розуміння внутрішніх механізмів хмарної інфраструктури.

2) Економія часу: замість ручного налаштування та підтримки інфраструктури користувачі отримують автоматизовані процеси, що значно скорочують час розгортання.

3) Висока безпека: хмарні провайдери забезпечують захист даних і застосунків на професійному рівні, оскільки їхні системи розроблені та підтримуються експертами у сфері кібербезпеки.

Завдяки цим перевагам хмарні рішення стали популярним вибором для компаній та розробників, які прагнуть створювати стабільні та масштабовані веб-додатки.

Amazon Web Services (AWS) пропонує Elastic Beanstalk, що забезпечує зручне розгортання веб-сайтів та веб-додатків із автоматичним масштабуванням.

Microsoft Azure надає Azure App Service, який також спрощує процес розгортання, пропонуючи інтегровані можливості для роботи з різними технологіями [17].

Google Cloud Platform (GCP) включає App Engine, аналог AWS Elastic Beanstalk та Azure App Service, який дозволяє автоматично масштабувати застосунки та управляти ними без серверної інфраструктури.

IBM Cloud пропонує Cloud Foundry, платформу як сервіс (PaaS), що дозволяє розробникам швидко створювати, розгортати та масштабувати хмарні додатки.

Oracle Cloud має Oracle Cloud Infrastructure, яка містить різноманітні сервіси, зокрема віртуальні машини, мережеві рішення та сховища даних, а також спеціалізовані функції для корпоративних завдань.

DigitalOcean, хоча і поступається масштабом провідним хмарним провайдерам, відомий простотою та зручністю, що робить його популярним серед невеликих компаній та стартапів. Його Droplets (віртуальні приватні сервери) та Kubernetes-орієнтовані сервіси пропонують ефективні інструменти для швидкого розгортання за стосунків [10].

Кожен із цих провайдерів має свої особливості, варіанти ціноутворення та методи управління хмарними ресурсами, що дає користувачам можливість вибору відповідно до їхніх потреб та бюджету. При прийнятті рішення важливо враховувати такі фактори, як масштабованість, доступність, рівень безпеки та підтримка.

При виборі хмарних рішень, зокрема Google Cloud Platform (GCP), варто звернути увагу на кілька ключових факторів [5].

- Широкий набір функцій: GCP пропонує потужні можливості для контейнеризації, оркестрації, машинного навчання, штучного інтелекту, роботи з базами даних, аналітики, інфраструктури та бізнес-інтелекту.
- Інноваційність: платформа активно розвиває передові технології, зокрема Kubernetes, TensorFlow, Vertex AI та Cloud Spanner, що робить її привабливою для технологічно орієнтованих компаній.
- Гнучкість та масштабованість: GCP дозволяє підприємствам легко адаптуватися до змінних потреб, зберігаючи високий рівень безпеки для захисту даних та застосунків.

Серед переваг GCP порівняно з AWS та Azure можна виділити доступніші ціни та глобальну інфраструктуру, що робить платформу привабливою для міжнародних компаній.

Однак варто враховувати й певні недоліки, зокрема меншу частку ринку порівняно з AWS та Azure, що може вплинути на доступність навчальних ресурсів та підтримки.

Зважений вибір хмарної платформи залежить від конкретних потреб бізнесу, технічних вимог та стратегічних цілей. GCP залишається одним із провідних рішень для створення масштабованих та інноваційних хмарних інфраструктур.

При виборі хмарного провайдера важливо враховувати специфічні потреби бізнесу, оцінювати вартість рішень, а також звертати увагу на географічну доступність і вимоги до безпеки. Google Cloud Platform, завдяки своїй гнучкості та широкому спектру інноваційних можливостей, може бути оптимальним рішенням для компаній, які шукають надійну, масштабовану та ефективну хмарну платформу.

GCP має низку переваг над іншими хмарними провайдерами. Він пропонує розширений набір функцій, включаючи підтримку контейнеризації та оркестрації,

машинне навчання та штучний інтелект, роботу з базами даних, аналітику та бізнес-інтелект. Крім того, платформа активно розвивається завдяки інвестиціям у передові технології, такі як Kubernetes, TensorFlow, Vertex AI та Cloud Spanner. Гнучкість та масштабованість дозволяють підприємствам легко адаптувати інфраструктуру відповідно до змінних потреб, забезпечуючи високий рівень захисту даних та застосунків.

GCP також має конкретні переваги над AWS і Azure. Його спеціалізовані функції можуть бути особливо корисними для компаній із вузькопрофільними вимогами. Крім того, конкурентне ціноутворення робить платформу доступнішою для певних підприємств. Глобальна інфраструктура забезпечує ефективну роботу компаній, що оперують на міжнародному рівні.

Водночас GCP має і певні недоліки. Його частка ринку менша порівняно з AWS та Azure, що може вплинути на доступність навчальних матеріалів та рівень підтримки. Крім того, він не пропонує весь спектр функціональних можливостей, доступних у конкурентних платформах, що може бути критично важливим для підприємств із специфічними потребами [16].

Попри ці особливості, GCP залишається потужним хмарним провайдером, який пропонує широкий спектр можливостей та інноваційних рішень. Він може бути оптимальним вибором для підприємств, які потребують гнучкої, масштабованої та надійної хмарної інфраструктури (рис. 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

Рис. 1.2. Порівняння хмарних провайдерів

РОЗДІЛ 2. ПРОЕКТУВАННЯ ХМАРНОЇ ІНФРАСТРУКТУРИ

2.1. Вибір провайдера хмарних послуг

Вибір хмарного рішення для оркестрації контейнерів є ключовим етапом у процесі розгортання веб-сайту в хмарному середовищі. Від правильного вибору залежить рівень безпеки, масштабованості та доступності веб-ресурсу.

При виборі відповідного хмарного провайдера необхідно враховувати кілька важливих аспектів. По-перше, оцінити функціональність платформи та переконатися, що вона містить усі необхідні інструменти для роботи із веб-сайтом. По-друге, порівняти цінові пропозиції різних хмарних сервісів, щоб знайти оптимальний варіант з точки зору вартості. Крім того, слід звернути увагу на можливість масштабування інфраструктури, що дозволяє адаптувати веб-сайт до змін рівня навантаження.

Останнім, але не менш важливим фактором є надійність, рівень безпеки та якість підтримки, адже стабільність роботи та захист даних відіграють ключову роль у виборі відповідного хмарного рішення.

2.2. Обґрунтування вибору Kubernetes як платформи

При виборі платформи для оркестрації контейнерів необхідно враховувати такі ключові аспекти, як функціональність, складність використання та вартість. Для розгортання веб-сайту в хмарному середовищі було обрано Google Kubernetes Engine (GKE), яке є повністю керованим рішенням, що оптимізує процес управління контейнерами.

Однією з головних переваг GKE є його глибока інтеграція з екосистемою Google Cloud Platform (GCP). Як нативний сервіс GCP, він забезпечує безперебійну роботу з іншими хмарними інструментами, такими як Cloud Storage, Cloud SQL, Cloud Load Balancing та Cloud Monitoring. Це спрощує процес розгортання та

адміністрування веб-сайту, надаючи централізоване керування всіма необхідними ресурсами [4].

GKE також є повністю керованим рішенням, що звільняє адміністраторів від необхідності самостійного управління Kubernetes-інфраструктурою. Google відповідає за оновлення кластерів, застосування патчів безпеки, автоматичне масштабування та відновлення, дозволяючи розробникам зосередитися на створенні та впровадженні веб-сайту, не відволікаючись на підтримку інфраструктури.

GKE забезпечує високу масштабованість і доступність, дозволяючи веб-сайту адаптуватися до змін у трафіку шляхом автоматичного масштабування та самовідновлення. Кластер GKE підтримує стабільну роботу навіть під час пікових навантажень, гарантуючи безперебійну доступність ресурсу.

Платформа також пропонує розширені механізми безпеки, зокрема Shielded GKE Nodes, Confidential GKE Nodes та Workload Identity, які захищають контейнеризовані робочі процеси від несанкціонованого доступу та потенційних вразливостей.

Зручність управління GKE забезпечується інтуїтивним інтерфейсом Google Cloud Console та gcloud CLI, що значно спрощує процес розгортання, адміністрування та моніторингу кластерів. Додатково доступні інструменти, такі як Cloud Code, що дозволяють розробникам інтегрувати та керувати Kubernetes-застосуваннями безпосередньо зі своїх IDE [13].

GKE підтримується великою та активною спільнотою розробників і користувачів, що забезпечує доступ до детальної документації, навчальних матеріалів, форумів та груп підтримки. Крім того, Google надає офіційну підтримку для клієнтів, які потребують додаткової технічної допомоги.

Google Cloud Kubernetes є одним із найпопулярніших рішень для оркестрації контейнерів. Серед його альтернативних платформ варто відзначити Amazon Elastic

Kubernetes Service (EKS), Azure Kubernetes Service (AKS), IBM Cloud Kubernetes Service і Mesosphere DC/OS.

Кожне з цих рішень має схожий набір функцій, але відрізняється унікальними можливостями та особливостями. При виборі платформи для оркестрації контейнерів важливо враховувати специфічні потреби проєкту та вимоги до інфраструктури.

Хоча Kubernetes та Google Kubernetes Engine (GKE) взаємопов'язані, вони є окремими технологіями у сфері контейнерного управління, і відповідно, їх аналоги також мають свої характерні особливості та підходи до розгортання.

Kubernetes – це платформа з відкритим кодом, призначена для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Серед альтернатив Kubernetes можна виділити OpenShift (Red Hat), Docker Swarm та інші рішення, які також виконують функції оркестрації контейнерів. Вони надають схожий набір можливостей, включаючи управління кластерами, автоматичне розгортання та механізми самовідновлення, проте відрізняються за рівнем інтеграції, масштабованості та доступними інструментами управління [14].

Google Kubernetes Engine (GKE) є хмарним сервісом від Google, що забезпечує кероване управління Kubernetes. У цьому випадку Google бере на себе адміністрування інфраструктури, зокрема управління кластерами, безпеку та масштабування. Інші хмарні сервіси Kubernetes, аналогічні GKE, також надають подібний функціонал, але можуть відрізнятися деталями реалізації, ступенем інтеграції з іншими хмарними сервісами, рівнем безпеки, функціями моніторингу та автоматичного масштабування.

Серед хмарних альтернатив GKE можна виділити [21]:

- Amazon Elastic Kubernetes Service (EKS) – це рішення від Amazon Web Services, яке надає повноцінну платформу для розгортання кластерів Kubernetes, а також додаткові функції безпеки та адміністрування.

- Azure Kubernetes Service (AKS) – платформа від Microsoft Azure, що забезпечує зручне розгортання кластерів Kubernetes із розширеними можливостями керування та безпеки.
- IBM Cloud Kubernetes Service (IKS) – хмарна платформа від IBM Cloud, яка пропонує комплексне рішення для управління контейнеризованими робочими навантаженнями, включаючи захист даних та адміністрування кластерів.

Таким чином, вибір конкретної платформи залежить від потреб бізнесу, рівня інтеграції з іншими сервісами та вимог до безпеки та масштабованості.

Oracle Cloud Infrastructure Container Engine for Kubernetes (OKE) – це кероване хмарне рішення від Oracle Cloud Infrastructure, що забезпечує розгортання та адміністрування кластерів Kubernetes. OKE пропонує всі необхідні компоненти для оркестрації контейнерів, а також розширені функції безпеки та управління.

Red Hat OpenShift Service on AWS (ROSA) – це хмарна Kubernetes-платформа, розроблена спільно Red Hat та Amazon Web Services. ROSA надає повноцінне середовище для розгортання контейнеризованих застосунків, включаючи інструменти управління безпекою та адміністрування кластерів.

VMware Tanzu Kubernetes Grid Service (TKGS) – рішення від VMware, яке забезпечує оркестрацію контейнерів у хмарному середовищі. TKGS включає всі необхідні компоненти для керування Kubernetes-кластерами та пропонує розширені функції безпеки та масштабування.

Рис. 2.1 містить порівняльну характеристику цих хмарних рішень для 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

Рис. 2.1. Порівняння хмарних Kubernetes

2.3. Налаштування та конфігурація Kubernetes-кластеру

Після вибору GKE як платформи для оркестрації контейнерів була розроблена конфігурація кластера. У процесі налаштування виконано такі кроки: створення кластера GKE за допомогою Google Cloud Console, конфігурація мережевих параметрів для забезпечення безпеки та доступу між контейнерами, налаштування механізмів зберігання для підвищення надійності та можливості масштабування, а також впровадження заходів безпеки для захисту від несанкціонованого доступу та потенційних загроз, як зображено на рис. 2.2.

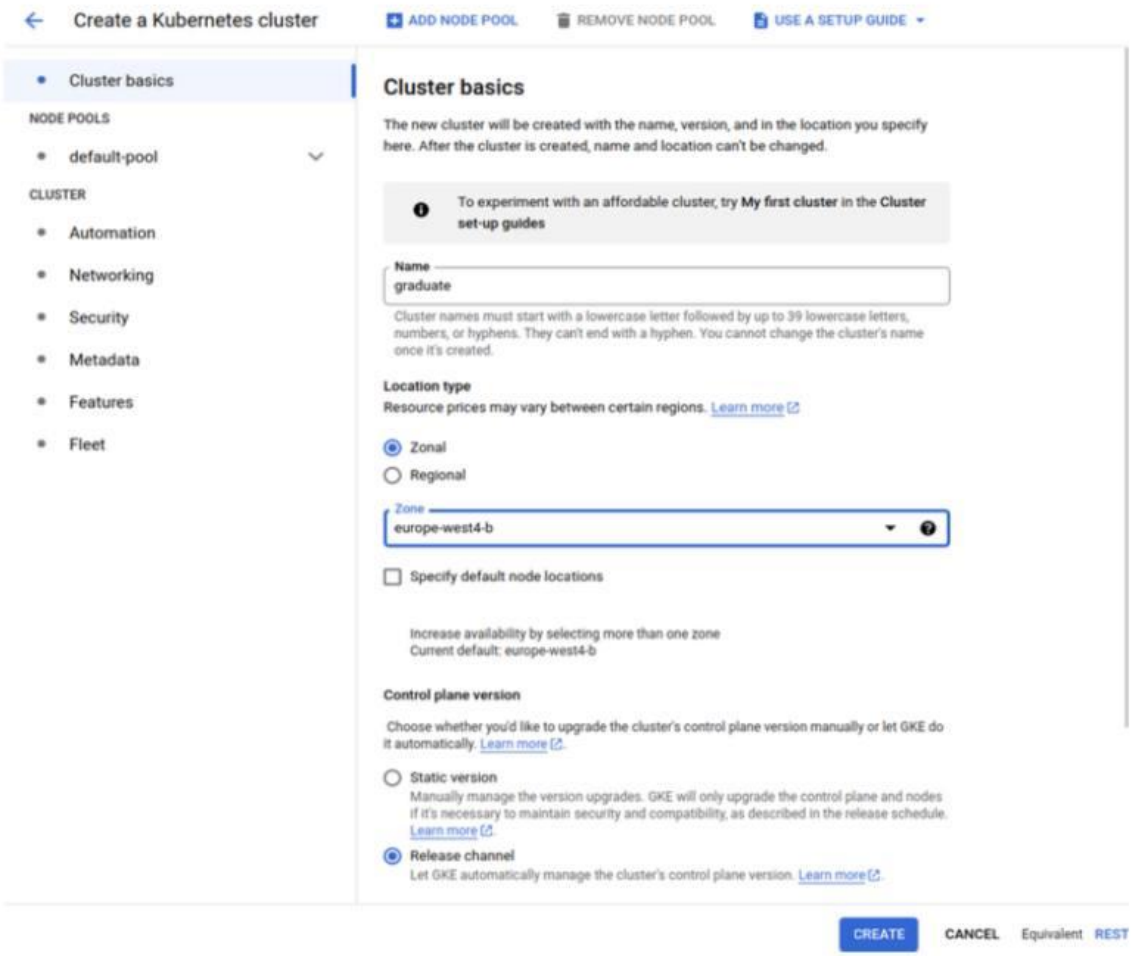

Рис. 2.2. Створення кластеру через Google Cloud Console

Мережеве середовище є ключовим елементом конфігурації Kubernetes-кластера, забезпечуючи взаємодію між контейнерами всередині інфраструктури. Для гарантування безпеки та контролю доступу було реалізовано мережеві політики Kubernetes Network Policies, що дозволяють регулювати комунікацію між компонентами системи. Детальний процес налаштування представлений на рис.2.3.

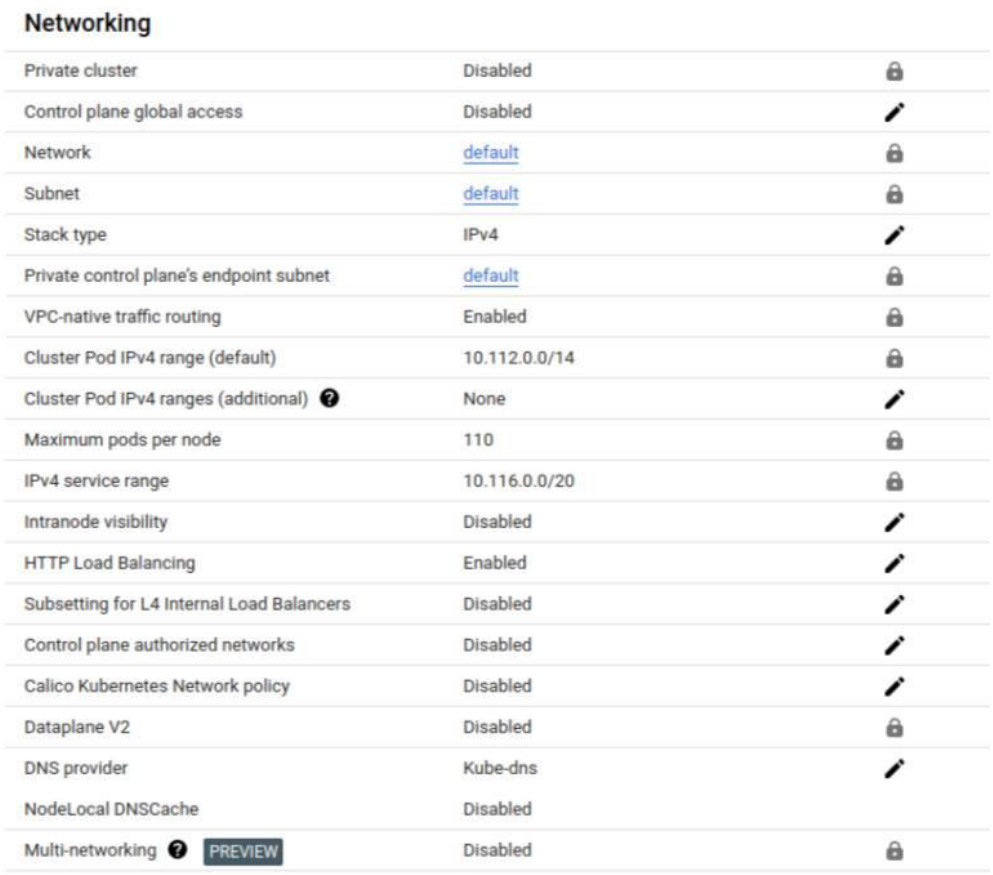

Рис. 2.3. Мережеві параметри кластеру

Кластер Kubernetes налаштований як публічний, оскільки опція private cluster вимкнена. Це забезпечує можливість доступу до нього з будь-якої точки з підключенням до Інтернету, що необхідно для надання користувачам безперешкодного доступу до веб-сайту.

Кластер використовує IPv4-адреси, оскільки стек мережевих протоколів налаштований на IPv4. Це означає, що піди в кластері можуть обмінюватися даними через IPv4-адресацію.

Трафік між вузлами кластера передається через VPC, у якому він розміщений, оскільки маршрутизація через VPC активована. Це покращує безпеку, оскільки дані не передаються через відкритий Інтернет.

Для створення підів кластер може використовувати діапазон IPv4-адрес 10.112.0.0/14, який містить 262144 IP-адреси, що забезпечує можливість масштабування.

Контрольна площина доступна лише в межах регіону розміщення кластера, оскільки глобальний доступ до неї вимкнено.

Для служб кластер може використовувати IPv4-діапазон 10.116.0.0/20, що включає 4096 адрес, дозволяючи розміщувати велику кількість службових компонентів.

Вузли кластера не мають прямого доступу один до одного, оскільки їхня внутрішня видимість вимкнена. Це означає, що для комунікації вони повинні використовувати налаштовану мережу.

Кластер підтримує балансування навантаження, що дозволяє оптимально розподіляти трафік між подами, забезпечуючи ефективне функціонування застосунків.

Зберігання даних є важливим елементом конфігурації Kubernetes-кластера, забезпечуючи надійність та масштабованість контейнерного середовища. Для цього у кластері реалізовано Persistent Volumes (рис. 2.4).

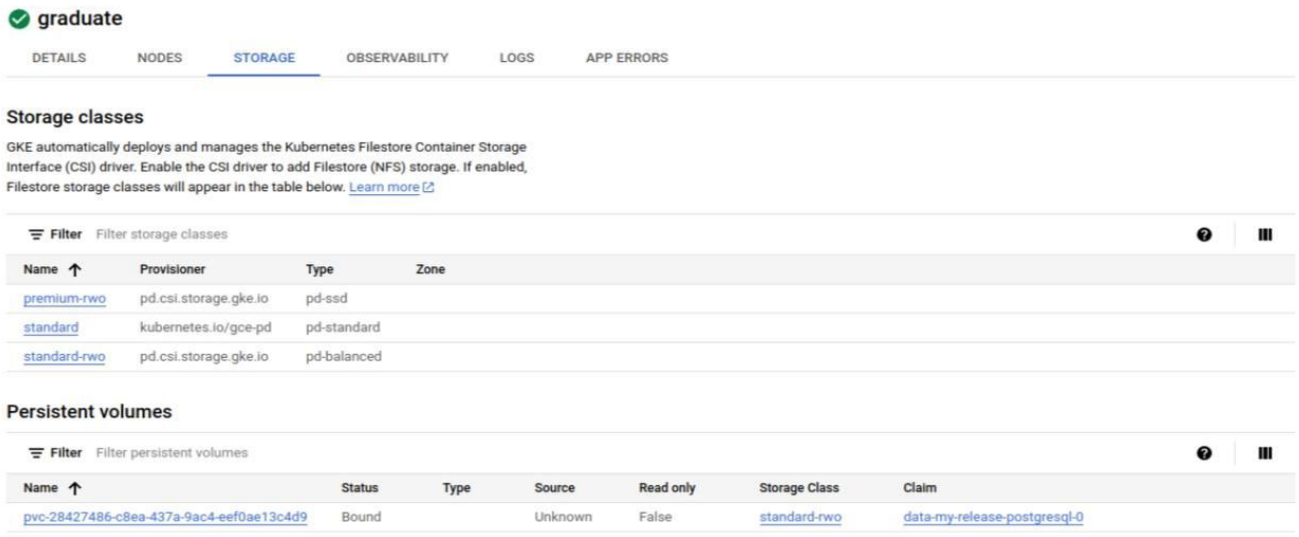

Рис. 2.4. Сховище Kubernetes кластеру

У кластері використовуються два класи сховищ. premium-rwopd.csi.storage.gke.io працює на базі SSD-дисків, що забезпечує високу швидкість читання та запису, тому він оптимальний для застосунків, що потребують швидкого доступу до даних, таких як бази даних. standard-balanced.csi.storage.gke.io використовує диски загального призначення, пропонуючи збалансоване співвідношення між продуктивністю та вартістю. Він підходить для завдань, де швидкість не є критичною [20].

Окрім цього, у кластері наявний постійний том pvc-28427486-c8ea-437a-9ac4-eef0ae13c4d9, який працює на основі класу standard-rwopd. Це означає, що він використовує диск загального призначення та підтримує операції читання і запису. Він пов’язаний із клеймом data-my-release-postgresql-0, що вказує на його використання для зберігання даних бази PostgreSQL.

Безпека є ключовим елементом конфігурації Kubernetes-кластера, забезпечуючи захист контейнеризованих робочих навантажень від несанкціонованого доступу та потенційних загроз. Для гарантування безпечного функціонування кластера були налаштовані наступні механізми [23]:

- Shielded GKE Nodes – функція, яка захищає контейнеризовані застосунки від несанкціонованого доступу до пам’яті та жорстких дисків, підвищуючи рівень безпеки середовища.
- Confidential GKE Nodes – забезпечує ізоляцію конфіденційної інформації, запобігаючи доступу до критичних даних без відповідного дозволу.
- Workload Identity – дозволяє контейнеризованим робочим навантаженням проходити аутентифікацію через існуючі системи ідентифікації, такі як Google Cloud Identity and Access Management (IAM), що підвищує контроль доступу.

На рисунку 2.5 представлена схема налаштувань безпеки в кластері.

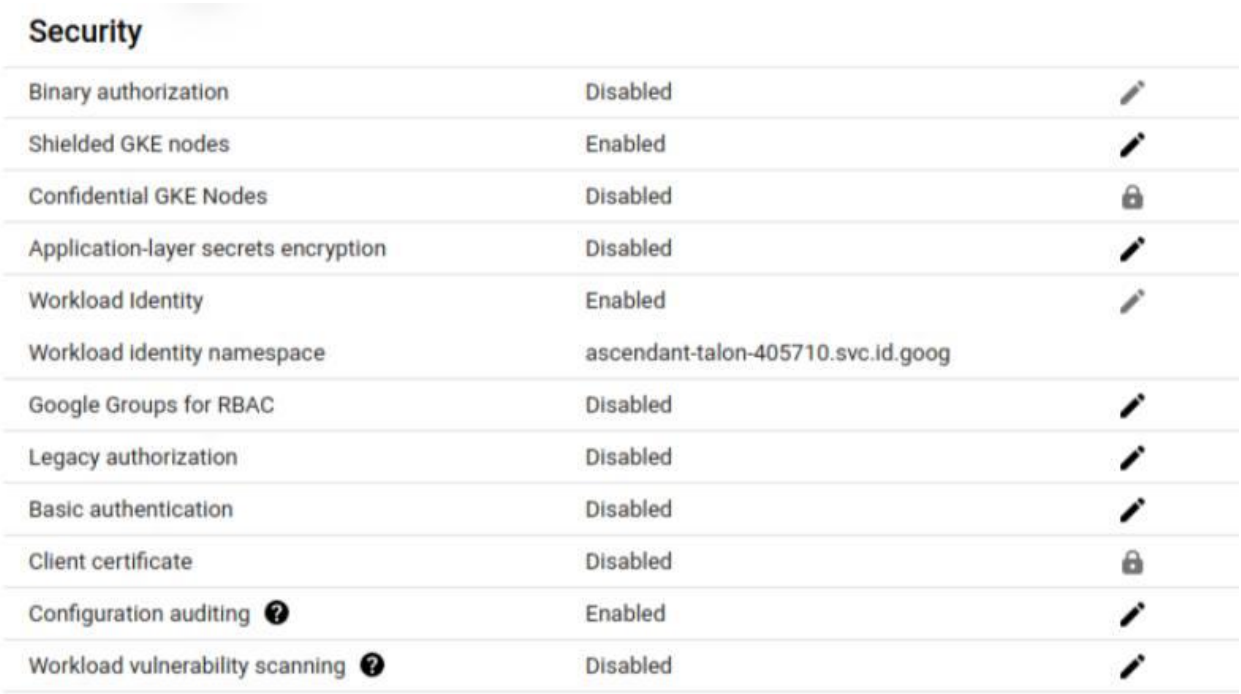

Рис. 2.5. Налаштування безпеки кластеру

2.4. Впровадження та керування контейнерами у Docker

Нижче описані основні компоненти інфраструктури сайту, що працює у середовищі Docker.

Файл app.js є центральним елементом Node.js-додатку, визначаючи логіку роботи сервера та його взаємодію з клієнтами. Він відіграє ключову роль у запуску веб-сайту всередині контейнера Docker.

Директорія node_modules містить усі залежності проєкту, встановлені через npm. Ці модулі розширюють функціонал застосунку та додаються в контейнер у процесі побудови образу.

Файли package.json та package-lock.json відіграють важливу роль в управлінні залежностями Node.js. package.json містить інформацію про проєкт, перелік необхідних залежностей та конфігурацію, тоді як package-lock.json гарантує точність встановлених версій, забезпечуючи стабільність середовища.

Детальна схема конфігурації представлена на рис. 2.6.

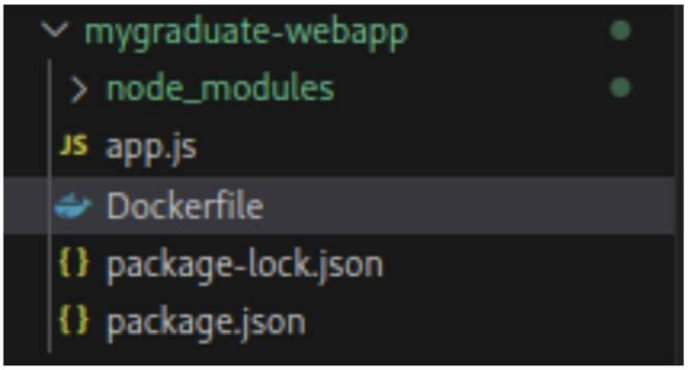

Рис. 2.6. Інфраструктура сайту

У середовищі Docker залежності встановлюються під час побудови образу контейнера за допомогою команди npm install, яка використовує дані з package.json. Це гарантує, що всі необхідні модулі будуть доступні в контейнері, як показано на рисунку 2.7.

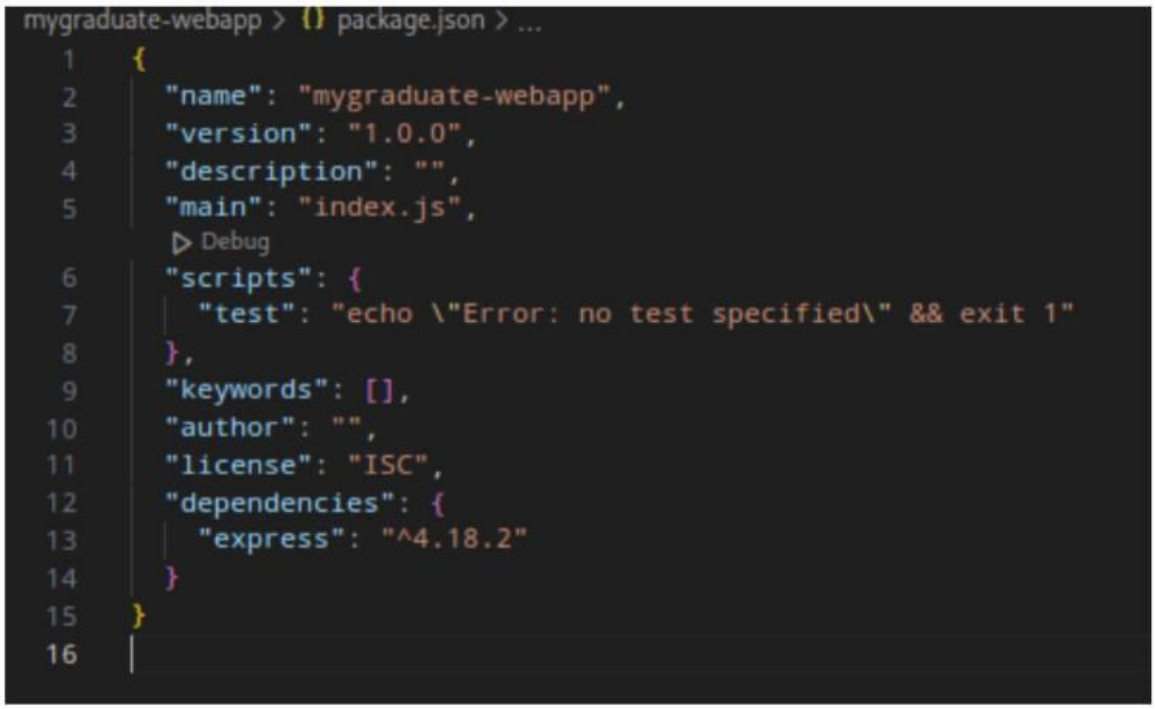

Рис. 2.7. Конфігурація package.json.

Процес створення контейнерного образу реалізується через команди в Dockerfile. Вони забезпечують копіювання потрібних файлів до контейнера, встановлення залежностей та налаштування середовища для запуску застосунку.

Кінцевий образ містить усе необхідне для функціонування веб-сайту, включаючи програму, її залежності та конфігурацію.

Для створення Docker-контейнера для веб-сайту було використано наступний Dockerfile, як представлено на рисунку 2.8.

```
# Використання легкого базового образу
FROM node:14-alpine as builder
WORKDIR /usr/src/app

# Встановлення залежностей
COPY package*.json ./
RUN npm install

# Копіювання вихідного коду
COPY . .

# Багатоетапна збірка: Копіювання лише необхідних файлів у кінцевий образ
FROM node:14-alpine
WORKDIR /usr/src/app
COPY --from=builder /usr/src/app .

EXPOSE 8080
CMD ["node", "app.js"]
```

Рис. 2.8. Конфігурація Dockerfile

Інструкція FROM визначає базовий образ контейнера, у цьому випадку — node:14, який забезпечує необхідне середовище для запуску веб-сайту на Node.js 14.

Команда WORKDIR встановлює робочий каталог /usr/src/app, де зберігатимуться всі файли веб-сайту.

Інструкція COPY копіює файли package.json і package-lock.json з локальної файлової системи в контейнер, оскільки вони містять перелік необхідних залежностей для веб-сайту.

Команда RUN виконує npm install, встановлюючи всі залежності проєкту, що є необхідним для його коректної роботи в контейнері.

Друга COPY копіює всі інші файли та каталоги до контейнера, включаючи вихідний код, дані та конфігураційні файли веб-сайту.

Інструкція EXPOSE відкриває порт 8080 для доступу до веб-сайту ззовні.

Команда CMD ["node", "app.js"] визначає команду запуску веб-додатку всередині контейнера.

Цей Dockerfile створений для забезпечення швидкого та зручного розгортання веб-сайту в хмарному середовищі, включаючи всі необхідні файли та залежності для його роботи.

У ході оптимізації Dockerfile для веб-додатку на Node.js було здійснено ряд змін, спрямованих на скорочення розміру контейнерного образу та підвищення ефективності. Замість стандартного образу node:14 використано його полегшену версію – node:14-alpine, яка містить тільки необхідні пакети, що дозволяє значно зменшити загальний розмір образу.

Щоб максимально ефективно використовувати кеш Docker, команди копіювання файлів та встановлення залежностей були розділені на окремі шари. Це дає змогу уникнути зайвого перевстановлення залежностей npm при зміні вихідного коду, скорочуючи час збірки при повторних зборах контейнера.

Крім того, застосовано багатоетапну збірку, у якій проміжний контейнер використовується для встановлення залежностей, а в остаточний образ копіюються лише необхідні файли. Такий підхід дозволяє суттєво скоротити кінцевий розмір образу, залишаючи лише ті компоненти, які потрібні для стабільної роботи застосунку.

Для розгортання веб-сайту в Kubernetes використовується Docker-образ, створений на основі Dockerfile. Для управління процесом встановлення застосунку використовується Helm – менеджер пакетів для Kubernetes, що відповідає за упаковку, конфігурацію та розгортання контейнеризованих сервісів.

Helm chart – це набір файлів, які описують необхідні ресурси Kubernetes для розгортання застосунку або сервісу (рис. 2.10). До стандартних компонентів Helm chart входять [22]:

- Chart.yaml – містить мета-інформацію про chart, включаючи його назву, версію та опис.
- Values. yaml – файл конфігураційних параметрів, що можуть бути змінені користувачем під час розгортання.
- Templates – каталог із шаблонами ресурсів Kubernetes (Deployments, Services, Ingresses, ConfigMaps тощо), що використовують значення з values.yaml.
- _helpers.tpl – набір шаблонів і функцій, які можна багаторазово використовувати у chart.
- Dependencies – визначені залежності від інших charts, що можуть бути описані у Chart.yaml або в requirements.yaml.
- Tests – містить тести для перевірки коректності розгортання Helm chart після встановлення.

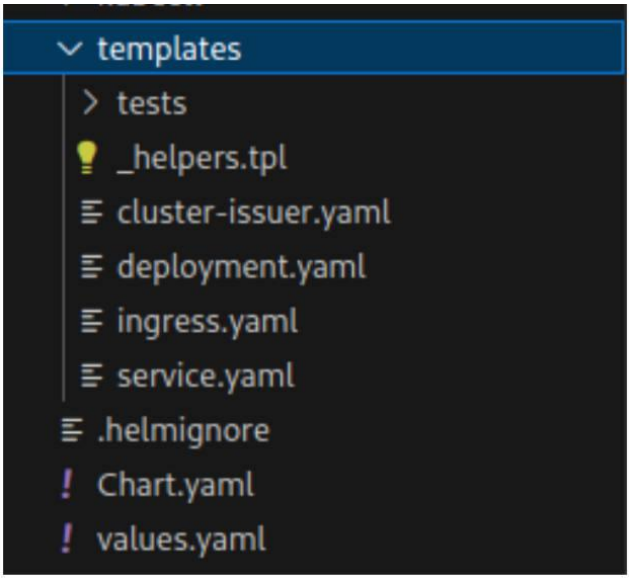

Рис. 2.10. Структура helm чарту

Helm спрощує процес розгортання Kubernetes-застосунків, забезпечуючи гнучке конфігурування та ефективне управління їх ресурсами.

Для визначення Docker-образу веб-сайту в Helm-чарті використовується відповідна конфігурація у файлі values.yaml, який містить усі параметри для розгортання Helm-чартів (рис 2.11).

Рис. 2.11. Інструкція для визначення Docker-образу

У цьому налаштуванні зазначено, що контейнерний образ веб-додатку розміщується в репозиторії gcr.io/ascendant-talon-405710/mygraduate-webapp, а його версія встановлена як "latest". Політика завантаження контейнера визначена як IfNotPresent, що означає, що образ буде завантажено лише за відсутності локальної копії.

2.5. Забезпечення безпеки в Kubernetes

Забезпечення безпеки є ключовим аспектом будь-якої хмарної інфраструктури, і Kubernetes не є винятком. Платформа пропонує широкий набір засобів для захисту додатків і даних.

Основні принципи безпеки Kubernetes включають [18]:

- централізоване управління, яке забезпечує контроль над усіма ресурсами, включаючи засоби безпеки, що спрощує адміністрування та підвищує прозорість;
- ізоляцію контейнерів, що запобігає впливу одного контейнера на інший;
- різноманітні механізми контролю доступу, які дозволяють обмежувати доступ до ресурсів.

Контроль доступу (Access Control, AC) є одним із найважливіших компонентів безпеки в Kubernetes. Платформа пропонує кілька механізмів управління доступом, що допомагають захистити ресурси від несанкціонованого використання.

Одним із найбільш ефективних методів управління доступом у Kubernetes є рольова модель (Role-Based Access Control, RBAC). RBAC дозволяє задавати ролі з відповідними дозволами, які потім можна призначати окремим користувачам або їх групам.

Наприклад, можна створити роль із правами на запуск і створення контейнерів та призначити її тим, хто відповідає за розгортання додатків.

Ще одним механізмом контролю доступу є мережеві політики (Network Policies). Вони дають змогу визначати правила взаємодії між контейнерами, регулюючи потік даних.

Наприклад, можна налаштувати мережеву політику, що блокує весь вхідний трафік до контейнерів, забезпечуючи їх захист від несанкціонованого доступу.

Kubernetes пропонує додаткові засоби безпеки для захисту додатків і даних. До них належать [19]:

- ✓ Сканування безпеки – процес перевірки контейнерів на вразливості, що дозволяє виявляти та усувати потенційні загрози.
- ✓ Автоматичне відновлення – механізм, що забезпечує автоматичне перезапускання контейнерів у разі їхнього збою.
- ✓ Моніторинг безпеки кластера – засіб для спостереження за станом та активністю, що допомагає оперативно виявляти аномалії та реагувати на можливі ризики.

У даному проекті особливу увагу було приділено забезпеченню безпеки як на рівні додатків, так і на рівні всього кластера. Ось ключові методи та стратегії, що були застосовані [24]:

- ✓ Role та RoleBinding – ефективні механізми управління доступом, що дозволяють контролювати, які користувачі або процеси мають дозвіл на виконання певних дій.

- ✓ Role – об'єкт Kubernetes, що визначає права доступу до ресурсів у конкретному namespace. Він містить набір правил, що регулюють виконання операцій.
- ✓ RoleBinding – механізм призначення ролей користувачам, групам або сервісним обліковим записам, що встановлює взаємозв’язок між ролями та їхніми власниками.

На першому етапі створюється Role, що визначає дозволи, після чого відбувається його прив’язка до користувачів або служб, які потребують відповідного доступу (рис. 2.12).

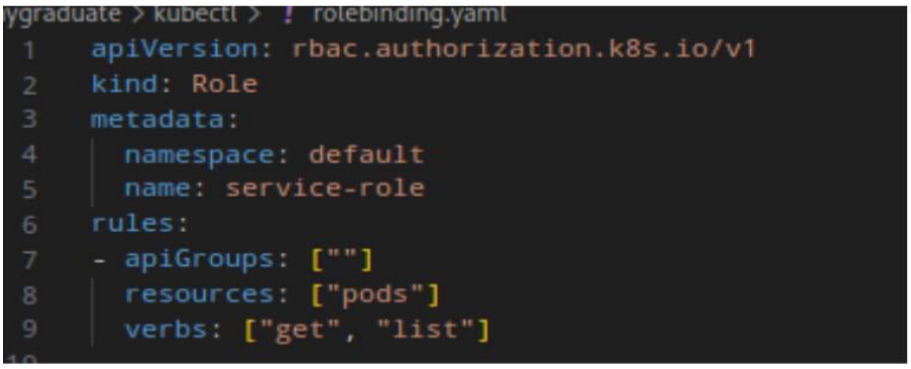

Рис. 2.12. Конфігурація role.yaml

Далі створюється RoleBinding, який прив’язує раніше визначену роль до конкретного користувача або сервісного облікового запису (рис. 2.13).

Рис. 2.13. Конфігурація rolebinding.yaml

На основі наведених параметрів у проекті було створено необхідні Role та RoleBindings, які впроваджено у кластер за допомогою команди: kubectlapply-f rolebinding.yaml

Ці механізми забезпечують більш безпечне та контрольоване середовище для роботи з Kubernetes, обмежуючи доступ лише до потрібних ресурсів для кожного користувача чи процесу.

У Kubernetes також використовується Secrets для безпечного зберігання конфіденційної інформації, зокрема паролів, OAuth-токенів та SSH-ключів. Завдяки цій функції чутливі дані можуть бути інтегровані у додатки без необхідності зберігати їх у коді або конфігураційних файлах.

Створення секретів здійснюється за допомогою команди: kubectl create secret або через YAML-файл. Наприклад, можна створити Secret для зберігання пароля до бази даних (рис. 2.14).

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

Рис. 2.14. Конфігурація secret.yaml

У цьому випадку параметр type: Opaque вказує на те, що пароль має бути зашифрований у форматі base64.

У конфігурації деплойменту Kubernetes секрети можуть використовуватися для налаштування змінних середовища або збереження конфіденційних даних у файлах всередині подів (рис. 2.15).

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

Рис. 2.15. Передача значень secret.yaml у values.yaml

Оскільки в проєкті використовується Helm, передача параметрів здійснюється через файл values.yaml.

З точки зору мережевої безпеки необхідно налаштувати Network Policies для обмеження доступу між подами в кластері, що запобігає несанкціонованому трафіку. Network Policies дозволяють визначати правила взаємодії між подами, наприклад, обмежити доступ до бази даних тільки для певних подів (рис. 2.16).

```
1  apiVersion: networking.k8s.io/v1
2  kind: NetworkPolicy
3  metadata:
4    name: db-network-policy
5  spec:
6    podSelector:
7      matchLabels:
8        role: database
9    policyTypes:
10   - Ingress
11   ingress:
12   - from:
13     - podSelector:
14         matchLabels:
15           app: web
```

Рис. 2.16. Конфігурація networkpolicy.yaml

Після створення Network Policy вона застосовується в Kubernetes, регулюючи трафік відповідно до встановлених правил. Це дає змогу точно контролювати, які сервіси мають доступ до ресурсів усередині кластера, що значно підвищує рівень безпеки.

Поєднання Secrets, Network Policies, Role та RoleBindings забезпечує контрольований доступ до конфіденційних даних і мережевих ресурсів, підвищуючи загальний рівень безпеки застосунку та даних, якими він оперує. Ці інструменти є

важливими для формування надійного та захищеного середовища в Kubernetes-кластері.

Захист від DDoS-атак здійснюється через такі заходи [21]:

- ✓ Обмеження кількості запитів – встановлення лімітів на обробку запитів подами протягом певного часу, що дозволяє запобігти перевантаженню системи.
- ✓ Використання зовнішнього балансування навантаження – інтеграція з сервісами балансування трафіку, які здатні розпізнавати та нейтралізувати вплив DDoS-атак.
- ✓ Шифрування переданих даних – застосування механізмів шифрування для захисту інформації від несанкціонованого доступу та кібератак.

Ці заходи забезпечують підвищену стійкість інфраструктури Kubernetes до загроз, створюючи захищене середовище для веб-додатків.

Для контролю кількості запитів, які можуть бути оброблені подами, Kubernetes підтримує Resource Quotas та Limit Ranges. Вони допомагають обмежити споживання ресурсів і запобігти надмірному навантаженню.

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: api-limit-range
spec:
  limits:
    - type: Pod
      max:
        cpu: "1"
        memory: "1Gi"
    - type: Container
      defaultRequest:
        cpu: "500m"
        memory: "500Mi"
```

Рис. 2.17. Конфігурація limitrange.yaml

Наприклад, рис. 2.17 демонструє відповідний код, що встановлює максимальні та стандартні ліміти ресурсів для подів і контейнерів.

РОЗДІЛ 3. РОЗГОРТАННЯ ТА АДМІНІСТРУВАННЯ ВЕБ-САЙТУ

3.1. Деплоймент веб-додатку в середовищі Kubernetes

Для розгортання веб-додатку в Kubernetes було використано Helm, менеджер пакетів, який спрощує розгортання та управління складними застосунками за допомогою єдиного YAML-файлу.

Для створення Helm-чарта було сформовано каталог my-graduate-webapp, що містить такі файли:

- Chart.yaml – визначає основні параметри чарта, включаючи назву, версію та опис.

Цей фрагмент коду містить метадані Helm-чарта під назвою «mygraduate». Використання apiVersion v2 підтверджує сумісність із Helm 3. Опис «A Helm chart for Kubernetes» пояснює його призначення – розгортання додатку в Kubernetes.

Тип application вказує, що цей чарт орієнтований на розгортання веб-додатків, а не бібліотек чи службових компонентів. Версія чарта 0.1.0 сигналізує про початкову стадію розробки, а версія додатку latest відповідає останньому доступному релізу (рис. 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"
```

Рис. 3.1. Конфігурація Chart.yaml

- Нижче представлено `service.yaml` – файл, що визначає ресурс Service для веб-додатку.

Цей фрагмент описує Service у Kubernetes у межах Helm-чарта "mygraduate". Спочатку вказується версія API та тип ресурсу, після чого визначається ім'я сервісу, що формується динамічно через шаблон Helm.

Метадані містять мітки (labels), які також задаються через шаблон. У spec визначається тип сервісу, який може бути налаштований через параметри у файлі values.yaml.

Налаштування портів включає параметри для прийому трафіку, де визначається порт, на якому сервіс слухає вхідні запити, цільовий порт у подах, а також використаний протокол (TCP) і назва порту.

Selector застосовується для прив’язки сервісу до конкретних подів, визначаючи, на які саме поди буде спрямовуватися трафік (рис. 3.2).

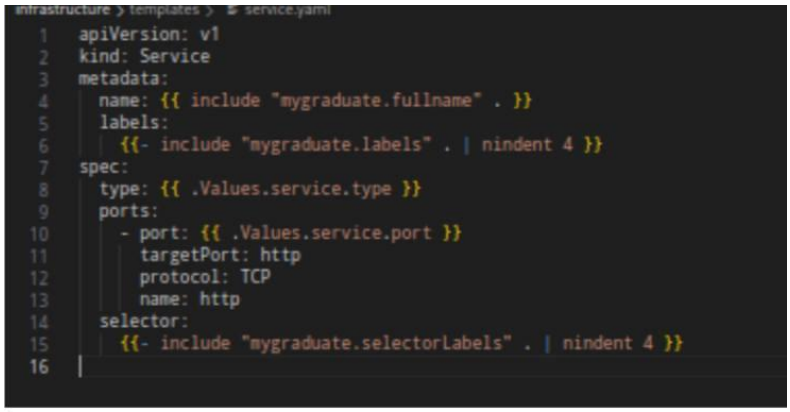

Рис. 3.2. Конфігурація service.yaml

- Нижче наведено `Deployment.yaml` – файл, що визначає ресурс Deployment для веб-додатку.

Цей фрагмент описує Deployment у Kubernetes як частину Helm-чарта «mygraduate», що регулює розгортання та управління додатком у кластері.

Deployment починається із зазначення версії API та типу ресурсу. У метаданих включено динамічно згенеровану назву Deployment та мітки, які формуються за допомогою шаблону Helm.

У spec вказується кількість реплік поду, які мають бути створені, якщо автоматичне масштабування не увімкнено. Селектор matchLabels визначає, які саме поди перебуватимуть під контролем Deployment.

Далі йде опис template – шаблону поду, що містить анотації та мітки, побудовані з використанням динамічних значень. У spec поду визначаються параметри imagePullSecrets для доступу до приватних реєстрів контейнерів, serviceAccountName для прив'язки до відповідного облікового запису служби, а також контекст безпеки.

У розділі конфігурації контейнера вказані:

- Образ контейнера та політика його завантаження
- Параметри портів
- Проби готовності та живучості
- Ресурсні ліміти
- Налаштування змінних середовища для підключення до бази даних, використовуючи значення зі Secrets Kubernetes

Також передбачені nodeSelector, affinity та tolerations для контролю розміщення подів у кластері (рис. 3.3).

Рис. 3.3. Конфігурація deployment.yaml

- Наступний файл – ingress.yaml, що визначає ресурс Ingress для веб-додатку.

Цей фрагмент містить умову, яка активує конфігурацію Ingress, якщо у файлі values.yaml відповідний параметр встановлено як enabled. Він описує ресурс Ingress у Kubernetes, який використовується у межах Helm-чарта «mygraduate» для керування доступом зовнішніх користувачів до сервісів всередині кластера.

Конфігурація починається з визначення apiVersion і типу ресурсу, після чого задається назва Ingress, що формується динамічно через шаблон Helm. Мітки (labels) та анотації (annotations) також налаштовуються за допомогою шаблонів.

Специфікація Ingress включає ім’я класу Ingress та набір правил, які визначають, як запити до певних хостів та шляхів мають оброблятися. Кожен шлях прив’язується до певного сервісу в кластері та відповідного порту, що забезпечує коректний розподіл трафіку.

Якщо TLS увімкнено, для Ingress також налаштовуються відповідні параметри, включаючи список хостів та секрети, що містять SSL-сертифікати.

Цей фрагмент демонструє, як Helm надає гнучкість у керуванні складними конфігураціями, такими як Ingress, дозволяючи легко активувати або вимикати окремі компоненти залежно від потреб розгортання (рис. 3.4).

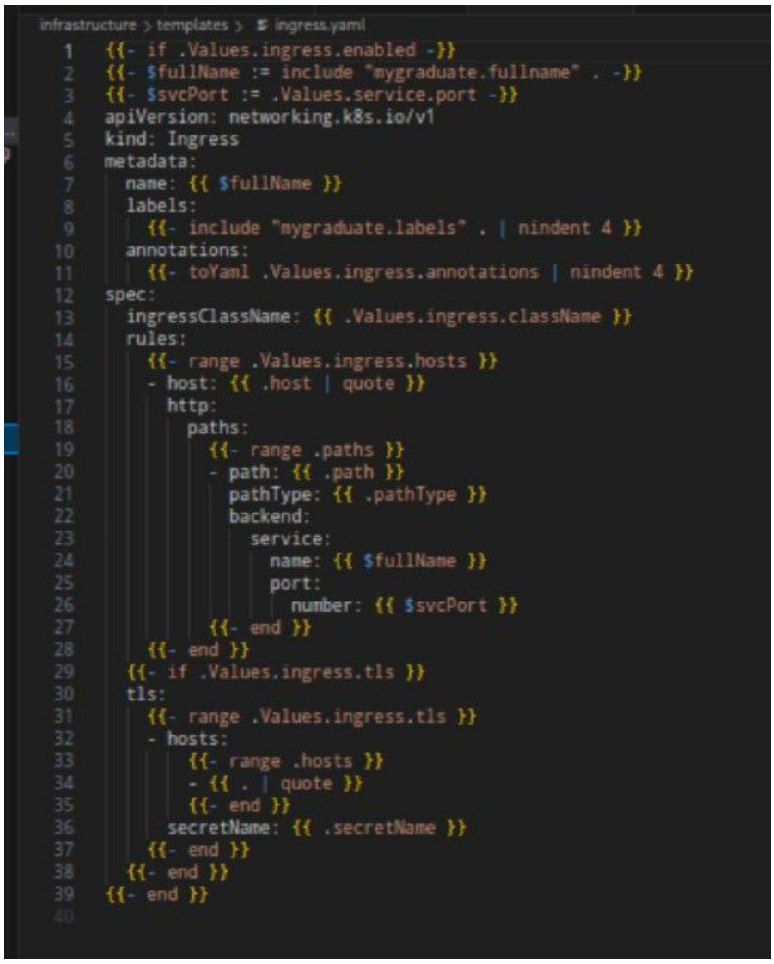

Рис. 3.4. Конфігурація ingress.yaml

Файл values.yaml містить параметри конфігурації для Helm-чарта, що використовуються для керування розгортанням та налаштуванням додатка в Kubernetes:

- replicaCount: задає кількість реплік подів для Deployment (наприклад, `1`).
- image: містить налаштування образу контейнера, включаючи:
- `repository` – шлях до контейнера,
- `tag` – версія образу,
- `pullPolicy` – політика завантаження.
- imagePullSecrets: пустий масив (`[]`), що означає, що секрети для завантаження образів не використовуються.
- nameOverride / fullnameOverride: параметри для перевизначення стандартних назв ресурсів.
- serviceAccount: визначає, чи потрібно створювати новий обліковий запис сервісу, його анотації та назву.
- database: налаштування підключення до бази даних, включаючи хост, користувача та секрети для доступу.
- podAnnotations, podSecurityContext, securityContext: параметри безпеки та анотації для подів.
- service: визначає тип сервісу Kubernetes та порт, на якому він слухає.
- ingress: містить налаштування Ingress, включаючи:
  - Активацію (`enabled`),
  - Клас (`className`),
  - Анотації,
  - Правила для хостів та шляхів,
  - Налаштування TLS.
- resources: задає обмеження та запити на ресурси для подів (CPU, пам’ять).
- autoscaling: параметри автоматичного масштабування, включаючи мінімальну та максимальну кількість реплік та цільове використання CPU.

- nodeSelector, tolerations, affinity: параметри для контролю розміщення подів у кластері (рис. 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: {}
```

Рис. 3.5. Файл values.yaml

Файл values.yaml є ключовим елементом конфігурації Helm-чарта, дозволяючи користувачам адаптувати розгортання додатків відповідно до їхніх потреб.

Для розгортання Helm-чарта було виконано команду:helm install my-graduate-webapp my-graduate-fvalues.yaml. Ця команда ініціалізувала Helm-чарт у кластері та розгорнула його (рис. 3.6).

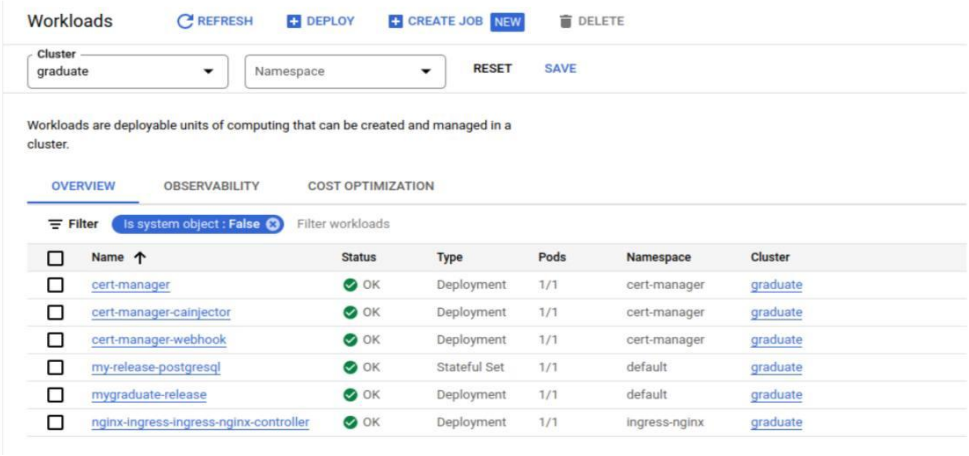

Рис. 3.6. Список workloads в Kubernetes

ClusterIP використовується для веб-додатка, що означає, що сервіс доступний лише з інших подів у кластері Kubernetes. Порт 8080 веб-додатка був зіставлений з портом 80 сервісу (рис. 3.7).

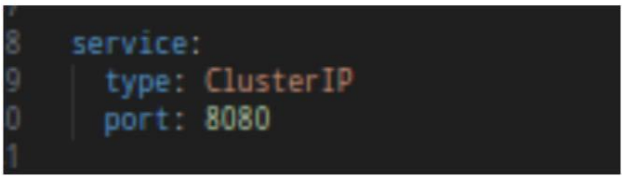

Рис. 3.7. Налаштування для service.yaml

Для доступу до веб-додатку було налаштовано Ingress, що дозволяє використання хоста mygraduate.theraven.tech (рис. 3.8).

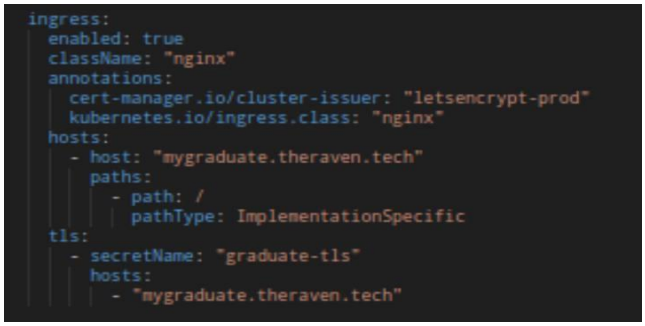

Рис. 3.8. Налаштування для ingress.yaml

Ingress інтегрований з cert-manager cluster-issuer для автоматичного управління SSL-сертифікатами (рис. 3.9).

```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
```

Рис. 3.9. Конфігурація cluster-issuer.yaml

Додаткові налаштування включають:

- Liveness та Readiness probes – дозволяють Kubernetes відстежувати стан веб-додатку та перезапускати його в разі необхідності.
- YAML-шаблони застосовані для налаштування Helm-чарта у різних середовищах.

Таким чином, ця конфігурація забезпечує масштабованість, безпеку та гнучкість розгортання веб-додатку в Kubernetes.

3.2. Інтеграція з системами управління базами даних

У сфері розробки веб-додатків PostgreSQL є одним із провідних рішень серед систем управління базами даних, перевершуючи популярні альтернативи, такі як MySQL та MongoDB. Його основна перевага полягає у розширеній підтримці стандартів SQL, надійності та можливості виконання складних операцій із даними. Крім того, PostgreSQL підтримує індексування JSON, що робить його особливо зручним для сучасних веб-додатків, де обмін даними відбувається через JSON.

У порівнянні з MySQL, PostgreSQL пропонує гнучкіші та потужніші механізми управління транзакціями, а також кращу підтримку паралельних запитів.

З точки зору забезпечення високої доступності, PostgreSQL має розвинуту підтримку реплікації даних, яка може бути налаштована як у синхронному, так і у асинхронному режимах, що часто перевершує аналогічні можливості MySQL та інших СУБД. Це дозволяє створювати надійні та відмовостійкі системи.

Для резервного копіювання PostgreSQL пропонує гнучкі інструменти, такі як pg_dump та pg_basebackup, що гарантують ефективне та безпечне збереження даних. На відміну від MongoDB, PostgreSQL використовує традиційні та перевірені методи резервного копіювання, що може бути критично важливим для додатків, де втрата даних неприпустима.

Вибір PostgreSQL надає розробникам розширені можливості для створення ефективних, надійних та безпечних веб-додатків, що вимагають інтенсивної роботи з даними.

Базу даних можна розгорнути безпосередньо у Kubernetes як окремий під або скористатися зовнішніми хмарними сервісами. Якщо PostgreSQL розгортається в Kubernetes, використовується Docker-образ, наприклад, офіційний PostgreSQL image із Docker Hub (рис. 3.10).

mygraduate > kubectl > ! postgres.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  containers:
    - name: postgres
      image: postgres:latest
      env:
        - name: POSTGRES_DB
          value: mydatabase
        - name: POSTGRES_USER
          value: user
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-password
              key: password
```

Рис. 3.10. Deployment.yaml для бази даних PostgreSQL

У Deployment.yaml змінні середовища для підключення до бази (`DATABASE_HOST`, DATABASE_USER, `DATABASE_PASSWORD`) задаються у специфікації контейнера (рис. 3.11).

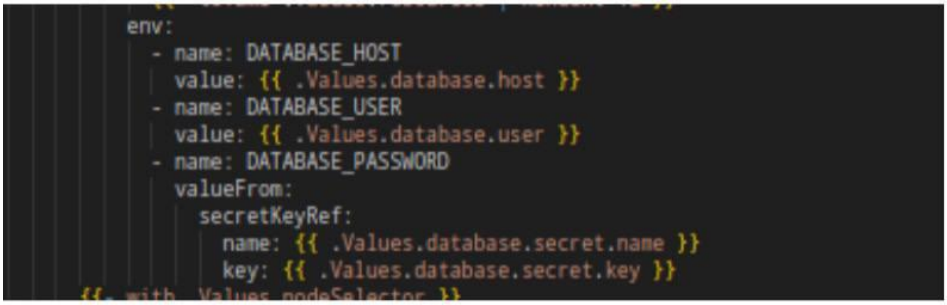

Рис. 3.11. Конфігурація бази даних в Deployment.yaml для helm чарта

Ці змінні отримують значення з values.yaml, а паролі зберігаються у Kubernetes Secret для підвищення безпеки (рис. 3.12).

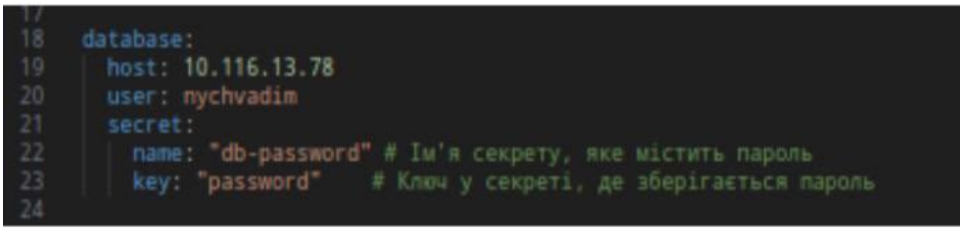

Рис. 3.12. Конфігурації бази даних у файлі values.yaml

Інтеграція з secret.yaml дозволяє отримувати DATABASE_PASSWORD безпосередньо з секрету, що гарантує захищене зберігання конфіденційної інформації (рис. 3.13).

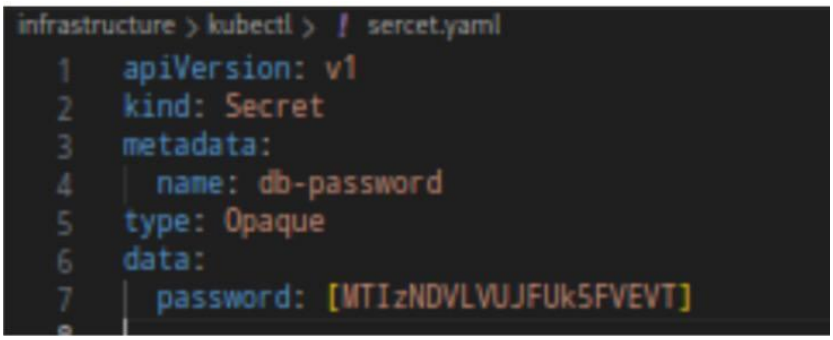

Рис. 3.13. Файл Secret.yaml

Застосування Kubernetes Secret (`db-password`) дозволяє уникнути жорсткого кодування чутливих даних у контейнерних образах або конфігураціях.

Service у Kubernetes – це ключова абстракція, яка забезпечує доступ до додатків, розгорнутих у подах, через єдиний адресований ресурс. У випадку my-release-postgresql, Service виконує кілька важливих функцій (рис. 3.14):

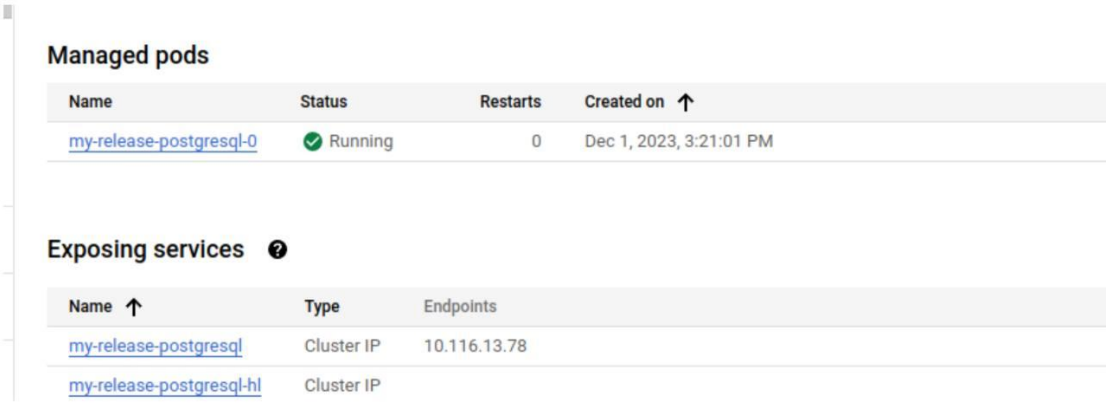

Рис. 3.14. Services для PostreSQL

- Стабільна IP-адреса та DNS-ім'я – незалежно від розташування пода з PostgreSQL, підключення до нього здійснюється через постійну адресу.
- Балансування навантаження – якщо база працює у декількох подах для реплікації або масштабування, Service рівномірно розподіляє запити між ними.

Таким чином, PostgreSQL у Kubernetes забезпечує стабільність, масштабованість та високу доступність, що критично важливо для сучасних веб-додатків.

3.3. Вирішення проблем, що виникають під час розгортання

У процесі розгортання веб-додатку в Kubernetes за допомогою Helm-чартів було виявлено кілька викликів, які вдалося ефективно вирішити. Ось основні з них:

- Проблема з сервісними обліковими записами.

Під час розгортання виникли труднощі через відсутність необхідного сервісного облікового запису. Для виправлення ситуації було оновлено конфігурацію Helm-чарта, щоб використовувався існуючий сервісний обліковий запис або щоб чарт не створював новий.

- Проблеми з доступністю Docker-образу.

Виникли обмеження доступу до Docker-образу додатку в приватному реєстрі, що ускладнювало процес розгортання.

Щоб усунути цю проблему, образ було перенесено до загальнодоступного реєстру, такого як Google Container Registry, що забезпечило стабільний доступ до контейнера під час деплойменту (рис. 3.15).

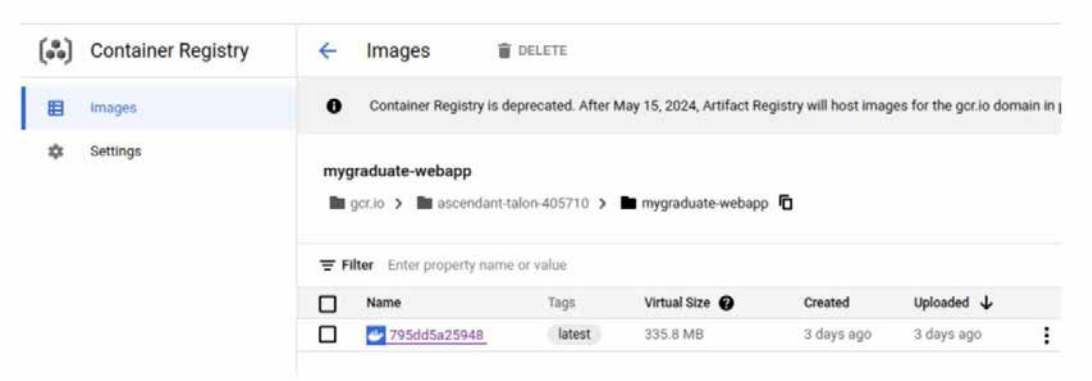

Рис. 3.15. Інтерфейс Google Container Registry

Завдяки цим оптимізаціям розгортання веб-додатку стало безпечнішим та більш контрольованим, спрощуючи процес інтеграції у Kubernetes.

Під час налаштування SSL/TLS для веб-додатку виникли складнощі, зокрема проблеми з конфігурацією Ingress та сертифікатів. Для їх вирішення було використано Cert-Manager, що дозволило автоматизувати управління сертифікатами SSL/TLS. Крім того, Ingress було налаштовано таким чином, щоб коректно вказувати на ці сертифікати (рис. 3.16).

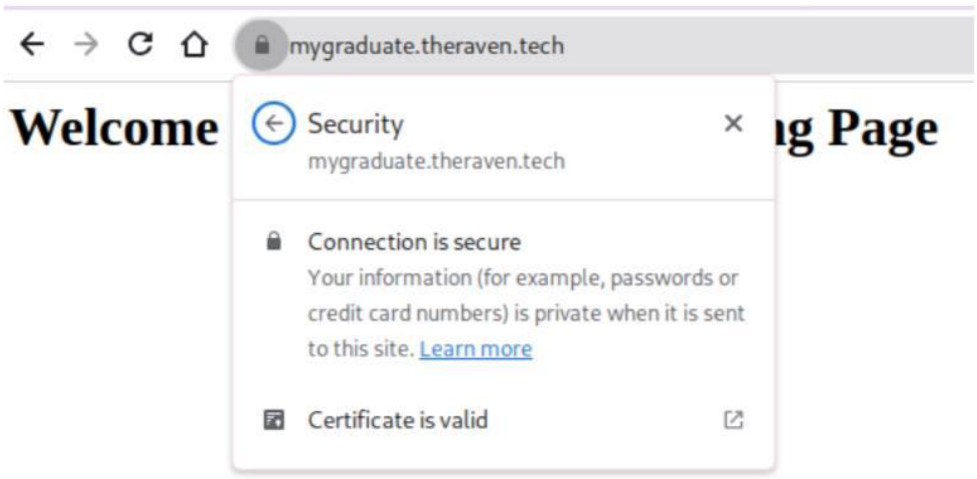

Рис. 3.16. Результат використання CertManager

Серед проблем, що виникли під час роботи з мережею та Ingress, було некоректне маршрутизування трафіку до додатку.

Конфігурація Ingress була перероблена, з правильною специфікацією шляхів та сервісів, а також інтеграцією з SSL-сертифікаціями (рис. 3.17).

```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"
```

Рис. 3.17. Інструкція для ingress.yaml з values.yaml

Ці випадки демонструють гнучкість у вирішенні проблем, характерних для розгортання сучасних веб-додатків у Kubernetes. Успішне впровадження рішень стало можливим завдяки глибокому розумінню технічних аспектів Kubernetes та Helm, а також здатності швидко реагувати на непередбачені виклики.

ВИСНОВКИ

У ході дослідження було проведено комплексний аналіз технологій хмарних обчислень і контейнеризації, що дозволило визначити їхні ключові переваги та виклики. Було розглянуто сучасні хмарні платформи, серед яких Kubernetes виявився оптимальним вибором завдяки своїй масштабованості, гнучкості та розвинутим механізмам управління контейнерами.

У другому розділі було спроектовано хмарну інфраструктуру, включаючи вибір провайдера хмарних послуг, налаштування Kubernetes-кластеру та забезпечення безпеки. Це забезпечило надійне та кероване середовище, придатне для розгортання веб-додатків. Особливу увагу було приділено контейнеризації за допомогою Docker, що спростило управління застосунками.

Третій розділ був присвячений розгортанню веб-додатку, інтеграції з системами керування базами даних та усуненню проблем, що виникли в процесі розгортання. Було протестовано використання Helm-чартів для автоматизації деплойменту, реалізовані механізми автоматичного масштабування та балансування навантаження, а також впроваджені засоби захисту веб-додатку.

Загалом, проведене дослідження підтвердило ефективність застосування Kubernetes для масштабованих веб-додатків та продемонструвало переваги контейнеризованого підходу в управлінні інфраструктурою. Виконані завдання дозволили не лише глибше зрозуміти принципи роботи Kubernetes, а й на практиці застосувати методи адміністрування та забезпечення безпеки веб-додатків у хмарному середовищі.

Отримані результати можуть бути використані для оптимізації та вдосконалення хмарних веб-рішень, а також слугувати основою для подальших досліджень у сфері контейнеризації та масштабування веб-додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Архітектура хмарних додатків: Побудова масштабованих танадійних систем. Том Ласковіч. Видавництво «O'Reilly Media», 2020. - 310 с.

2. Бази даних: Повне керівництво для розробників. Пол Вілтон. Видавництво «Addison-Wesley», 2018. - 350 с.

3. Використання Helm для управління Kubernetes: Ефективні стратегії. Метт Батчер. Видавництво «O'Reilly Media», 2019. - 270 с.

4. Майстерність Kubernetes: Розширений підхід до розгортання та управління. Джессі Фразель. Видавництво «Addison-Wesley», 2021. - 300 с.

5. Модернізація існуючих додатків з Docker та Kubernetes. Джеймі Данкан. Видавництво «Packt Publishing», 2020. - 300 с.

6. Основи хмарних обчислень: Принципи та практики. Кевін Л. Джексон. Видавництво «McGraw-Hill Education», 2021. - 360 с.

7. Основи Kubernetes: Розгортання і управління сучасними додатками. Найджел Поултон. Видавництво «Packt Publishing», 2019. - 280 с.

8. Поглиблене вивчення Kubernetes: Занурення в архітектуру та внутрішню реалізацію. Брендан Бернс. Видавництво «O'Reilly Media», 2022. - 340 с.

9. Продуктивність баз даних: Оптимізація та масштабування. Сімона Бруно. Видавництво «Apress», 2019. - 250 с.

10. Професійна робота з CI/CD: Використання Jenkins, Docker, і Kubernetes. Сара Міллер. Видавництво «Manning Publications», 2020. - 320 с.

11. Професійне використання Docker: Від розробки до продакшену. Елтон Стоунемен. Видавництво «Manning Publications», 2020. - 280 с.

12. Розробка хмарних додатків з використанням Microsoft Azure. Скотт Гатрі. Видавництво «Microsoft Press», 2021. - 320 с.

13. Розробка хмарних додатків з Google Cloud Platform. Джорджія Костара. Видавництво «O'Reilly Media», 2021. - 330 с.

14. Системи управління базами даних: Теорія та практика. Рене Ф. Кабрера. Видавництво «Cambridge University Press», 2020. - 380 с.

15. Сучасні бази даних: Менеджмент та оптимізація. Роберт Мейер. Видавництво «McGraw-Hill Education», 2019. - 300 с.

16. Хмарні обчислення: Принципи, системи та застосування. Нік Антонопулос. Видавництво «Springer», 2020. - 400 с.

17. Amazon Web Services в дії. Майкл Віт, Андреас Віттіг. Видавництво «Manning Publications», 2021. - 450 с.

18. Amazon Web Services: Практичний підхід до розгортання та управління. Елісон Сміт. Видавництво «Packt Publishing», 2020. - 290 с.

19. Azure для розробників: Все, що потрібно знати. Джонатан Туліс. Видавництво «O'Reilly Medi», 2019. - 300 с.

20. CI/CD із Docker та Kubernetes: Повне керівництво. Джонатан Баер. Видавництво «Packt Publishing», 2020. - 310 с.

21. GitHub Actions: Автоматизація процесів розробки. Майкл Коффман. Видавництво «Apress», 2021. - 280 с.

22. Helm: Ефективне управління Kubernetes додатками. Ендрю Блок. Видавництво «Packt Publishing», 2019. - 250 с.

23. Kubernetes: Повний курс з нуля до профі. Джеймс Лі. Видавництво «O'Reilly Media», 2020. - 450 с.

24. Microsoft Azure: Керівництво для розробників. Деніел Баскет. Видавництво «Apress»,2021.-320с.