МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет економіки, підприємництва та інформаційних технологій

Кафедра інформаційних технологій

Допущено до захисту

“_____” _____ 2026 р.

Зав. кафедри, к.т.н., доцент

_____ **Ліщук Р. І.**

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА ДОДАТКУ ДЛЯ УПРАВЛІННЯ ХМАРНИМИ РЕСУРСАМИ НА БАЗІ ТЕХНОЛОГІЙ КОНТЕЙНЕРИЗАЦІЇ

Виконав – студент 4 курсу, групи 41-з- кн
освітньої програми «Комп'ютерні науки»
спеціальності 122 Комп'ютерні науки

ПРОКОПЧУК Богдан Ігорович

Науковий керівник – к.т.н., доцент

ЛІЩУК Роман Ігорович

Умань – 2026

ЗМІСТ

ВСТУП …… 4

РОЗДІЛ 1. АНАЛІЗ СТАНУ РОЗРОБКИ ЗАСОБІВ УПРАВЛІННЯ ХМАРНИМИ РЕСУРСАМИ НА БАЗІ ТЕХНОЛОГІЙ КОНТЕЙНЕРИЗАЦІЇ …….. 6

1.1. Поняття та особливості хмарних обчислень ……... 6

1.2. Управління й оркестрація контейнеризованих застосунків …….. 8

1.3. Принцип Infrastructure as Code (IaC) в управлінні хмарною інфраструктурою …… 10

1.4. Технології контейнеризації та їх значення для сучасних інформаційних систем ……..…… 12

1.5. Моніторинг, телеметрія та використання хмарних сервісів для управління ресурсами ……..……. 13

РОЗДІЛ 2. ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ УПРАВЛІННЯ ХМАРНИМИ РЕСУРСАМИ ……..…….. 15

2.1. Аналіз вимог до програмного додатку ……. 15

2.2. Обґрунтування вибору технологій та програмних засобів …… 18

2.3. Проєктування архітектури системи …….. 24

2.4. Інструменти для моніторингу та телеметрії ……... 30

2.5. Огляд хмарних провайдерів …….. 32

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ …… 36

3.1. Впровадження хмарних сервісів …… 36

3.2. Контейнеризація інформаційної системи ……..…. 40

3.3. Інтеграція контейнеризації та засобів управління ресурсами ….. 42

3.4. Інтеграція системи моніторингу …… 47

3.5. Програмна реалізація та аналіз результатів розгортання інфраструктури в Kubernetes-кластері ……. 49

ВИСНОВКИ…….. 56

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…… 57

ДОДАТКИ…… 61

ВСТУП

Актуальність дослідження. Актуальність теми дослідження зумовлена стрімким розвитком хмарних технологій та зростанням потреби в ефективному управлінні обчислювальними ресурсами. Сучасні інформаційні системи потребують гнучких, масштабованих і надійних рішень для розгортання та підтримки програмних сервісів, що обумовлює широке використання технологій контейнеризації. Контейнерні платформи забезпечують автоматизацію процесів розгортання, масштабування та моніторингу застосунків, сприяючи оптимізації використання ресурсів хмарної інфраструктури. У зв’язку з цим розробка додатку для управління хмарними ресурсами на базі технологій контейнеризації є актуальним завданням, спрямованим на підвищення ефективності адміністрування хмарного середовища, забезпечення стабільності роботи сервісів та автоматизацію процесів управління ресурсами.

Об’єкт дослідження – процес управління хмарними ресурсами в середовищі контейнеризованих обчислень.

Предмет дослідження – методи, моделі та програмні засоби автоматизації управління, розгортання, масштабування та моніторингу хмарних ресурсів на базі технологій контейнеризації.

Для досягнення мети дослідження було поставлено та вирішено **такі завдання:**

– дослідити теоретичні засади хмарних обчислень і технологій контейнеризації;

– проаналізувати сучасні підходи до управління хмарними ресурсами та контейнеризованими застосунками;

– дослідити можливості використання технологій Docker і Kubernetes для розгортання та оркестрації контейнерів;

– проаналізувати засоби автоматизації управління хмарною інфраструктурою;

– розглянути сучасні інструменти моніторингу та контролю використання ресурсів;

– виконати порівняльний аналіз програмних засобів контейнеризації та оркестрації;

– обґрунтувати вибір технологій і програмних засобів для реалізації додатку;

– розробити додаток для управління хмарними ресурсами на базі технологій контейнеризації;

– провести тестування розробленого додатку та оцінити ефективність його функціонування.

У роботі застосовано такі методи дослідження: аналіз і узагальнення наукової та технічної літератури – для дослідження сучасних підходів до управління хмарними ресурсами та використання технологій контейнеризації; порівняльний аналіз – для оцінювання функціональних можливостей існуючих засобів контейнеризації, оркестрації та моніторингу; методи проєктування програмних систем – для розробки додатку управління хмарними ресурсами; методи моделювання та програмної реалізації – для створення функціональних компонентів системи; метод експериментального дослідження та тестування – для перевірки працездатності, надійності та оцінювання ефективності розробленого програмного рішення.

РОЗДІЛ 1. АНАЛІЗ СТАНУ РОЗРОБКИ ЗАСОБІВ УПРАВЛІННЯ ХМАРНИМИ РЕСУРСАМИ НА БАЗІ ТЕХНОЛОГІЙ КОНТЕЙНЕРИЗАЦІЇ

1.1. Поняття та особливості хмарних обчислень

Контейнеризація є сучасною технологією розробки та розгортання програмного забезпечення, яка передбачає упаковку застосунку разом із необхідними бібліотеками, конфігураціями та іншими залежностями в ізольоване середовище – контейнер. Такий підхід забезпечує стабільну роботу програмного продукту незалежно від особливостей операційної системи чи апаратної платформи.

Широке використання контейнеризації зумовлене низкою переваг, які роблять цю технологію важливим інструментом сучасної IT-індустрії:

– ефективне використання ресурсів – контейнери споживають менше обчислювальних ресурсів порівняно з традиційними віртуальними машинами, що дозволяє підвищити продуктивність системи;

– ізоляція середовища виконання – кожний контейнер функціонує незалежно від інших, що сприяє підвищенню безпеки та стабільності роботи застосунків;

– переносимість – контейнеризовані додатки можуть без змін запускатися на різних платформах і в різних середовищах, які підтримують відповідну технологію;

– автоматизація процесів – використання контейнерів спрощує розгортання, оновлення та масштабування програмних продуктів, що особливо важливо для розподілених інформаційних систем.

Порівняно з традиційною віртуалізацією, контейнеризація характеризується вищою швидкодією та меншою потребою в системних ресурсах [5-6]. На відміну від віртуальних машин, контейнери використовують спільне ядро операційної системи, що зменшує накладні витрати на підтримку окремих операційних середовищ. Завдяки цьому контейнеризовані застосунки швидше запускаються, ефективніше використовують обчислювальні ресурси та забезпечують кращу масштабованість. Саме тому контейнеризація стала однією з ключових технологій побудови сучасних хмарних та розподілених інформаційних

систем (рис. 1.1).

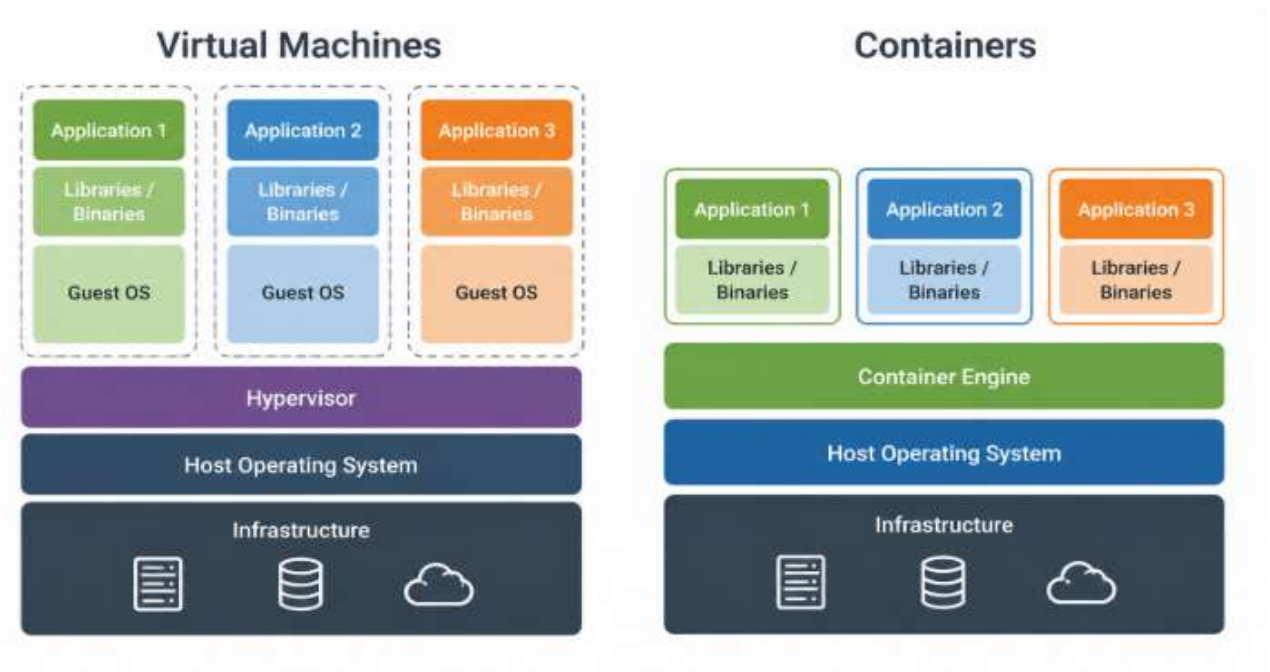


Рис. 1.1. Порівняння архітектури віртуальних машин і контейнерів

Джерело: Джерело: сформовано автором на основі [1; 21].

Контейнеризація забезпечує високий рівень переносимості програмного забезпечення, дозволяючи запускати застосунки на різних програмних і апаратних платформах без необхідності додаткового налаштування. Контейнер об’єднує всі компоненти, необхідні для функціонування програми, зокрема вихідний код, бібліотеки, конфігураційні файли та змінні середовища. Завдяки цьому програмний продукт зберігає однакову працездатність незалежно від середовища виконання, що значно спрощує процеси розгортання, оновлення та масштабування.

Важливою перевагою контейнеризації є забезпечення ізоляції застосунків [4]. Кожен контейнер працює у власному середовищі, що мінімізує ризик виникнення конфліктів між програмами та їхніми залежностями. Такий підхід підвищує рівень безпеки, надійності та стабільності функціонування інформаційних систем. Крім того, контейнерні технології спрощують резервне копіювання, перенесення та відновлення програмних сервісів, що є особливо важливим для сучасних розподілених середовищ.

Використання контейнерів також позитивно впливає на процес розробки програмного забезпечення. Завдяки уніфікованому середовищу виконання розробники можуть проводити тестування та налагодження застосунків в

умовах, максимально наближених до виробничого середовища. Це сприяє підвищенню якості програмного продукту та скороченню часу його впровадження.

Таким чином, контейнеризація є однією з ключових технологій сучасної IT-інфраструктури, оскільки поєднує високу переносимість, ефективне використання ресурсів, безпеку та зручність управління. Її застосування дозволяє оптимізувати процеси створення, розгортання та супроводу програмного забезпечення, забезпечуючи стабільну роботу розподілених інформаційних систем.

1.2. Управління й оркестрація контейнеризованих застосунків

У сучасних умовах розвитку інформаційних технологій дедалі більшого поширення набувають контейнеризовані та мікросервісні архітектури, які забезпечують гнучкість, масштабованість і швидкість розгортання програмних систем. Проте зі збільшенням кількості сервісів ускладнюється процес їхнього адміністрування, координації та підтримки. Саме тому важливого значення набувають технології управління та оркестрації, які дозволяють автоматизувати виконання багатьох операцій і забезпечити стабільне функціонування інформаційної інфраструктури.

Оркестрація контейнерів являє собою комплекс засобів і механізмів, призначених для автоматизованого розгортання, масштабування, моніторингу та керування контейнеризованими додатками. Завдяки використанню систем оркестрації значно спрощується управління великими програмними комплексами, підвищується ефективність використання ресурсів і забезпечується висока доступність сервісів.

Управління та оркестрація охоплюють процеси координації взаємодії між окремими компонентами програмної системи, контролю їхнього стану та автоматизації виконання типових адміністративних завдань. Такі підходи особливо актуальні для сучасних розподілених систем, у яких функціональність реалізується через набір незалежних сервісів, що взаємодіють між собою через мережу.

Однією з найпоширеніших моделей побудови інформаційних систем є

клієнт-серверна архітектура. Вона передбачає взаємодію між клієнтськими застосунками та сервером, який обробляє запити й надає необхідні дані або послуги. Такий підхід дозволяє централізувати управління ресурсами, підвищити продуктивність системи та спростити її супровід. Подальшим розвитком клієнт-серверної моделі стала мікросервісна архітектура, у якій програмна система складається з окремих незалежних сервісів, кожен з яких виконує визначений набір функцій. Ефективне функціонування таких систем значною мірою залежить від використання сучасних засобів оркестрації та автоматизованого управління контейнерами.

Разом із перевагами використання клієнт-серверної та мікросервісної архітектур зростає складність управління їхніми компонентами. Велика кількість сервісів, контейнерів і мережевих взаємодій потребує застосування спеціалізованих механізмів координації та автоматизації процесів експлуатації.

Управління передбачає виконання комплексу заходів, спрямованих на підтримку стабільної роботи окремих компонентів системи. До таких завдань належать моніторинг стану сервісів, налаштування параметрів функціонування, забезпечення безпеки, резервне копіювання даних та контроль використання ресурсів.

Оркестрація, своєю чергою, забезпечує узгоджену взаємодію між усіма компонентами інформаційної системи. Вона охоплює автоматизацію процесів розгортання контейнерів, масштабування сервісів залежно від навантаження, балансування трафіку між вузлами, маршрутизацію запитів та підтримку безперервності роботи застосунків. Завдяки оркестрації значно підвищується ефективність адміністрування складних розподілених систем і забезпечується їхня надійна робота в умовах змінного навантаження (рис. 1.2). Автоматизація процесів розгортання та управління інфраструктурою є одним із ключових напрямів розвитку сучасних інформаційних технологій. Зі збільшенням масштабів програмних систем і хмарних сервісів традиційні методи ручного налаштування серверів та мережевих ресурсів стають менш ефективними, що зумовлює необхідність впровадження автоматизованих підходів до адміністрування інфраструктури.

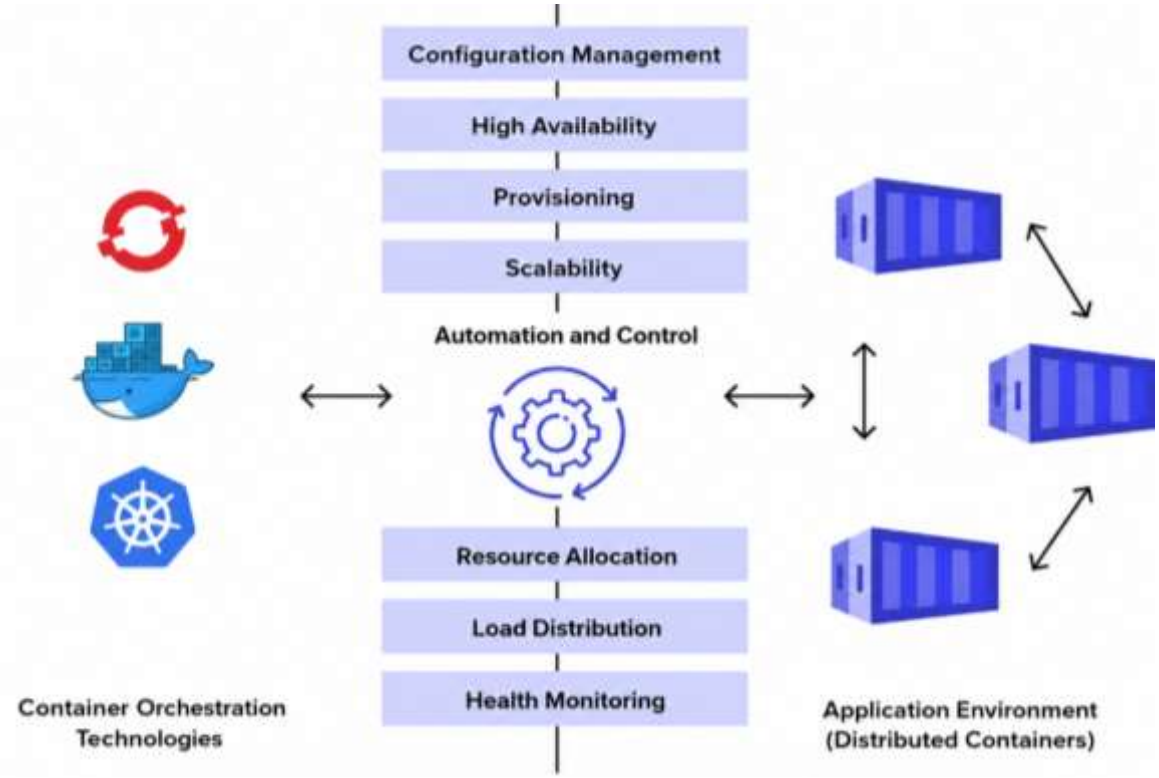


Рис. 1.2. Основні функції оркестрації контейнеризованих застосунків
Джерело: сформовано автором на основі [1-2; 7-8].

1.3. Принцип Infrastructure as Code (IaC) в управлінні хмарною інфраструктурою

Одним із таких підходів є Infrastructure as Code (IaC) – концепція, що передбачає опис та управління IT-інфраструктурою за допомогою програмного коду [11-12]. Відповідно до цього підходу конфігурації серверів, мереж, баз даних та інших ресурсів визначаються у спеціальних файлах, які можуть автоматично виконуватися відповідними програмними засобами. Це дозволяє відмовитися від ручного налаштування та забезпечити стандартизоване розгортання інфраструктури.

Використання IaC значно спрощує процес створення та підтримки інформаційних систем. Завдяки автоматизації адміністратори та розробники можуть швидко розгортати нові середовища, вносити зміни до конфігурацій і відтворювати інфраструктуру в разі потреби. Крім того, застосування програмного коду для управління ресурсами підвищує точність налаштувань, зменшує ризик людських помилок та забезпечує однаковість середовищ розробки, тестування і промислової експлуатації.

Таким чином, концепція Infrastructure as Code є важливою складовою сучасних DevOps-практик і хмарних технологій, оскільки забезпечує ефективне, масштабоване та надійне управління IT-інфраструктурою.

Після визначення бажаної конфігурації система IaC автоматично порівнює її з поточним станом інфраструктури та виявляє необхідні зміни. На основі результатів порівняння формуються операції зі створення, модифікації або видалення ресурсів. Далі відповідні зміни виконуються автоматично, що дозволяє швидко та безпомилково привести інфраструктуру до заданого стану (рис. 1.3).

Завдяки такому підходу забезпечується стандартизація процесів розгортання, скорочується час адміністрування систем, підвищується надійність конфігурацій та зменшується ймовірність виникнення помилок, пов’язаних із ручним налаштуванням інфраструктури.

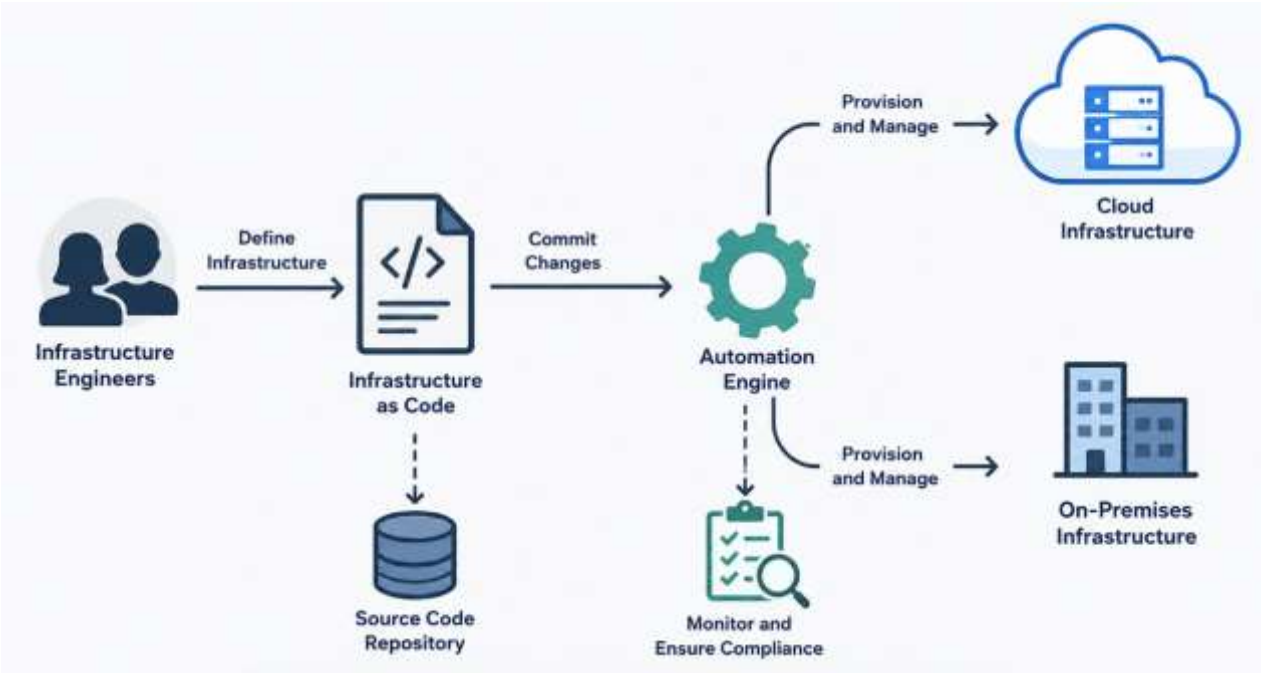


Рис. 1.3. Схема управління інфраструктурою на основі концепції Infrastructure as Code (IaC)

Джерело: сформовано автором на основі [6-7; 43].

Інфраструктура як код виступає важливим компонентом сучасних підходів до розробки та адміністрування IT-інфраструктури, забезпечуючи підвищення ефективності, надійності та рівня безпеки комп’ютерних систем.

1.4. Технології контейнеризації та їх значення для сучасних інформаційних систем

Моніторинг і телеметрія є ключовими складовими сучасних інформаційних систем, що забезпечують безперервний збір, обробку та аналіз даних для підтримки ефективного управління та прийняття обґрунтованих рішень. Завдяки цим технологіям можливе отримання в режимі реального часу актуальної інформації про стан системи, її продуктивність та основні параметри функціонування, зокрема навантаження, затримки, температуру, використання ресурсів тощо.

Моніторинг і телеметрія виступають важливими інструментами забезпечення стабільності та надійності програмних рішень, сприяючи їх ефективному супроводу та розвитку.

Моніторинг – це процес безперервного спостереження за станом програмних систем з метою виявлення відхилень, помилок або потенційних збоїв [5-7]. Він охоплює збір, обробку та аналіз технічних показників, таких як продуктивність, рівень завантаження ресурсів, час відгуку та інші критичні метрики. Завдяки моніторингу забезпечується своєчасне виявлення проблем і оперативне їх усунення, що позитивно впливає на стабільність і якість роботи систем.

Телеметрія передбачає автоматизований збір та передачу даних про функціонування системи, поведінку користувачів та використання функціональних можливостей програмного забезпечення.

Основні переваги впровадження моніторингу та телеметрії:

– Підвищення ефективності та продуктивності: постійний аналіз показників системи дозволяє виявляти вузькі місця та оптимізувати роботу додатків для досягнення максимальної продуктивності.

– Зниження витрат часу та ресурсів: раннє виявлення збоїв і відхилень дає змогу уникнути масштабних інцидентів і зменшити витрати на їх усунення.

– Покращення користувацького досвіду: своєчасне усунення помилок і вдосконалення функціоналу сприяє підвищенню задоволеності користувачів.

– Зменшення ризиків та підвищення надійності: систематичний контроль стану системи дозволяє запобігати критичним збоям і втраті даних.

Таким чином, моніторинг і телеметрія є невід'ємними елементами життєвого циклу сучасних програмних систем, що забезпечують їхню стабільність, безпеку та адаптивність до змінних умов експлуатації.

1.5. Моніторинг, телеметрія та використання хмарних сервісів для управління ресурсами

Хмарні сервіси – це інфраструктурні, платформні або програмні рішення, що надаються через мережу Інтернет і розміщуються на віддалених серверах, а не на локальних пристроях користувачів. Вони забезпечують доступ до обчислювальних ресурсів, програмного забезпечення та даних у межах хмарного середовища, що суттєво підвищує гнучкість і ефективність використання IT-інфраструктури.

Значення використання хмарних сервісів:

– Масштабованість: хмарні технології дозволяють оперативно збільшувати або зменшувати обсяг ресурсів відповідно до потреб користувача чи організації.

– Оптимізація витрат: застосування хмарних рішень знижує витрати на придбання, встановлення та обслуговування фізичного обладнання.

– Доступність: забезпечується можливість доступу до даних і застосунків з будь-якого місця та пристрою за наявності підключення до Інтернету.

– Безпека: провайдери хмарних послуг впроваджують сучасні механізми захисту даних, включаючи шифрування та відповідність міжнародним стандартам безпеки.

– Швидкість розгортання: хмарні платформи дають змогу оперативно запускати нові сервіси та оновлювати існуючі рішення без значних затримок.

Основні переваги хмарних сервісів:

– Економічна ефективність: модель оплати «за фактичне використання» дозволяє раціонально використовувати фінансові ресурси.

– Мобільність і доступність: користувачі можуть працювати з даними та застосунками незалежно від місця перебування.

– Гнучкість і масштабованість: інфраструктура легко адаптується до змін навантаження та потреб бізнесу.

– Підвищений рівень безпеки: сучасні хмарні провайдери забезпечують багаторівневий захист інформації та резервне копіювання даних.

– Висока продуктивність: швидке розгортання ресурсів сприяє підвищенню ефективності бізнес-процесів і цифрових сервісів [44-45].

Таким чином, хмарні сервіси є важливою складовою цифрової трансформації, що забезпечує організаціям можливість підвищувати ефективність, адаптивність та інноваційність у сучасному інформаційному середовищі (рис. 1.4).

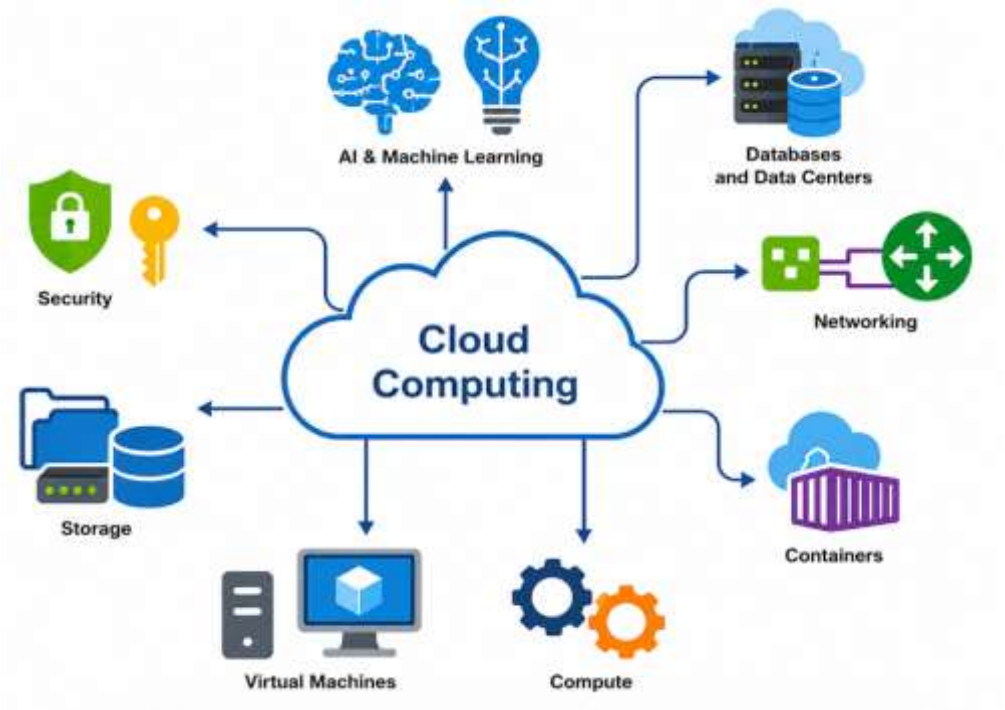


Рис. 1.4. Загальна модель хмарних обчислень

Джерело: сформовано автором на основі [12-13; 15].

РОЗДІЛ 2. ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ УПРАВЛІННЯ ХМАРНИМИ РЕСУРСАМИ

2.1. Аналіз вимог до програмного додатку

Для контейнеризації додатків існує значна кількість інструментів, що відрізняються функціональністю та підходами до реалізації. Найпоширенішими серед них є Docker та Podman.

Docker – це програмна платформа для створення, запуску та керування контейнерами. Контейнер Docker є легким виконуваним пакетом, що містить усі необхідні компоненти для запуску застосунку, зокрема код, бібліотеки та змінні середовища. Контейнери ізольовані від основної операційної системи та один від одного, що забезпечує стабільну роботу в різних середовищах. Крім того, Docker підтримує розповсюдження контейнерних образів через репозиторії, такі як Docker Hub [22].

Основні особливості Docker:

- ✓ Портативність: контейнери можуть бути легко перенесені між різними операційними системами (Linux, Windows, macOS), що спрощує процеси розробки, тестування та розгортання.
- ✓ Висока продуктивність: контейнери запускаються безпосередньо в середовищі ядра операційної системи, що забезпечує менше споживання ресурсів порівняно з віртуальними машинами. Також вони можуть працювати паралельно без конфліктів.
- ✓ Ізоляція та безпека: використання механізмів простору імен (namespaces) та контрольних груп (cgroups) забезпечує ізоляцію процесів і запобігає несанкціонованому доступу між контейнерами.

Переваги Docker:

- ➢ Спрощення розробки: забезпечує стандартизоване середовище виконання, яке легко відтворюється на різних пристроях, зменшуючи проблеми сумісності та залежностей.
- ➢ Гнучкість розгортання: дозволяє запускати застосунки на будь-якому сервері з підтримкою Docker Engine, що підвищує масштабованість і доступність систем.

- Інтеграція з іншими технологіями: сумісний із такими платформами, як Kubernetes, OpenShift, Rancher та іншими інструментами оркестрації контейнерів, що розширює можливості керування інфраструктурою.

Робочий процес створення контейнерів та образів за допомогою Docker подано на рис. 2.1.

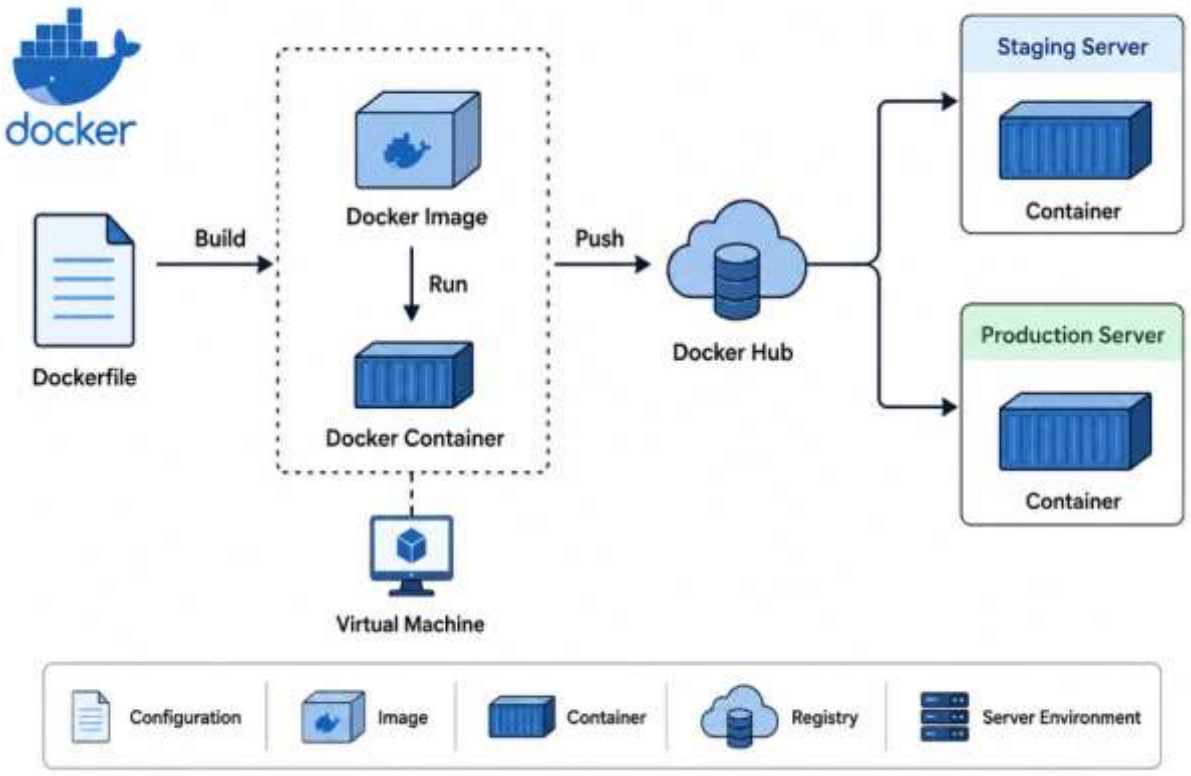


Рис. 2.1. Схема процесу створення та розгортання контейнеризованих застосунків із використанням Docker

Джерело: сформовано автором на основі [18-20].

Podman – це інструмент контейнеризації, який є альтернативою Docker і дозволяє створювати та керувати контейнерами без використання центрального демона [25]. Крім того, Podman підтримує використання Kubernetes YAML-манифестів для розгортання контейнеризованих застосунків.

Основні особливості Podman:

- Бездемонна архітектура: Podman не потребує постійно працюючого демона для керування контейнерами. Замість цього використовується бібліотека libpod, яка забезпечує API для роботи з контейнерами. Це знижує навантаження на систему та підвищує рівень безпеки.

- Сумісність із Docker: Podman підтримує ті самі образи, команди та репозиторії, що й Docker, що спрощує міграцію між цими інструментами.
- Інтеграція з Kubernetes: Podman може генерувати та застосовувати Kubernetes YAML-конфігурації, що полегшує розгортання контейнерних застосунків у кластерних середовищах.

Переваги Podman:

- Підвищена безпека: контейнери запускаються як звичайні процеси користувача без потреби у привілейованому демонові, що зменшує ризики ескалації привілеїв.
- Гнучкість використання: Podman забезпечує можливість запуску контейнерів у різних операційних системах і середовищах, що підвищує портативність рішень.
- Зручне управління: інструмент підтримує роботу з подами (pods), образами (images), мережами (network) та іншими компонентами контейнерної інфраструктури.

Порівняння Docker і Podman:

Docker є більш поширеним і функціонально розвиненим інструментом контейнеризації, який фактично став галузевим стандартом. Він має широку екосистему, велику кількість інтеграцій і активно використовується у промислових середовищах.

Podman, у свою чергу, має переваги в архітектурі без демона та підвищеній безпеці. Він забезпечує сумісність із Docker-контейнерами та пропонує спрощене управління контейнерною інфраструктурою.

Таким чином, вибір між Docker і Podman залежить від вимог до системи, рівня безпеки, а також досвіду команди розробників та адміністраторів.

Обґрунтування вибору Docker:

Docker обрано як основний інструмент контейнеризації з огляду на такі переваги:

- Зрілість і стабільність: Docker існує з 2013 року та є перевіреним часом рішенням, яке широко використовується у промисловості.

- Висока популярність та підтримка: інструмент має велику спільноту, розвинену документацію та значну кількість навчальних матеріалів.
- Широка інтеграція: Docker підтримує взаємодію з оркестраторами, такими як Kubernetes, OpenShift та іншими хмарними платформами, включно з AWS, Microsoft Azure та Google Cloud Platform.

2.2. Обґрунтування вибору технологій та програмних засобів

Оркестрація контейнеризованих застосунків являє собою комплекс механізмів автоматизованого управління процесами розгортання, налаштування, масштабування та функціонування контейнерів у межах кластерної інфраструктури. Використання оркестрації дає змогу забезпечити безперервність роботи сервісів, підвищити рівень відмовостійкості та ефективно розподіляти обчислювальні ресурси між окремими компонентами системи.

Сьогодні для реалізації оркестрації контейнеризованих додатків застосовується широкий спектр програмних платформ, які відрізняються функціональними можливостями та особливостями використання. Найбільш поширеними рішеннями у цій сфері є Kubernetes та OpenShift, що забезпечують ефективне управління контейнерними середовищами різного масштабу.

Основні особливості Kubernetes [12; 27; 29]:

– Кластерна архітектура. Kubernetes забезпечує об’єднання фізичних або віртуальних серверів у кластер, що складається з окремих вузлів (Node). На цих вузлах розміщуються контейнери, які об’єднуються в логічні групи – поди (Pod). Под є базовою одиницею розгортання в Kubernetes і може містити один або декілька взаємопов’язаних контейнерів. Розподіл подів між вузлами здійснюється планувальником (Scheduler), який враховує доступність ресурсів, задані політики та інші параметри конфігурації.

– Механізми самовідновлення. Однією з ключових переваг Kubernetes є підтримка автоматичного відновлення працездатності системи. Платформа постійно контролює стан вузлів і подів та у разі виникнення збоїв автоматично виконує перезапуск або перенесення контейнерів на інші

доступні вузли. Крім того, Kubernetes підтримує автоматичне масштабування застосунків відповідно до поточного навантаження, що сприяє ефективному використанню обчислювальних ресурсів.

– Мережеві сервіси та балансування навантаження. Kubernetes надає механізми організації взаємодії між контейнеризованими компонентами та зовнішніми користувачами за допомогою сервісів (Service). Залежно від призначення можуть використовуватися внутрішні сервіси (ClusterIP), сервіси з доступом через вузол (NodePort), сервіси з балансуванням навантаження (LoadBalancer) або безпосередній доступ до подів через Headless Service. Для виявлення та маршрутизації трафіку застосовуються мітки (Labels) і селектори (Selectors), що забезпечує гнучке управління мережевими з’єднаннями.

– Управління конфігурацією та безпекою. Kubernetes надає інструменти для централізованого керування параметрами контейнеризованих застосунків. Для зберігання конфігураційних даних використовуються ресурси ConfigMap, які дозволяють відокремити налаштування від програмного коду. Конфіденційна інформація, зокрема паролі, токени доступу та криптографічні ключі, зберігається за допомогою ресурсу Secret, що підвищує рівень безпеки та спрощує адміністрування програмних систем.

Переваги використання Kubernetes:

– Висока продуктивність. Kubernetes забезпечує раціональне використання обчислювальних ресурсів завдяки автоматизованому розподілу навантаження, масштабуванню та оптимальному розміщенню контейнерів у кластері. Платформа підтримує механізми сервісного виявлення, маршрутизації мережевого трафіку та балансування навантаження, що сприяє підвищенню швидкодії контейнеризованих застосунків і стабільності їх роботи.

– Висока надійність та відмовостійкість. Kubernetes забезпечує безперервне функціонування програмних сервісів завдяки механізмам автоматичного моніторингу та відновлення компонентів системи. У разі виникнення помилок або відмов контейнери можуть бути автоматично перезапущені чи перенесені на інші вузли кластера. Крім того, платформа

підтримує сучасні підходи до розгортання та оновлення застосунків, зокрема поступове оновлення (Rolling Update), а також стратегії Blue-Green і Canary Deployment, що дозволяє мінімізувати ризики під час впровадження нових версій програмного забезпечення.

– Гнучкість і масштабованість. Kubernetes може використовуватися в різних середовищах виконання, включаючи локальні сервери, віртуалізовану інфраструктуру та хмарні платформи. Це дозволяє адаптувати платформу до специфічних вимог організації та забезпечувати ефективне управління контейнерними застосунками незалежно від масштабу інфраструктури.

Таким чином, Kubernetes є потужною платформою оркестрації контейнерів, яка поєднує високу продуктивність, надійність і гнучкість, що робить її одним із найбільш поширених рішень для розгортання сучасних контейнеризованих систем.

На рис. 2.2 наведено архітектуру Kubernetes, яка демонструє складові елементи кластера та особливості їхньої взаємодії в процесі управління контейнеризованими застосунками.

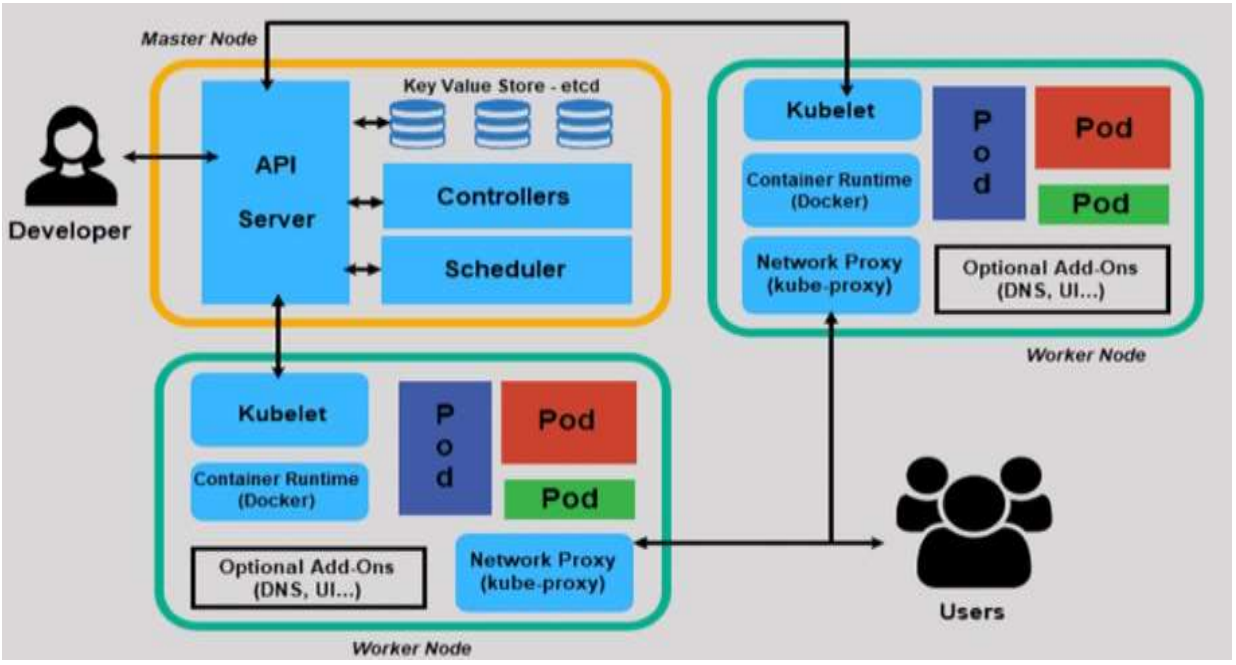


Рис. 2.2. Схема архітектури Kubernetes із відображенням основних компонентів

Джерело: сформовано автором на основі [11; 17-18].

OpenShift являє собою програмну платформу для управління контейнеризованими застосунками, розроблену на базі технології Kubernetes.

Її використання забезпечує не лише оркестрацію контейнерів, а й розширені можливості адміністрування, контролю безпеки, моніторингу та автоматизації процесів розроблення і впровадження програмного забезпечення. Завдяки комплексному підходу до керування інфраструктурою OpenShift широко застосовується для побудови масштабованих корпоративних інформаційних систем [29; 42].

До ключових характеристик платформи належать:

– Забезпечення безпеки середовища виконання. OpenShift реалізує комплекс засобів захисту, спрямованих на контроль доступу до ресурсів кластера та збереження цілісності даних. Для цього використовуються механізми керування обліковими записами, політики доступу, рольове розмежування повноважень, а також інструменти аудиту та шифрування інформації. Додатковий рівень захисту досягається завдяки використанню технологій ізоляції процесів і контейнерів.

– Контроль та аналіз роботи системи. Платформа надає інструменти для збору, обробки та візуалізації інформації про стан кластерної інфраструктури. Це дозволяє здійснювати моніторинг використання ресурсів, оцінювати продуктивність застосунків, аналізувати журнали подій та оперативно реагувати на відхилення в роботі сервісів.

– Автоматизація життєвого циклу застосунків. OpenShift підтримує автоматизоване виконання основних операцій, пов’язаних із розгортанням, оновленням і масштабуванням програмних компонентів. Інтеграція із сучасними засобами безперервної інтеграції та доставки сприяє прискоренню процесів розроблення програмного забезпечення та зменшенню кількості помилок під час його впровадження.

– Підтримка різних інфраструктурних середовищ. Платформа може функціонувати в локальних центрах обробки даних, приватних, публічних і гібридних хмарах. Така універсальність забезпечує можливість вибору оптимальної моделі розміщення сервісів залежно від вимог організації. Крім того, OpenShift підтримує інтеграцію з численними зовнішніми сервісами та програмними рішеннями, що розширює сферу її практичного застосування.

Отже, OpenShift є сучасною платформою для управління контейнеризованими середовищами, яка поєднує функціональні можливості Kubernetes із додатковими інструментами адміністрування, безпеки та автоматизації. Це робить її ефективним засобом для створення, розгортання та підтримки масштабованих програмних систем у хмарних і локальних інфраструктурах.

Переваги використання OpenShift

– Оптимізація процесу розроблення програмного забезпечення. OpenShift надає широкий набір інструментів, які спрощують створення та тестування контейнеризованих застосунків. Платформа містить готові шаблони розгортання, каталоги сервісів, графічний інтерфейс керування та засоби командного рядка, що прискорюють виконання типових операцій. Крім того, розробники можуть використовувати локальне середовище для моделювання роботи кластера, що дозволяє проводити тестування та налагодження застосунків до їх впровадження у виробниче середовище.

– Універсальність розгортання. Однією з важливих переваг OpenShift є можливість її використання в різних інфраструктурних середовищах. Платформа підтримує розгортання застосунків як на локальних серверах і віртуалізованих системах, так і в хмарних середовищах. Також OpenShift забезпечує підтримку гібридних і мультихмарних моделей, що дозволяє організаціям поєднувати ресурси кількох платформ та ефективно розподіляти навантаження між ними.

– Розширені функціональні можливості. На відміну від базової реалізації Kubernetes, OpenShift пропонує додаткові інструменти для автоматизації та керування контейнеризованими середовищами. Зокрема, використання операторів (Operators) спрощує адміністрування складних сервісів, технологія Service Mesh забезпечує контроль взаємодії між мікросервісами, а Serverless-підхід дозволяє виконувати окремі функції без необхідності постійного керування серверною інфраструктурою. Завдяки цьому підвищується ефективність використання ресурсів і розширюються можливості створення сучасних хмарних застосунків.

Таким чином, OpenShift поєднує інструменти розробки, розгортання та

управління контейнеризованими застосунками, забезпечуючи високу гнучкість, масштабованість і функціональність програмних рішень.

Порівняльна характеристика Kubernetes та OpenShift

Kubernetes і OpenShift належать до найбільш поширених платформ для управління контейнеризованими застосунками та побудови хмарної інфраструктури. Обидва рішення використовуються для автоматизації процесів розгортання, масштабування та супроводу програмних систем, проте відрізняються рівнем функціональності, підходами до адміністрування та особливостями використання.

Kubernetes є відкритою платформою оркестрації контейнерів, яка забезпечує базовий набір інструментів для керування кластерною інфраструктурою. Система дозволяє автоматизувати процеси розгортання застосунків, балансування навантаження, моніторингу стану контейнерів та масштабування ресурсів відповідно до поточних потреб. Важливою перевагою Kubernetes є його широка підтримка спільнотою розробників, що сприяє постійному вдосконаленню платформи та розширенню її функціональних можливостей.

На відміну від Kubernetes, OpenShift являє собою комплексну платформу корпоративного рівня, побудовану на його основі. Крім стандартних механізмів оркестрації контейнерів, OpenShift містить додаткові інструменти для забезпечення інформаційної безпеки, моніторингу інфраструктури, автоматизації процесів розроблення та інтеграції із зовнішніми сервісами. Наявність готових механізмів адміністрування значно спрощує експлуатацію контейнеризованих середовищ, особливо в організаціях із великими обсягами обчислювальних ресурсів.

Під час вибору між Kubernetes та OpenShift необхідно враховувати особливості проєкту, вимоги до інфраструктури та рівень складності адміністрування. Kubernetes забезпечує більшу свободу налаштування та інтеграції з різноманітними програмними компонентами, тоді як OpenShift орієнтований на надання готового середовища з розширеним набором сервісів.

Для реалізації даного проєкту обрано платформу Kubernetes. Таке

рішення обумовлене низкою переваг.

– Відкритість та широке поширення. Kubernetes є відкритою платформою з великим професійним співтовариством користувачів і розробників. Завдяки цьому доступна значна кількість документації, навчальних матеріалів та програмних рішень, що спрощує впровадження й підтримку системи.

– Гнучкість конфігурації. Архітектура Kubernetes передбачає можливість адаптації платформи до різних сценаріїв використання шляхом підключення додаткових модулів, операторів та інших компонентів. Це дозволяє створювати інфраструктуру відповідно до конкретних вимог проєкту без обмежень, характерних для готових комерційних рішень.

– Незалежність від постачальника. Kubernetes підтримується широким колом компаній та може функціонувати на різних типах інфраструктури, включаючи локальні сервери, приватні та публічні хмарні середовища. Такий підхід мінімізує залежність від окремого виробника програмного забезпечення та забезпечує високу сумісність із сучасними технологічними платформами.

Отже, хоча OpenShift пропонує додаткові можливості для корпоративного використання, Kubernetes є більш універсальним, відкритим і гнучким рішенням, що зумовило його вибір як основної платформи оркестрації контейнерів у межах даного проєкту.

2.3. Проєктування архітектури системи

Інфраструктура як код (Infrastructure as Code, IaC) є сучасним підходом до управління обчислювальною інфраструктурою, який передбачає опис її компонентів у вигляді програмного коду. Використання концепції IaC дозволяє автоматизувати процеси створення, конфігурування та супроводу інфраструктурних ресурсів, забезпечуючи відтворюваність середовищ, зменшення кількості помилок і підвищення ефективності адміністрування [32-33]. Завдяки цьому інфраструктура розглядається як програмний артефакт, до якого можуть застосовуватися засоби контролю версій, тестування та автоматизованого розгортання.

Серед великої кількості інструментів, призначених для реалізації підходу Infrastructure as Code, найбільшого поширення набули Terraform і Pulumi [40-41]. Дані платформи забезпечують автоматизоване управління інфраструктурними ресурсами в локальних, хмарних і гібридних середовищах.

Terraform є одним із найпоширеніших інструментів IaC, розробленим компанією HashiCorp. Його основне призначення полягає у створенні, зміні та керуванні інфраструктурою за допомогою декларативного опису ресурсів. Користувач визначає бажаний стан інфраструктури у конфігураційних файлах, після чого Terraform автоматично виконує необхідні операції для приведення системи до зазначеного стану.

Для опису конфігурації використовується спеціалізована мова HCL (HashiCorp Configuration Language), яка характеризується зрозумілим синтаксисом і дозволяє задавати параметри ресурсів, взаємозв'язки між ними та правила їх розгортання. Завдяки декларативному підходу адміністратор описує кінцевий результат, тоді як процес реалізації необхідних змін виконується автоматично.

Важливою особливістю Terraform є підтримка великої кількості провайдерів, що забезпечує взаємодію з різноманітними платформами та сервісами. Інструмент може використовуватися для управління ресурсами хмарних середовищ, контейнерних платформ, систем віртуалізації та мережевої інфраструктури. Це робить його універсальним рішенням для автоматизації процесів розгортання сучасних інформаційних систем.

Додатковою перевагою Terraform є підтримка модульної архітектури. Використання модулів дозволяє структурувати конфігураційний код, повторно використовувати типові рішення та спрощувати управління великими інфраструктурними проєктами. Завдяки цьому підвищується масштабованість, зручність супроводу та ефективність розробки інфраструктурних рішень.

На рис. 2.3 наведено послідовність етапів створення та управління інфраструктурою за допомогою Terraform. Робочий процес включає підготовку конфігураційних файлів, ініціалізацію проєкту, перевірку та

планування змін, автоматизоване розгортання ресурсів і подальше управління станом інфраструктури. Такий підхід забезпечує відтворюваність, автоматизацію та контроль усіх операцій із інфраструктурними компонентами.

Незважаючи на значну популярність та широкі можливості автоматизації інфраструктурних процесів, Terraform має низку особливостей, які можуть ускладнювати його використання в окремих сценаріях.

Одним із обмежень інструменту є відсутність повноцінних механізмів обробки винятків, характерних для традиційних мов програмування. Унаслідок цього помилки в конфігураційних файлах або некоректно визначені параметри ресурсів можуть призводити до виникнення неузгодженостей між фактичним і бажаним станом інфраструктури, що потребує додаткового аналізу та ручного втручання адміністратора.

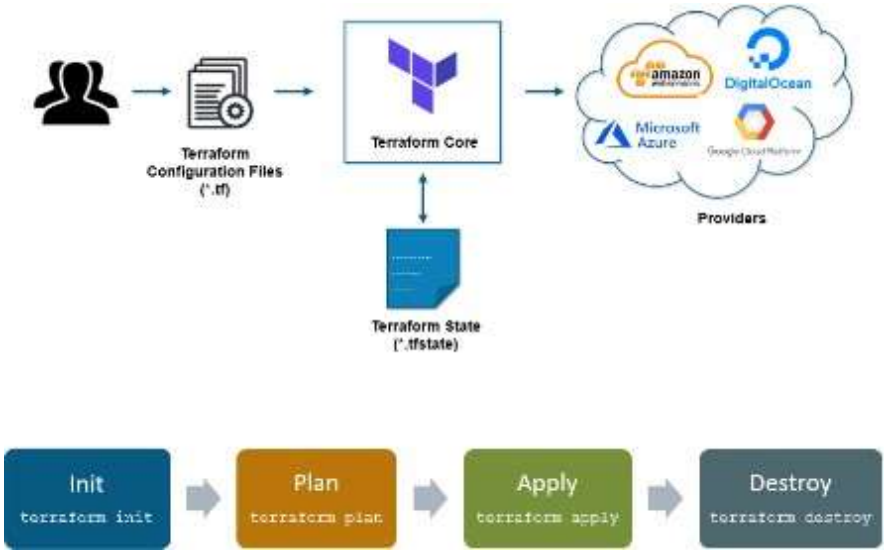


Рис. 2.3. Процес автоматизованого управління інфраструктурою за допомогою Terraform

Джерело: сформовано автором на основі [37-39].

Крім того, Terraform орієнтований переважно на створення та керування інфраструктурними ресурсами, тому його можливості щодо виконання складних післярозгортальних операцій є обмеженими. Для реалізації додаткового налаштування програмного забезпечення, встановлення пакетів або зміни конфігурацій після створення ресурсів часто виникає потреба у використанні спеціалізованих засобів автоматизації.

Певні труднощі можуть виникати і під час реалізації складних алгоритмів керування інфраструктурою. Оскільки Terraform використовує декларативний підхід та мову HCL, можливості опису складної бізнес-логіки є менш гнучкими порівняно з рішеннями, що базуються на повноцінних мовах програмування.

Додатковим обмеженням є відсутність вбудованих засобів модульного тестування конфігурацій. Це може ускладнювати перевірку правильності роботи інфраструктурного коду та потребувати застосування сторонніх інструментів для тестування й валідації створених конфігурацій.

Таким чином, Terraform є ефективним інструментом реалізації концепції Infrastructure as Code, проте під час його використання необхідно враховувати особливості, пов’язані з обробкою помилок, тестуванням конфігурацій та реалізацією складних сценаріїв керування інфраструктурою.

Додатковим фактором, який вплинув на розвиток ринку інструментів Infrastructure as Code, стала зміна ліцензійної політики компанії HashiCorp. У серпні 2023 року було оголошено про перехід низки продуктів компанії, зокрема Terraform, від відкритої ліцензії Mozilla Public License 2.0 (MPL 2.0) до ліцензії Business Source License (BSL 1.1). Таке рішення обмежило можливості використання Terraform у певних комерційних сценаріях, зокрема у випадках створення продуктів або сервісів, що можуть конкурувати з рішеннями HashiCorp. Зміна ліцензійної моделі викликала активне обговорення в професійній спільноті та сприяла зростанню інтересу до альтернативних платформ автоматизації інфраструктури.

Одним із таких рішень є Pulumi – сучасний інструмент реалізації концепції Infrastructure as Code, який надає можливість описувати інфраструктурні ресурси за допомогою повноцінних мов програмування. На відміну від багатьох інших платформ IaC, Pulumi не потребує використання спеціалізованих мов конфігурації, що дозволяє розробникам застосовувати вже наявні знання та навички програмування.

Платформа підтримує широкий спектр мов програмування, серед яких Python, JavaScript, TypeScript, Go, C#, Java та YAML. Такий підхід забезпечує гнучкість розробки та дозволяє обирати інструментарій відповідно до вимог

проєкту та досвіду команди розробників.

Важливою перевагою Pulumi є підтримка великої кількості хмарних платформ і сервісів. Інструмент забезпечує інтеграцію з провідними постачальниками хмарних послуг, контейнерними платформами та іншими інфраструктурними рішеннями, що робить його універсальним засобом автоматизації розгортання ресурсів у різних середовищах.

Завдяки використанню сучасних мов програмування, широким можливостям інтеграції та активному розвитку спільноти користувачів Pulumi розглядається як одна з перспективних альтернатив традиційним інструментам Infrastructure as Code.

Робочий процес створення та управління інфраструктурою за допомогою Pulumi наведено на рис. 2.4.

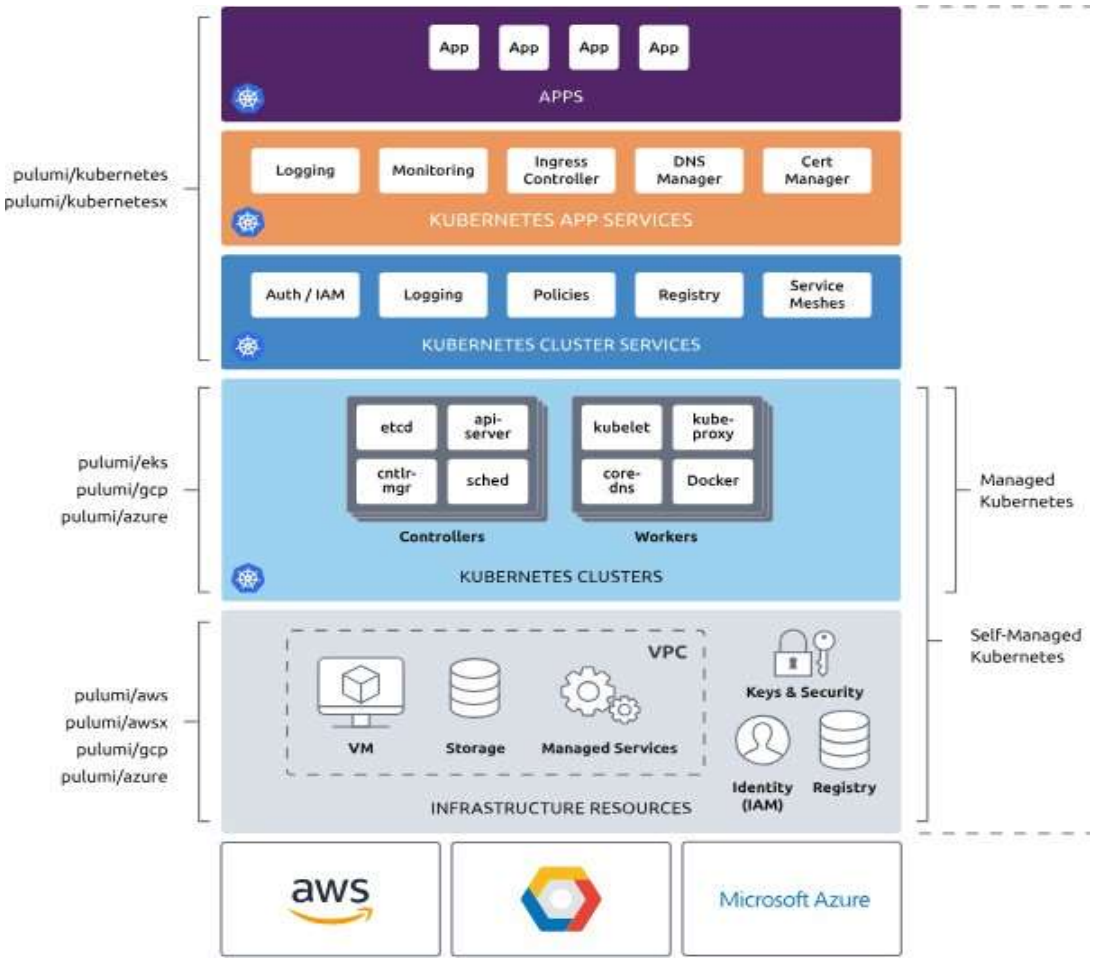


Рис. 2.4. Процес автоматизованого управління інфраструктурою за допомогою Pulumi

Джерело: сформовано автором на основі [18-20].

Terraform і Pulumi належать до найбільш поширених інструментів реалізації концепції Infrastructure as Code та забезпечують автоматизацію

процесів створення, налаштування й управління інфраструктурними ресурсами. Обидві платформи підтримують роботу з хмарними сервісами, контейнерними середовищами та іншими елементами сучасної IT-інфраструктури. Водночас вони відрізняються підходами до опису конфігурацій, рівнем гнучкості та можливостями інтеграції з процесами розроблення програмного забезпечення.

У межах даного проєкту для автоматизації управління інфраструктурою обрано платформу Pulumi. Таке рішення обумовлене низкою переваг, які роблять її більш придатною для реалізації поставлених завдань.

– Використання повноцінних мов програмування. На відміну від Terraform, який базується на декларативній мові HCL, Pulumi дозволяє описувати інфраструктуру за допомогою сучасних мов програмування, зокрема Python, TypeScript, JavaScript, Go та C#. Це забезпечує більшу гнучкість під час розроблення, спрощує підтримку коду та дозволяє використовувати знайомі інструменти й бібліотеки.

– Розширені можливості програмування інфраструктури. Завдяки підтримці повноцінних мов програмування Pulumi надає можливість реалізовувати складні алгоритми, умовні конструкції, цикли та інші механізми, які можуть бути необхідними під час автоматизації інфраструктурних процесів. Це особливо важливо для проєктів із високими вимогами до адаптивності та масштабованості.

– Гнучкість інтеграції з різними платформами. Pulumi підтримує широкий спектр хмарних провайдерів і технологічних платформ, включаючи сервіси публічних хмар, контейнерні оркестратори та локальні інфраструктурні рішення. Такий підхід забезпечує незалежність від конкретного постачальника послуг та спрощує перенесення інфраструктури між різними середовищами.

– Повторне використання коду та компонентів. Платформа дозволяє створювати власні модулі, бібліотеки та компоненти, які можуть використовуватися в різних проєктах. Це сприяє стандартизації процесів розроблення та підвищує ефективність управління інфраструктурою.

Отже, вибір Pulumi зумовлений можливістю використання сучасних мов

програмування, високою гнучкістю конфігурації та широкими можливостями інтеграції з різними інфраструктурними платформами. Такі характеристики роблять даний інструмент ефективним засобом реалізації підходу Infrastructure as Code в межах розроблюваної системи.

2.4. Інструменти для моніторингу та телеметрії

Моніторинг і телеметрія є важливими складовими сучасних інформаційних систем, оскільки забезпечують постійний контроль за станом інфраструктури, продуктивністю сервісів та використанням обчислювальних ресурсів. Застосування спеціалізованих інструментів дозволяє своєчасно виявляти несправності, аналізувати причини виникнення збоїв і приймати обґрунтовані рішення щодо оптимізації роботи системи. Особливої актуальності такі засоби набувають у середовищах хмарних обчислень і контейнерної оркестрації, де кількість компонентів та взаємозв’язків між ними постійно зростає.

Серед найбільш поширених рішень у сфері моніторингу та телеметрії варто виділити зв’язку Grafana і Prometheus, а також платформу DataDog. Дані інструменти забезпечують збір, обробку, аналіз і візуалізацію інформації про функціонування програмних систем та інфраструктурних компонентів.

Prometheus являє собою спеціалізовану платформу моніторингу, призначену для збору та зберігання часових рядів даних. Основною функцією системи є автоматизований збір метрик із серверів, контейнерів, мережевих пристроїв і програмних застосунків. Архітектура Prometheus забезпечує високу масштабованість та ефективність обробки великих обсягів телеметричної інформації. Крім того, система містить вбудовані механізми формування правил оповіщення, що дозволяє оперативно реагувати на критичні зміни показників функціонування інфраструктури [29].

Спільне використання Grafana та Prometheus формує комплексне рішення для моніторингу інформаційних систем. У такій архітектурі Prometheus виконує функції збору, обробки та зберігання метрик, тоді як Grafana забезпечує їх візуальне представлення та аналітичну обробку.

Поєднання цих інструментів дозволяє отримувати актуальну інформацію про стан інфраструктури, виявляти аномалії в роботі сервісів та здійснювати аналіз тенденцій зміни ключових показників.

Архітектуру взаємодії Grafana та Prometheus у процесі моніторингу інфраструктури наведено на рис. 2.5.

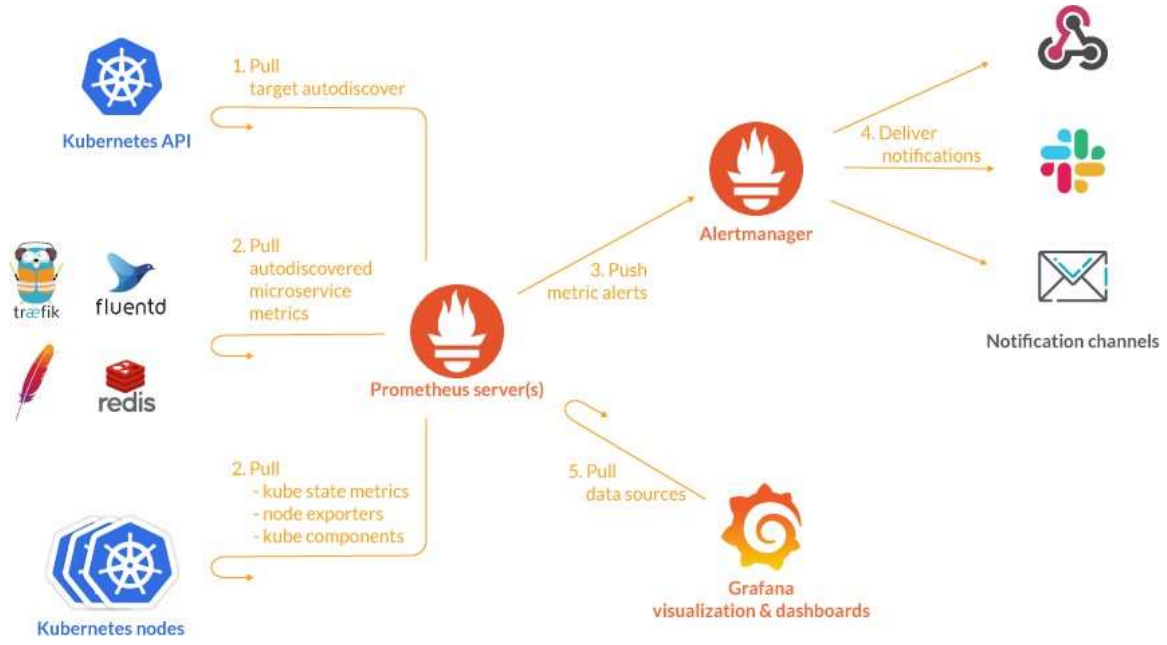


Рис. 2.5. Процес збору, обробки та візуалізації метрик за допомогою Grafana та Prometheus

Джерело: сформовано автором на основі [26-27; 30].

DataDog є комплексною хмарною платформою для моніторингу, аналізу та управління продуктивністю інформаційних систем. Основне призначення платформи полягає у забезпеченні безперервного контролю за станом інфраструктури, програмних застосунків і мережевих сервісів з метою підтримання їхньої стабільної та ефективної роботи.

Платформа забезпечує централізований збір телеметричних даних із різноманітних джерел, зокрема фізичних і віртуальних серверів, контейнерних середовищ, хмарних сервісів та програмних застосунків. Отримана інформація обробляється та відображається за допомогою інтерактивних інформаційних панелей, що дозволяє оперативно оцінювати стан інфраструктури та аналізувати ключові показники її функціонування.

Порівнюючи рішення Grafana/Prometheus та DataDog, слід зазначити, що обидва підходи забезпечують широкий спектр можливостей для моніторингу

інфраструктури та програмних сервісів. Водночас для середовищ, побудованих на основі Kubernetes, більш доцільним є використання зв'язки Grafana та Prometheus [22; 44].

Вибір Grafana/Prometheus обумовлений такими перевагами:

– Тісна інтеграція з Kubernetes. Prometheus розроблявся з урахуванням особливостей контейнеризованих середовищ і підтримує автоматичне виявлення вузлів, сервісів та контейнерів у кластері. Це забезпечує оперативний збір метрик без необхідності складного налаштування додаткових компонентів.

– Масштабованість і адаптивність. Відкритий вихідний код Prometheus дозволяє гнучко налаштовувати систему відповідно до вимог конкретного проєкту. Інструмент ефективно працює як у невеликих тестових середовищах, так і в масштабних виробничих кластерах.

– Комплексний моніторинг компонентів Kubernetes. Платформа забезпечує контроль стану основних елементів кластера, включаючи вузли, поди, сервіси, контролери та API-сервер, що дозволяє отримати повне уявлення про роботу інфраструктури.

– Розвинені засоби візуалізації. Grafana надає широкі можливості для побудови інформаційних панелей, графіків і аналітичних звітів. Інтеграція з Prometheus забезпечує наочне представлення метрик та спрощує аналіз продуктивності системи.

Отже, з урахуванням особливостей контейнерної оркестрації та вимог до моніторингу Kubernetes-кластерів, поєднання Grafana та Prometheus є найбільш доцільним рішенням для реалізації системи моніторингу в межах даного проєкту. Зазначені інструменти забезпечують високу масштабованість, гнучкість налаштування та ефективний контроль за станом інфраструктури й програмних сервісів.

2.5. Огляд хмарних провайдерів

Хмарні обчислення є одним із ключових напрямів розвитку сучасних інформаційних технологій, забезпечуючи доступ до обчислювальних

ресурсів, сховищ даних та програмних сервісів через мережу Інтернет. Надання таких ресурсів здійснюється спеціалізованими компаніями – хмарними провайдерами, які забезпечують користувачам можливість використовувати обчислювальну інфраструктуру без необхідності придбання та обслуговування власного апаратного забезпечення.

Використання хмарних платформ дозволяє організаціям зменшити витрати на створення та підтримку IT-інфраструктури, підвищити гнучкість використання ресурсів, а також забезпечити масштабування сервісів відповідно до поточних потреб. Завдяки цьому хмарні технології набули широкого поширення як серед великих підприємств, так і серед невеликих організацій та індивідуальних розробників.

На світовому ринку хмарних обчислень провідні позиції займають три основні постачальники послуг: Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Дані платформи пропонують широкий спектр сервісів для обробки та зберігання даних, розгортання програмних застосунків, управління контейнерними середовищами, реалізації технологій штучного інтелекту та забезпечення інформаційної безпеки.

Кожен із зазначених провайдерів має власні особливості, набір інструментів та підходи до організації хмарної інфраструктури, однак усі вони забезпечують високу доступність сервісів, масштабованість ресурсів та підтримку сучасних технологій розроблення програмного забезпечення.

На рис. 2.6 наведено розподіл часток світового ринку між провідними постачальниками хмарних послуг станом на початок 2023 року, що дозволяє оцінити рівень їхньої присутності та конкурентні позиції в галузі хмарних обчислень.

Amazon Web Services (AWS) є одним із провідних світових постачальників хмарних обчислювальних послуг. Компанія Amazon представила платформу AWS у 2006 році, і відтоді вона посіла одне з ключових місць у розвитку індустрії хмарних технологій [14; 28].

Платформа AWS охоплює широкий спектр сервісів, зокрема обчислювальні ресурси (Amazon EC2), об’єктне сховище даних (Amazon S3), системи управління базами даних (Amazon RDS), інструменти для розробки

програмного забезпечення, а також сервіси у сфері штучного інтелекту й машинного навчання (Amazon SageMaker) та інші.

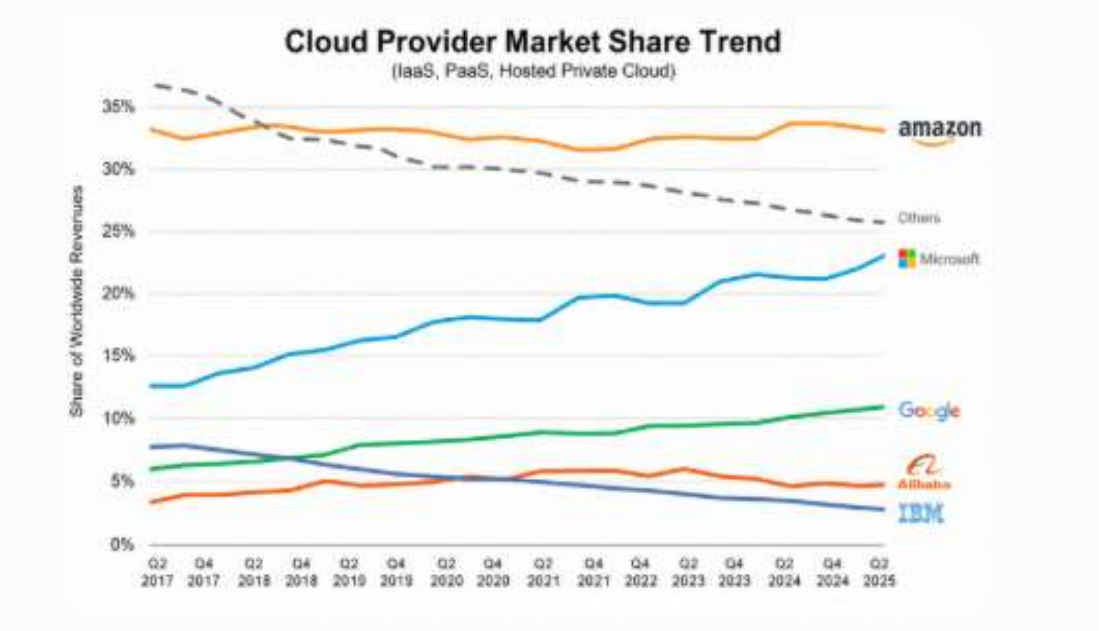


Рис. 2.6. Розподіл часток світового ринку хмарних послуг (2025 р.)

Джерело: сформовано автором на основі [20-21; 31].

Глобальна інфраструктура AWS представлена численними дата-центрами та регіонами розміщення по всьому світу, що дає змогу розгортати додатки максимально близько до кінцевих користувачів. Це сприяє підвищенню швидкодії, зниженню затримок і забезпеченню високого рівня відмовостійкості.

Окрему увагу в AWS приділено питанням інформаційної безпеки. Платформа пропонує комплексні механізми захисту, включаючи шифрування даних, засоби ідентифікації та автентифікації користувачів, а також інструменти моніторингу та управління безпекою.

Завдяки постійному розширенню функціональних можливостей AWS залишається одним із домінуючих гравців на ринку хмарних технологій, забезпечуючи підприємствам доступ до сучасних цифрових рішень для інноваційного розвитку.

Microsoft Azure є хмарною платформою, розробленою компанією Microsoft, яка надає широкий набір інфраструктурних і платформних сервісів для роботи в хмарному середовищі.

Azure включає різноманітні сервіси, такі як віртуальні машини (Virtual Machines), об’єктне сховище (Azure Blob Storage), реляційні бази даних

(Azure SQL Database), сервіси машинного навчання (Azure Machine Learning) та інструменти для аналітики великих даних (Azure Data Lake Analytics) тощо. Важливою перевагою платформи є її глибока інтеграція з екосистемою Microsoft, зокрема Windows Server, Active Directory та Microsoft 365.

Інфраструктура Azure представлена глобальною мережею дата-центрів, що дозволяє користувачам розміщувати дані та застосунки у найближчих регіонах, забезпечуючи високу доступність і продуктивність.

Порівняння хмарних платформ та обґрунтування вибору

AWS, Microsoft Azure та Google Cloud Platform є провідними гравцями ринку хмарних обчислень, кожен із яких має власні технологічні переваги та особливості застосування.

Водночас у межах даного дослідження обґрунтовано доцільність використання Microsoft Azure, що зумовлено його функціональною насиченістю та високим рівнем надійності.

Серед основних переваг Microsoft Azure можна виділити:

✓ масштабованість і надійність, що забезпечують стабільну роботу застосунків навіть при високих навантаженнях;

✓ широкий спектр сервісів, який охоплює обчислення, зберігання даних, бази даних, штучний інтелект, IoT та інші напрями;

✓ глибока інтеграція з екосистемою Microsoft, що забезпечує зручність використання в корпоративному середовищі та підвищує ефективність розробки й адміністрування систем.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

3.1. Впровадження хмарних сервісів

У процесі впровадження хмарних сервісів було обрано хмарну платформу Microsoft Azure для розгортання та адміністрування інфраструктурних і програмних ресурсів системи.

З метою впорядкування та централізованого управління створеними ресурсами сформовано окрему ресурсну групу. Такий підхід забезпечує логічне об’єднання пов’язаних компонентів, що, у свою чергу, спрощує їх адміністрування, моніторинг та подальше масштабування.

На рис. 3.1 наведено інтерфейс створення ресурсної групи в середовищі Microsoft Azure.

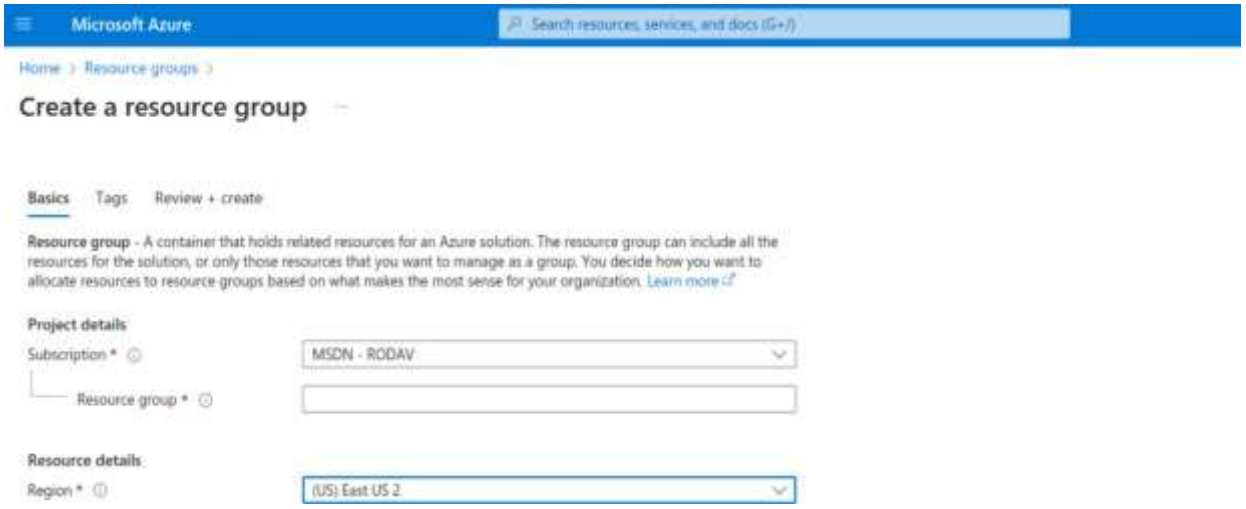


Рис. 3.1. Інтерфейс створення ресурсної групи в Microsoft Azure

Результат створення ресурсної групи зображено на рисунку 3.2. Для забезпечення ефективного розгортання інформаційної системи в середовищі Kubernetes-кластера було обрано сервіс Azure Kubernetes Service (AKS).

Azure Kubernetes Service (AKS) є керованою хмарною платформою для контейнерних застосунків, яка забезпечує можливість розгортання, адміністрування та масштабування контейнеризованих рішень із використанням оркестратора Kubernetes у хмарній інфраструктурі Microsoft Azure.

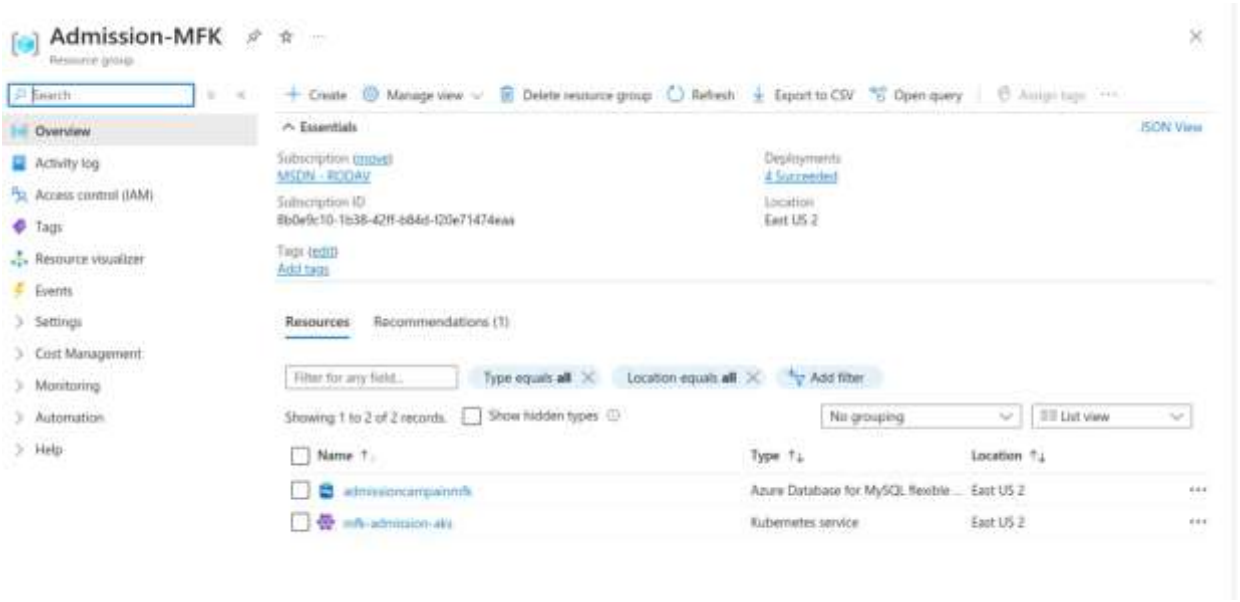


Рис. 3.2. Результат створення ресурсної групи в Microsoft Azure

Застосування AKS дозволило підвищити ефективність процесів розгортання та управління системою, забезпечивши її масштабованість, відмовостійкість і стабільність функціонування. Використання даного сервісу сприяє автоматизації ключових операцій інфраструктурного рівня, зокрема автоматичного масштабування ресурсів та підтримки високої доступності. Це дало змогу зосередити основні зусилля на розробці функціональної складової системи, мінімізувавши витрати на адміністрування інфраструктури. На рис. 3.3 наведено інтерфейс створення Kubernetes-кластера в середовищі AKS.

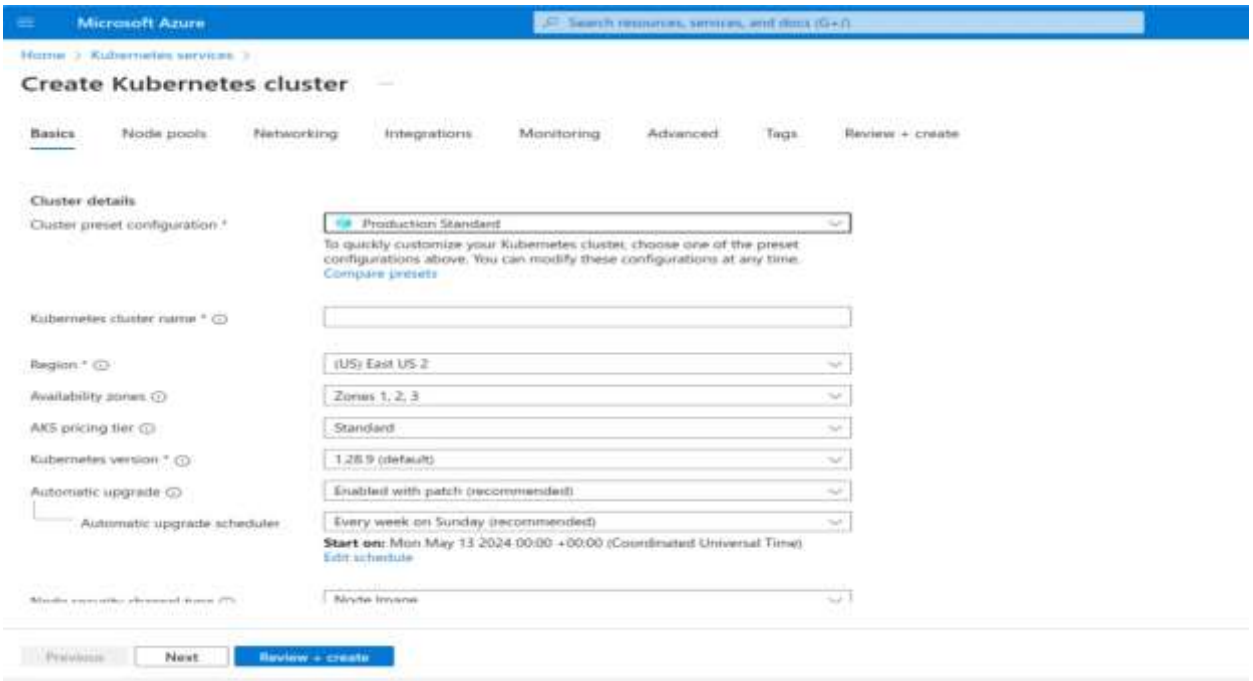


Рис. 3.3. Інтерфейс створення Kubernetes-кластера (AKS)

На рис. 3.4 наведено результат створення Kubernetes-кластера (AKS) для інформаційної системи.

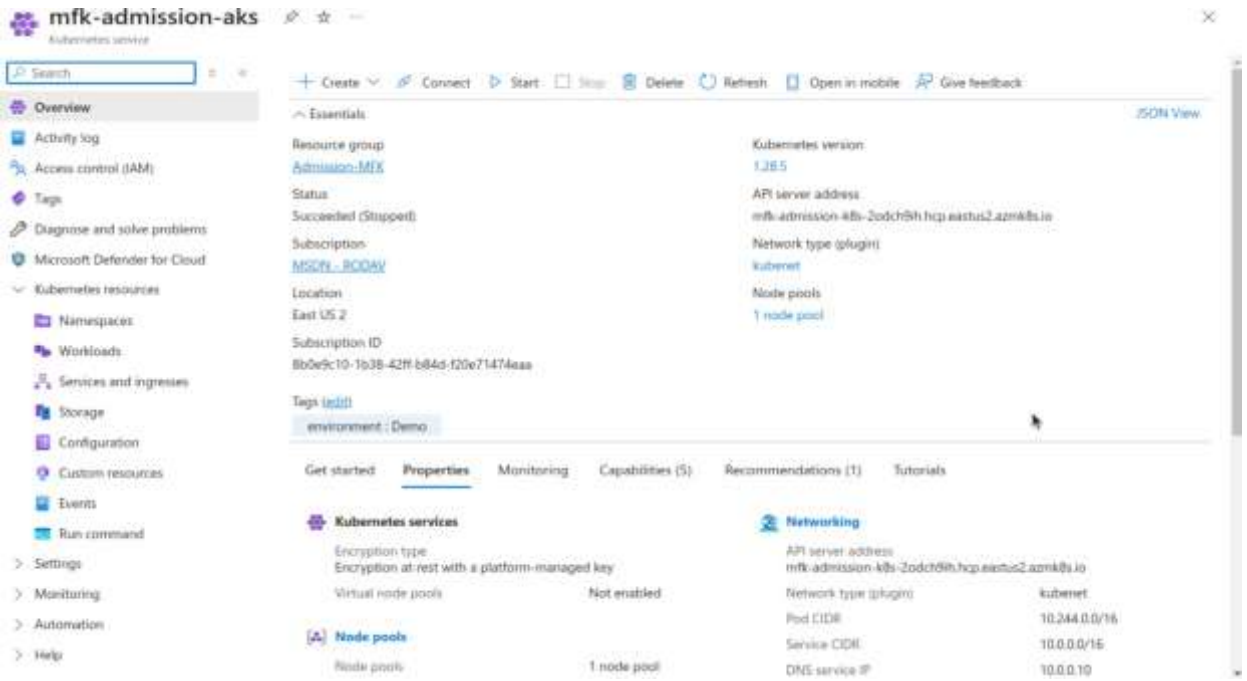


Рис. 3.4. Створений Kubernetes-кластер (AKS)

Застосування Azure Kubernetes Service стало одним із ключових чинників успішного впровадження інформаційної системи, оскільки забезпечило високу швидкість розгортання, надійність функціонування та масштабованість обчислювальних ресурсів.

Для забезпечення роботи інформаційної системи необхідною складовою є база даних, у зв’язку з чим було обрано сервіс Azure Database for MySQL.

Azure Database for MySQL є повністю керованим хмарним сервісом реляційної бази даних MySQL у середовищі Microsoft Azure [32-33; 40]. Даний сервіс забезпечує можливість створення, адміністрування та масштабування баз даних без необхідності управління фізичною інфраструктурою. Його використання дозволяє оперативно розгортати нові бази даних із забезпеченням високого рівня доступності, безпеки та продуктивності прикладних систем.

Крім того, сервіс автоматизує процеси резервного копіювання, моніторингу та технічного обслуговування, що дає змогу зосередити зусилля на розробці функціональної частини програмного забезпечення, мінімізуючи адміністративне навантаження.

Інтерфейс створення Azure Database for MySQL наведено на рис. 3.5.

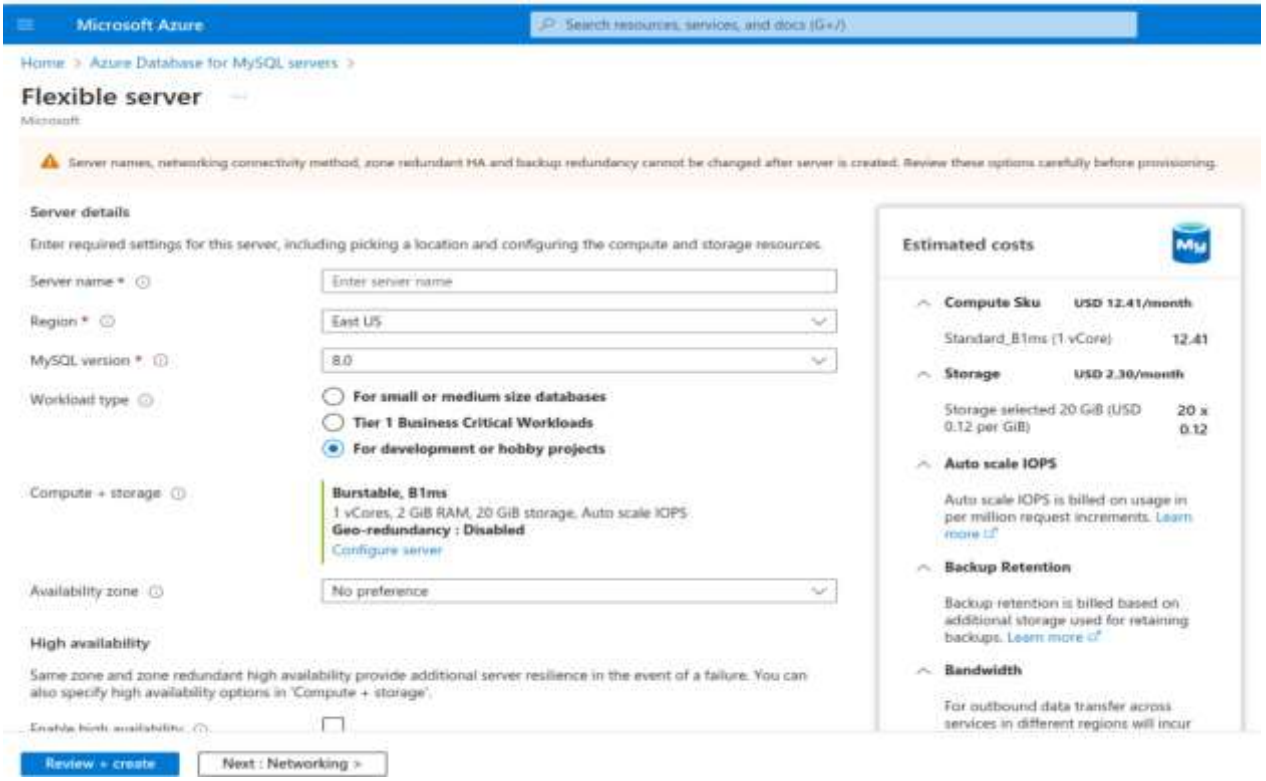


Рис. 3.5. Сторінка налаштування бази даних Azure Database for MySQL

На рис. 3.6 представлено результат створення бази даних у сервісі Azure Database for MySQL для інформаційної системи.

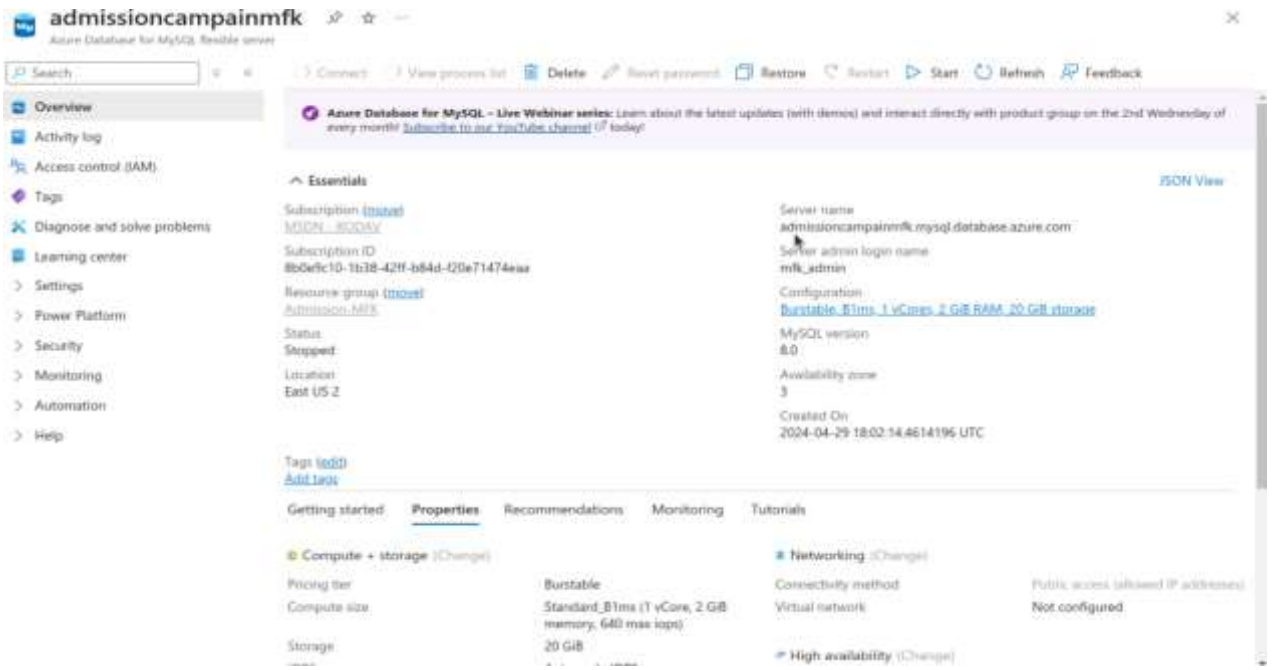


Рис. 3.6. Результат створення бази даних Azure Database for MySQL

Інтеграція сервісу Azure Database for MySQL забезпечила підвищений рівень безпеки та надійності зберігання даних, що використовуються в інформаційній системі. Даний сервіс надає розширені можливості адміністрування реляційних баз даних у хмарному середовищі, що сприяє досягненню високої продуктивності, відмовостійкості та доступності даних.

Узагальнюючи, застосування Azure Kubernetes Service у поєднанні з Azure Database for MySQL, а також коректна організація ресурсної групи в хмарній інфраструктурі Microsoft Azure, дозволили забезпечити надійну, масштабовану та ефективну архітектуру інформаційної системи.

3.2. Контейнеризація інформаційної системи

Для реалізації контейнеризації інформаційної системи в межах дослідження було використано платформу Docker, яка забезпечує створення ізольованого середовища виконання програмного забезпечення та спрощує процес його розгортання на різних обчислювальних платформах.

Об’єктом контейнеризації стала інформаційна система приймальної кампанії фахового коледжу, розроблена в межах попереднього кваліфікаційного дослідження. Використання контейнерних технологій дало змогу підвищити мобільність програмного продукту, спростити процес його встановлення та забезпечити однакові умови функціонування незалежно від характеристик серверного середовища.

Для формування контейнерного образу було розроблено конфігураційний файл Dockerfile, фрагмент якого наведено в додатку А. У файлі визначено послідовність дій, необхідних для створення готового до експлуатації середовища виконання вебдодатка.

На першому етапі як базове середовище використовується стабільна версія операційної системи Debian. Далі виконується оновлення системних репозиторіїв та встановлення необхідного програмного забезпечення, зокрема вебсервера Apache, інтерпретатора PHP, модуля інтеграції PHP з Apache, а також розширень для взаємодії з базами даних MySQL і використання механізму PDO.

Після інсталяції компонентів здійснюється активація модуля rewrite, який забезпечує підтримку механізмів маршрутизації та формування зрозумілих URL-адрес. Для застосування внесених змін виконується перезапуск вебсервера Apache [27-28; 40].

Наступним кроком визначається робочий каталог контейнера, яким виступає директорія */var/www/html/*. Саме в цьому каталозі розміщуються

файли вебзастосунку, що копіюються до контейнера під час побудови образу. Такий підхід забезпечує включення всіх необхідних програмних компонентів до складу контейнера та гарантує їхню доступність після запуску.

Для забезпечення доступу користувачів до вебресурсу в конфігурації контейнера відкривається порт 80, який використовується вебсервером Apache для обробки HTTP-запитів. Завершальним етапом є визначення команди запуску контейнера, що ініціює роботу вебсервера в активному режимі та підтримує безперервне функціонування застосунку.

Створення контейнерного образу здійснювалося за допомогою команди *docker build*, яка автоматично виконує всі інструкції, визначені у файлі Dockerfile. Результат успішної побудови образу наведено на рис. 3.7.



Рис. 3.7. Процес побудови контейнерного образу Docker

Після створення контейнерного образу наступним етапом стала його публікація в репозиторії Docker Hub. Використання Docker Hub забезпечує централізоване зберігання контейнерних образів та надає можливість їх подальшого поширення, оновлення й використання в різних середовищах розгортання.

Docker Hub являє собою хмарний сервіс для зберігання та керування Docker-образами, який дозволяє розробникам організовувати спільний доступ до контейнеризованих застосунків, автоматизувати процеси доставки програмного забезпечення та спрощувати розгортання інформаційних систем. Завдяки використанню цього сервісу забезпечується швидке

отримання необхідних образів із віддаленого репозиторію та їх подальше використання на серверах або в хмарній інфраструктурі.

Створений образ інформаційної системи було завантажено до репозиторію Docker Hub, що дозволило сформувати єдине джерело для його зберігання та подальшого розгортання. Такий підхід підвищує мобільність програмного продукту, спрощує процес його оновлення та забезпечує можливість швидкого масштабування системи в разі зростання навантаження.

Результат публікації контейнерного образу в Docker Hub наведено на рис. 3.8.

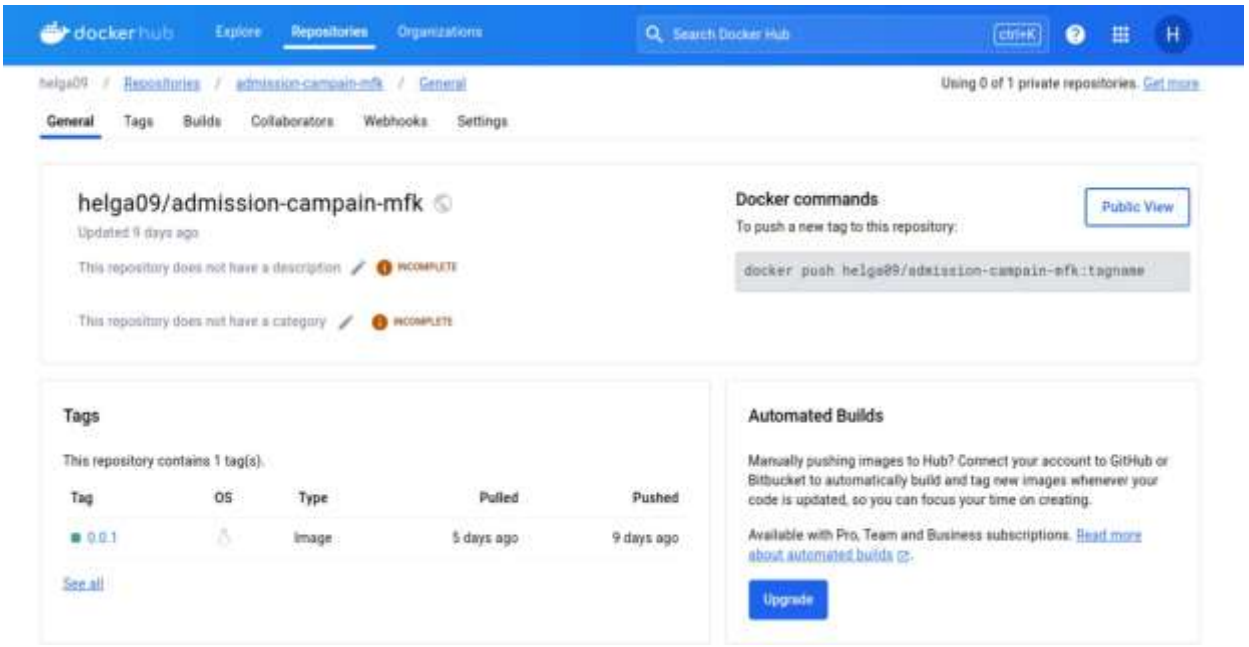


Рис. 3.8. Контейнерний образ інформаційної системи в Docker Hub

Розміщення контейнерного образу в репозиторії Docker Hub підтверджує успішне виконання процесу контейнеризації інформаційної системи та її готовність до подальшого розгортання, експлуатації й масштабування в різних обчислювальних середовищах.

3.3. Інтеграція контейнеризації та засобів управління ресурсами

Для реалізації підходу Infrastructure as Code (IaC) у межах роботи було використано платформу Pulumi, яка забезпечує можливість опису та керування інфраструктурними ресурсами за допомогою сучасних мов

програмування. На відміну від традиційних інструментів, що використовують декларативні конфігураційні файли, Pulumi дозволяє створювати інфраструктуру із застосуванням повноцінних мов програмування, що розширює можливості автоматизації та повторного використання коду.

Серед підтримуваних мов програмування для реалізації проєкту було обрано TypeScript. Такий вибір обумовлений високим рівнем типізації, зручністю розробки та широкими можливостями інтеграції з екосистемою JavaScript. Використання TypeScript сприяє підвищенню надійності програмного коду, спрощує супровід проєкту та зменшує ймовірність виникнення помилок під час розгортання інфраструктури.

Початковим етапом роботи стало встановлення Pulumi та налаштування робочого середовища відповідно до вимог платформи. Після завершення інсталяції було створено новий проєкт із використанням шаблону TypeScript, що забезпечило можливість програмного опису інфраструктурних компонентів та подальшого керування ними.

Наступним кроком стало налаштування взаємодії Pulumi із кластером Kubernetes. Для цього було виконано конфігурацію параметрів підключення та механізмів автентифікації, необхідних для доступу до ресурсів кластера через хмарну платформу Azure. Після успішної автентифікації система отримала можливість автоматизовано створювати, модифікувати та видаляти ресурси Kubernetes без необхідності ручного адміністрування.

Використання Pulumi у поєднанні з TypeScript дозволило реалізувати централізований підхід до управління інфраструктурою, за якого всі параметри розгортання визначаються програмним кодом. Це забезпечує відтворюваність конфігурацій, спрощує масштабування системи та підвищує ефективність процесів адміністрування контейнеризованих застосунків.

Після завершення налаштування проєкту була сформована структура каталогів і файлів, яка наведена на рис. 3.9. До складу проєкту входять основні конфігураційні та службові файли, необхідні для функціонування середовища розробки та керування інфраструктурою.

Зокрема, файл *.gitignore* містить перелік файлів і каталогів, що не підлягають відстеженню системою контролю версій Git. Основна логіка керування інфраструктурою реалізована у файлі *index.ts*, який містить програмний код для створення та налаштування ресурсів Kubernetes.

Каталог *node_modules* використовується для зберігання встановлених залежностей проєкту. Інформація про структуру проєкту, його залежності та службові сценарії міститься у файлі *package.json*, тоді як файл *package-lock.json* забезпечує фіксацію точних версій встановлених пакетів для гарантування відтворюваності середовища.

Конфігураційний файл *Pulumi.yaml* визначає основні параметри проєкту, включаючи назву стеку та налаштування взаємодії з хмарною інфраструктурою. Для керування процесом компіляції TypeScript використовується файл *tsconfig.json*, який містить параметри трансляції коду, налаштування модулів та правила перевірки типів.Таким чином, сформована структура проєкту забезпечує зручну організацію програмного коду та конфігураційних файлів, необхідних для автоматизованого управління інфраструктурою Kubernetes відповідно до концепції Infrastructure as Code.



Рис. 3.9. Структура проєкту Pulumi

Ключовим елементом проєкту є файл *index.ts*, який містить програмний код мовою TypeScript для автоматизованого створення та керування ресурсами Kubernetes за допомогою платформи Pulumi.

У даному файлі реалізовано основну логіку опису інфраструктури. Зокрема, виконуються імпорт необхідних бібліотек та модулів Pulumi і Kubernetes, оголошення змінних та об’єктів, а також визначення параметрів конфігурації для інфраструктурних компонентів. За допомогою програмного

коду задаються характеристики ресурсів, включаючи кількість реплік контейнерів, використовувані Docker-образи, мережеві порти, параметри середовища виконання та інші налаштування.

На основі визначених параметрів створюються та конфігуруються основні ресурси Kubernetes, зокрема *Deployment*, *Service*, *Secret*, *Namespace* та інші об’єкти, необхідні для функціонування контейнеризованого застосунку. Крім того, у файлі реалізовано експорт вихідних параметрів інфраструктури, що забезпечує можливість подальшого використання отриманих значень під час адміністрування та моніторингу системи.

Таким чином, файл *index.ts* виступає центральним компонентом проєкту та забезпечує програмне управління інфраструктурою відповідно до концепції Infrastructure as Code. Фрагмент програмного коду наведено в додатку Б.

Після завершення розробки інфраструктурного коду взаємодія з проєктом здійснюється через командний рядок Pulumi. Для керування життєвим циклом інфраструктури використовуються спеціалізовані команди, які дозволяють виконувати розгортання, оновлення, перевірку та видалення ресурсів.

Основною командою є *pulumi up*, яка виконує створення або оновлення інфраструктури відповідно до поточного стану програмного коду. Під час виконання команди система аналізує описані ресурси, визначає необхідні зміни та автоматично застосовує їх до цільового середовища.

Для попереднього аналізу майбутніх змін використовується команда *pulumi preview*. Вона дозволяє переглянути заплановані операції без фактичного внесення змін до інфраструктури, що сприяє підвищенню безпеки процесу розгортання.

Керування різними середовищами виконання здійснюється за допомогою команди *pulumi stack*. Використання стеків дає можливість підтримувати окремі конфігурації для середовищ розробки, тестування та експлуатації без дублювання програмного коду.

Для видалення створених ресурсів застосовується команда *pulumi destroy*, яка автоматично очищує інфраструктуру від усіх об’єктів,

розгорнутих за допомогою Pulumi. Перегляд експортованих параметрів виконується командою *pulumi stack output*, що дозволяє отримувати інформацію про створені ресурси та їхні характеристики.

Результат виконання процесу розгортання інфраструктури наведено на рисунку 3.10.

Після підтвердження операції Pulumi виконує всі необхідні зміни та здійснює автоматизоване розгортання інфраструктури. Після завершення процесу система формує звіт про результати виконання операції та відображає інформацію щодо успішно створених або модифікованих ресурсів.

Отже, використання Pulumi забезпечує ефективне управління інфраструктурою Kubernetes на основі програмного коду, підвищує рівень автоматизації процесів адміністрування та гарантує відтворюваність конфігурацій у різних середовищах розгортання.

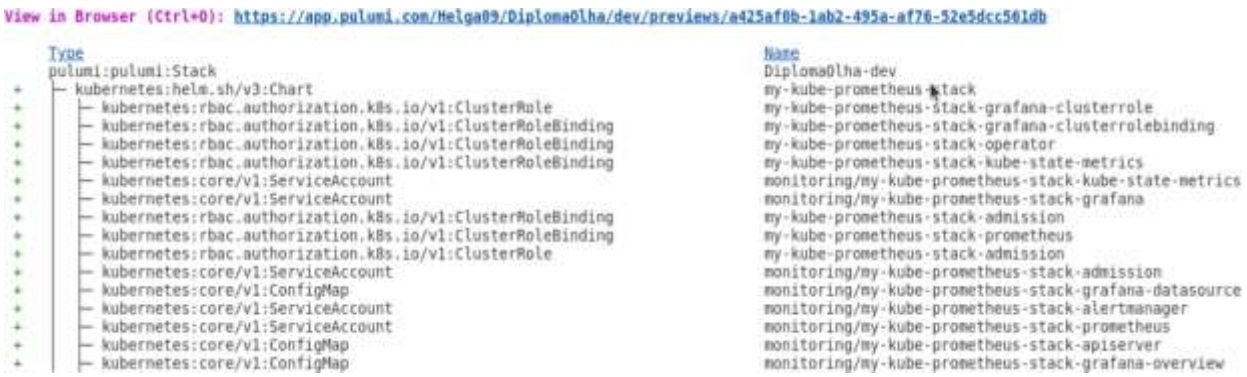


Рис. 3.10. Результат виконання команди pulumi up

За допомогою командного інтерфейсу Pulumi забезпечується ефективне керування життєвим циклом інфраструктури, зокрема виконання операцій зі створення, оновлення, перегляду та видалення ресурсів, описаних у програмному коді.

Окрім інструментів командного рядка, платформа Pulumi надає вебінтерфейс Pulumi UI, який забезпечує зручне графічне середовище для адміністрування інфраструктурних проєктів через браузер. Використання вебінтерфейсу спрощує процеси розроблення, розгортання та моніторингу інфраструктури, роблячи їх більш наочними та доступними.

Таким чином, Pulumi UI розширює можливості командного рядка, забезпечуючи зручний інструментарій для візуального контролю та адміністрування хмарної інфраструктури. На рис. 3.11 представлено приклад використання вебінтерфейсу Pulumi UI для перегляду та аналізу характеристик розгорнутих ресурсів.

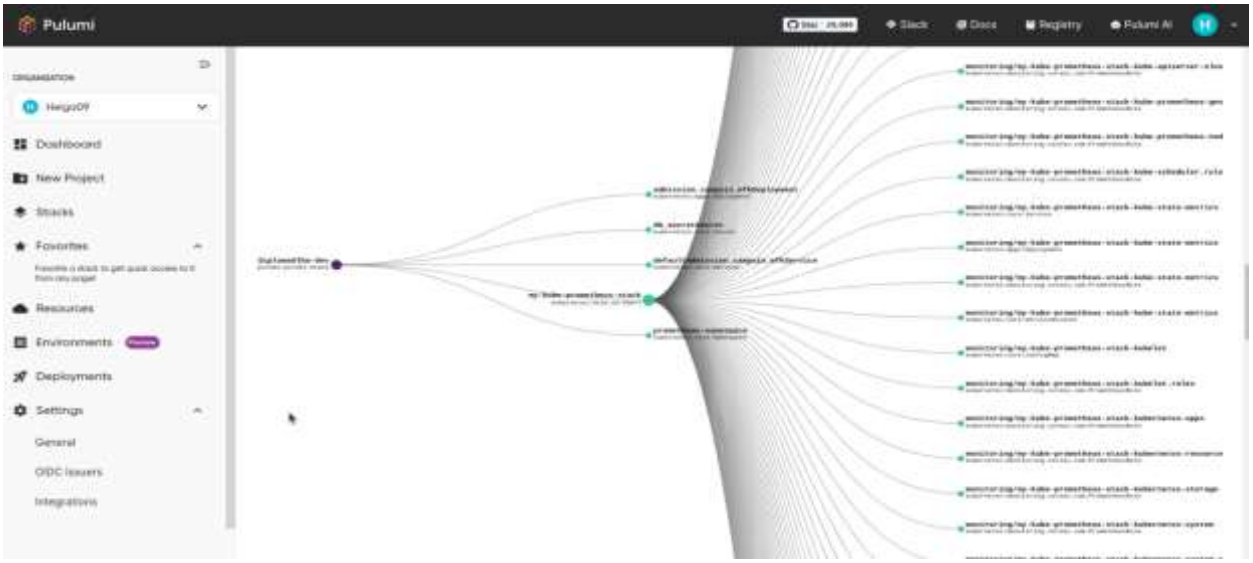


Рис. 3.11. Візуалізація та аналіз розгорнутих хмарних ресурсів у Pulumi UI

3.4. Інтеграція системи моніторингу

Автоматизацію процесу розгортання компонентів моніторингу реалізовано із застосуванням підходу Infrastructure-as-Code (IaC) за допомогою інструменту Pulumi. Використання IaC дозволяє описувати інфраструктурні ресурси програмним кодом, що спрощує їх супровід, відтворюваність та масштабування.

Однією з переваг Pulumi є підтримка роботи з Helm-чартами, які є стандартним механізмом керування застосунками та ресурсами в середовищі Kubernetes. Завдяки цьому забезпечується можливість автоматизованого створення, налаштування та розгортання необхідних компонентів безпосередньо через програмний код, що підвищує ефективність управління кластером та мінімізує ймовірність помилок під час конфігурації.

Для розгортання комплексної системи моніторингу було використано Helm-чарт Kube-prometheus-stack, який об’єднує набір інструментів для збору, зберігання та візуалізації показників функціонування Kubernetes-

кластера (рис. 3.12).

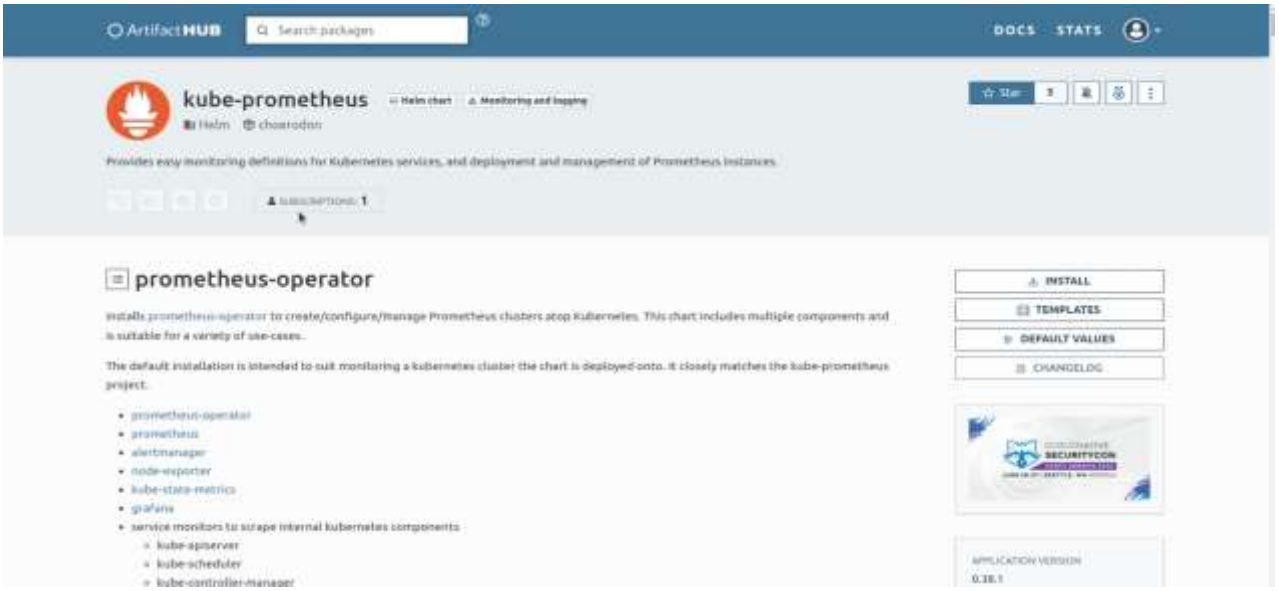


Рис. 3.12. Конфігурація та розгортання Kube-prometheus-stack

Kube-prometheus-stack являє собою комплексний Helm-чарт, призначений для автоматизованого розгортання та налаштування системи моніторингу в середовищі Kubernetes. Його використання дає змогу швидко впровадити інфраструктуру для збору, обробки, зберігання та візуалізації експлуатаційних метрик без необхідності окремого встановлення кожного компонента.

Код, використаний для розгортання Kube-prometheus-stack за допомогою Pulumi, наведено на рис. 3.13.

```
const prometheusNamespace = new k8s.core.v1.Namespace("prometheus-namespace", {
    metadata: {
        name: "monitoring",
    },
});

const prometheusChart = new k8s.helm.v3.Chart("my-kube-prometheus-stack", {
    namespace: prometheusNamespace.metadata.name,
    chart: "kube-prometheus-stack",
    version: "58.4.0",
    fetchOpts: {
        repo: "https://prometheus-community.github.io/helm-charts",
    },
    values: {
        grafana: {
            service: {
                type: "LoadBalancer",
            },
        }
    },
});
```

Рис. 3.13. Код розгортання Helm-чарта Kube-prometheus-stack за допомогою Pulumi

3.5. Програмна реалізація та аналіз результатів розгортання інфраструктури в Kubernetes-кластері

Після завершення процесу розгортання інфраструктури в кластері Azure Kubernetes Service (AKS) було виконано перевірку коректності створення та функціонування необхідних ресурсів Kubernetes. Для цього використано інструмент K9s – консольний засіб адміністрування Kubernetes, який забезпечує інтерактивну взаємодію з кластером через текстовий інтерфейс.

Використання K9s дає змогу оперативно отримувати інформацію про стан компонентів кластера, здійснювати моніторинг ресурсів, переглядати журнали подій, контролювати роботу контейнерів і подів, а також аналізувати поточний стан розгорнутих застосунків. Завдяки зручному інтерфейсу інструмент спрощує процес адміністрування та діагностики Kubernetes-середовища без необхідності виконання великої кількості команд через стандартний клієнт kubectl.

На рис. 3.14 наведено інтерфейс K9s із переліком активних подів, розгорнутих у Kubernetes-кластері, що підтверджує успішне створення та запуск компонентів інфраструктури.

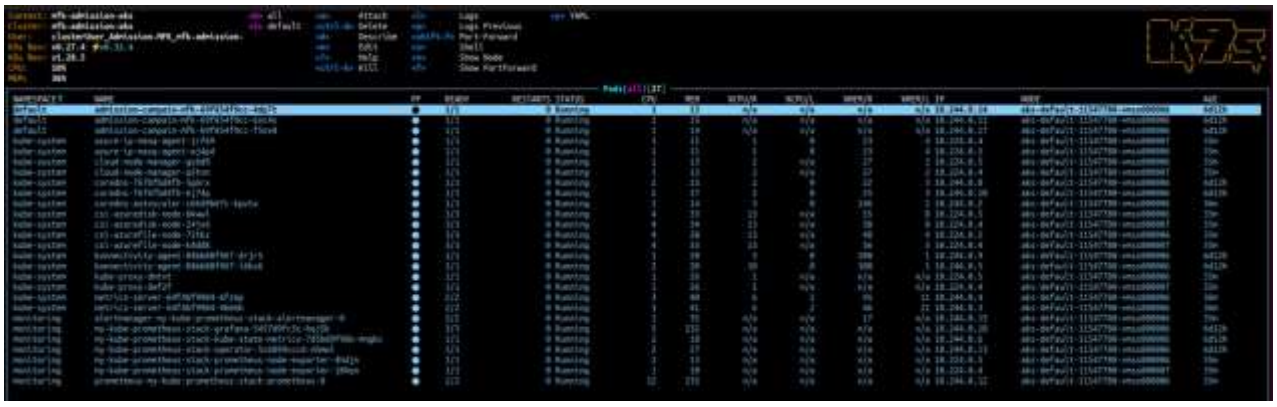


Рис. 3.14. Перелік активних подів Kubernetes-кластера в інтерфейсі K9s

Після аналізу результатів, наведених на рис. 3.14, можна зробити висновок, що всі поди перебувають у стані Running, що свідчить про успішне розгортання та коректне функціонування компонентів кластера. Такий стан підтверджує стабільну роботу контейнеризованих застосунків і є важливим показником надійності та працездатності Kubernetes-середовища.

Наступним етапом перевірки є аналіз створених сервісів Kubernetes. Особливу увагу приділено сервісам типу LoadBalancer, оскільки саме вони забезпечують зовнішній доступ до розгорнутих застосунків. Для

підтвердження коректності їх налаштування необхідно переконатися у призначенні зовнішніх IP-адрес та доступності відповідних сервісів з мережі Інтернет.

Перевірка наявності та стану сервісів дозволяє підтвердити успішність розгортання інфраструктури та готовність застосунків до експлуатації. На рис. 3.15 наведено результати виконання команди перегляду сервісів Kubernetes, що відображає створені мережеві ресурси та призначені їм IP-адреси.

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	admission-campaign-mfk	LoadBalancer	10.0.222.0	4.153.160.81	80:31702/TCP	7d2h
default	kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	8d
kube-system	kube-dns	ClusterIP	10.0.0.10	<none>	53/UDP,53/TCP	8d
kube-system	metrics-server	ClusterIP	10.0.211.185	<none>	443/TCP	8d
kube-system	my-kube-prometheus-stack-coredns	ClusterIP	None	<none>	9153/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-controller-manager	ClusterIP	None	<none>	10257/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-etcd	ClusterIP	None	<none>	2381/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-proxy	ClusterIP	None	<none>	10249/TCP	7d11h
kube-system	my-kube-prometheus-stack-kube-scheduler	ClusterIP	None	<none>	10259/TCP	7d11h
kube-system	my-kube-prometheus-stack-kubelet	ClusterIP	None	<none>	10250/TCP,10255/TCP,4194/TCP	8d
monitoring	alertmanager-operated	ClusterIP	None	<none>	9093/TCP,9094/TCP,9094/UDP	7d11h
monitoring	my-kube-prometheus-stack-alertmanager	ClusterIP	10.0.4.108	<none>	9093/TCP,8080/TCP	7d11h
monitoring	my-kube-prometheus-stack-grafana	LoadBalancer	10.0.106.258	4.153.20.255	80:30304/TCP	7d11h
monitoring	my-kube-prometheus-stack-kube-state-metrics	ClusterIP	10.0.45.30	<none>	8080/TCP	7d11h
monitoring	my-kube-prometheus-stack-operator	ClusterIP	10.0.149.71	<none>	443/TCP	7d11h
monitoring	my-kube-prometheus-stack-prometheus	ClusterIP	10.0.32.37	<none>	9090/TCP,8080/TCP	7d11h
monitoring	my-kube-prometheus-stack-prometheus-node-exporter	ClusterIP	10.0.142.54	<none>	9100/TCP	7d11h
monitoring	prometheus-operated	ClusterIP	None	<none>	9090/TCP	7d11h

Рис. 3.15. Перелік розгорнутих сервісів у Kubernetes-кластері

Аналіз результатів, наведених на рис. 3.15, свідчить про те, що всі сервіси типу LoadBalancer успішно отримали зовнішні IP-адреси. Це підтверджує коректність налаштування мережевої взаємодії в кластері та забезпечує можливість доступу до розгорнутих застосунків із зовнішніх мереж.

Після завершення процесу розгортання інфраструктури в кластері Azure Kubernetes Service (AKS) за допомогою коду TypeScript та інструменту Pulumi було сформовано набір вихідних параметрів (outputs), серед яких містяться зовнішні URL-адреси створених сервісів. Отримані адреси використовуються для доступу до компонентів інформаційної системи та взаємодії з ними через мережу Інтернет.

Приклад результатів, виведених Pulumi після виконання розгортання, наведено на рис. 3.16.

```
Outputs:
    externalIp     : "4.153.160.81/Main/html/index.php"
    externalIpAdmin: "4.153.160.81/Admin"
```

Рис. 3.16. Вихідні параметри (Outputs) розгортання інфраструктури в Pulumi

Перевірка відповідності отриманих URL-адрес розгорнутій системі дозволяє підтвердити її доступність та коректність функціонування.

На рис. 3.17 наведено результат звернення до інформаційної системи за відповідною URL-адресою, що підтверджує її коректну роботу та відповідь сервісу. Після переходу за відповідною URL-адресою адміністративної частини системи користувачу відображається сторінка автентифікації, яка забезпечує контроль доступу до керування ресурсами та функціоналом застосунку. Це є важливим елементом системи безпеки, оскільки дозволяє обмежити доступ до адміністративних можливостей лише авторизованим користувачам.

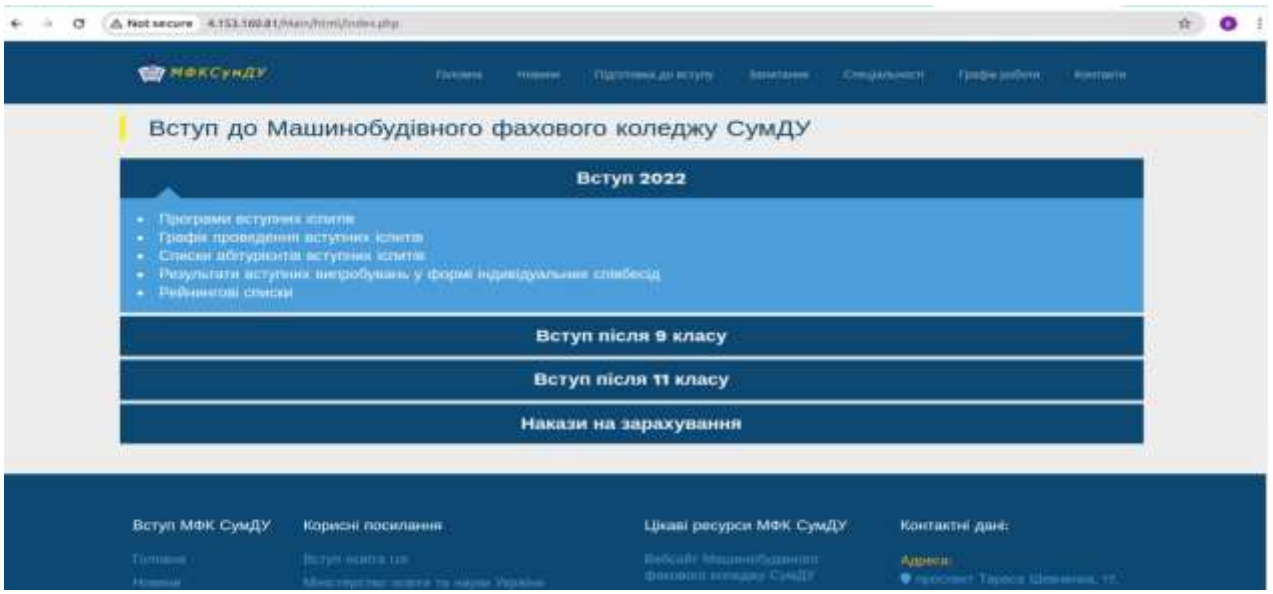


Рис. 3.17. Інтерфейс головної сторінки інформаційної системи

На рис. 3.18 наведено інтерфейс сторінки входу до адміністративної частини системи.



Рис. 3.18. Сторінка авторизації адміністративної частини

Після успішного проходження автентифікації в адміністративну частину системи користувач отримує доступ до головної сторінки інформаційної системи, яка наведена на рис. 3.19. Для підтвердження коректності розгортання системи моніторингу здійснено звернення до зовнішньої IP-адреси Grafana. Це дозволяє перевірити доступність сервісу та його готовність до роботи в межах розгорнутої інфраструктури.

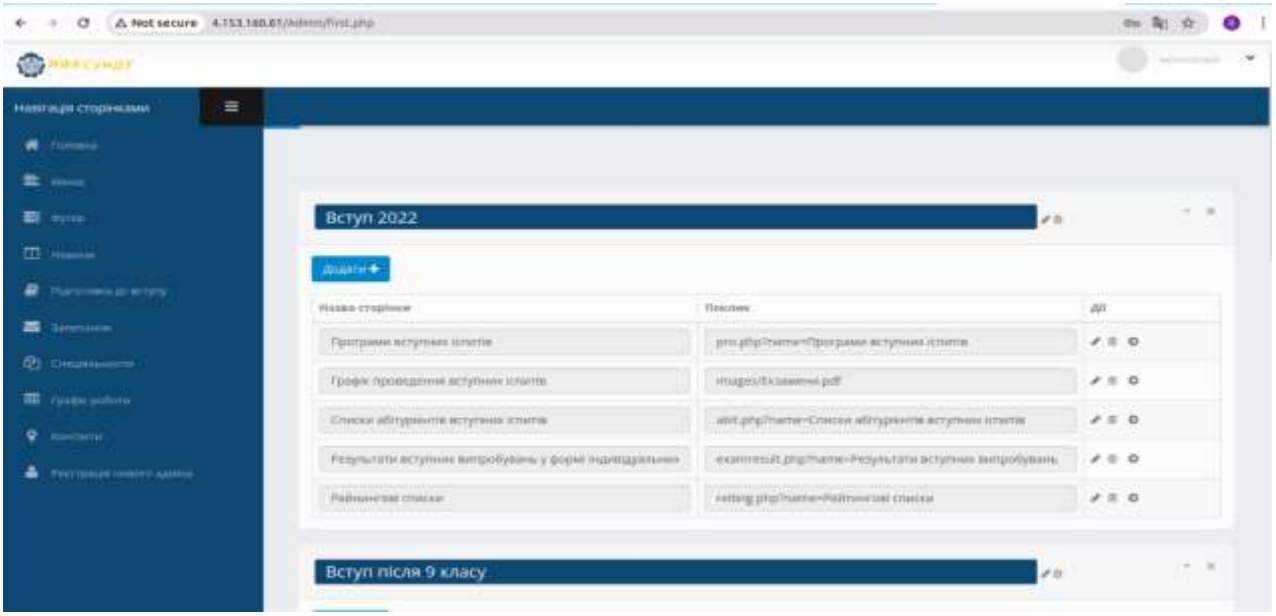


Рис. 3.19. Інтерфейс головної сторінки адміністративної частини інформаційної системи

На рис. 3.20 наведено інтерфейс сторінки автентифікації Grafana, що підтверджує успішне розгортання та доступність моніторингової платформи.



Рис. 3.20. Сторінка авторизації Grafana

Після успішної автентифікації в Grafana користувач отримує доступ до головної сторінки моніторингового сервісу, що містить основне навігаційне меню

та інструменти для роботи з дашбордами і метриками. Відповідний інтерфейс наведено на рис. 3.21. На рис. 3.22 наведено джерела даних, підключені до Grafana, які використовуються для збору та візуалізації метрик у межах моніторингової системи.

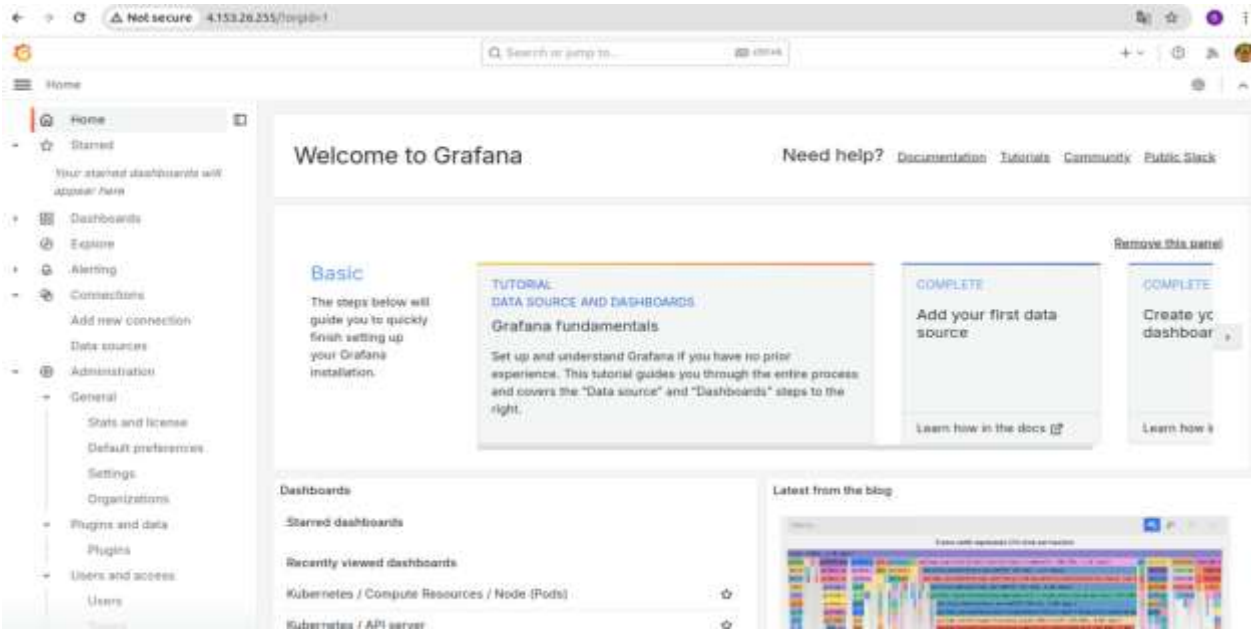


Рис. 3.21. Інтерфейс головної сторінки Grafana

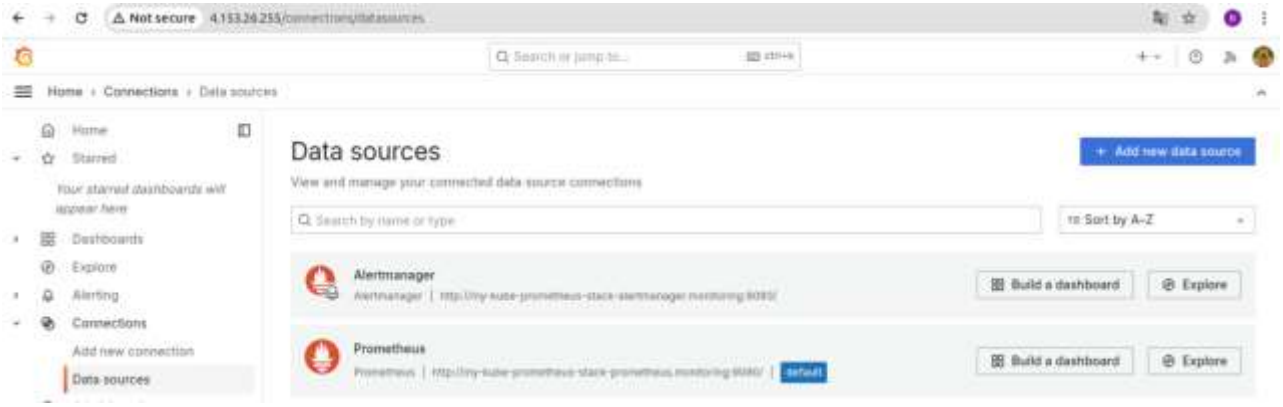


Рис. 3.22. Налаштування джерел даних у Grafana

На рис. 3.23 представлено розділ Dashboards у Grafana, який виконує роль центрального інструменту керування візуалізацією даних у системі моніторингу. У цьому розділі користувач має можливість створювати, редагувати та переглядати інформаційні панелі, графіки та звітні візуалізації, побудовані на основі даних, що надходять із різноманітних джерел, зокрема баз даних, систем моніторингу та інших інтегрованих сервісів. Такий функціонал забезпечує зручний аналіз метрик та сприяє оперативному оцінюванню стану інформаційної системи.

На рис. 3.24 представлено інформаційну панель Node Exporter / Nodes у Grafana, яка використовується для моніторингу ключових параметрів стану вузлів Kubernetes-кластера в режимі реального часу.

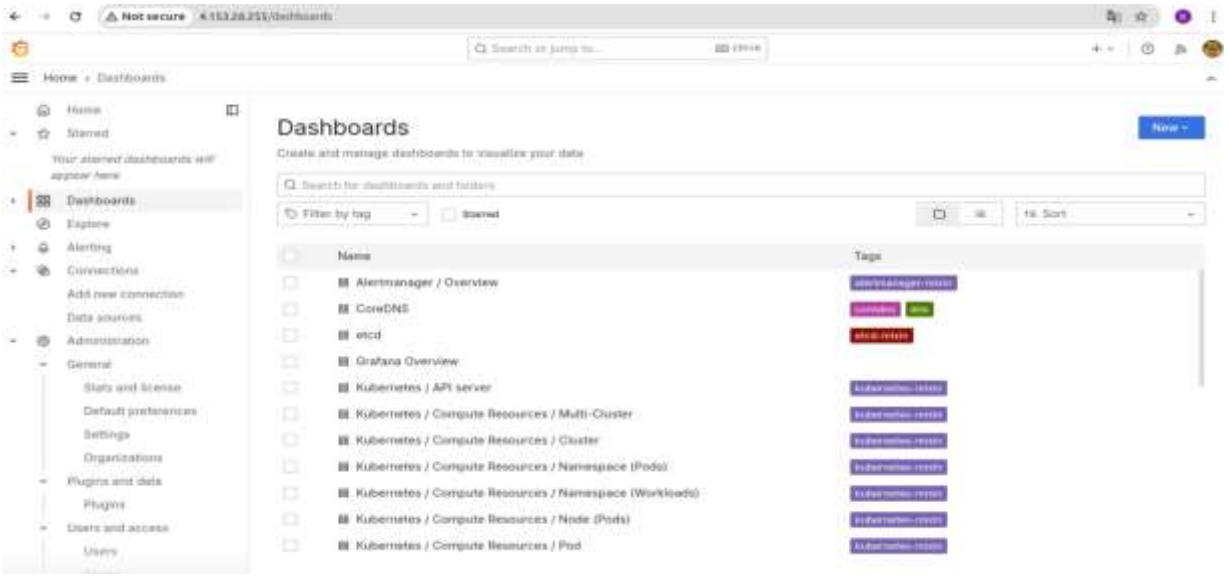


Рис. 3.23. Інтерфейс розділу Dashboards у Grafana

Даний дашборд забезпечує візуалізацію основних метрик продуктивності, зокрема навантаження на центральний процесор (CPU), використання оперативної пам’яті, стан дискової підсистеми та мережеву активність.

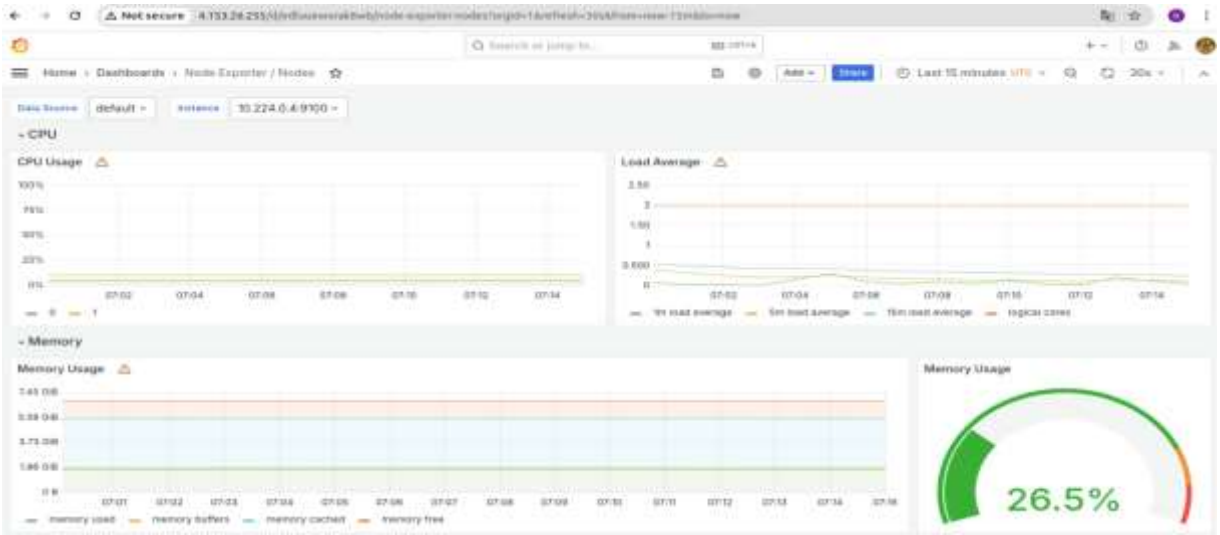


Рис. 3.24. Інформаційна панель «Node Exporter / Nodes» у Grafana

Використання даного дашборда забезпечує оперативний моніторинг стану системних ресурсів, що дозволяє своєчасно виявляти потенційні проблеми та реагувати на відхилення у роботі інфраструктури.

Інша інформаційна панель – «Kubernetes / Compute Resources /

Namespace (Pods)», представлена на рис. 3.25, призначена для аналізу використання обчислювальних ресурсів у межах визначеного простору імен (namespace) Kubernetes.

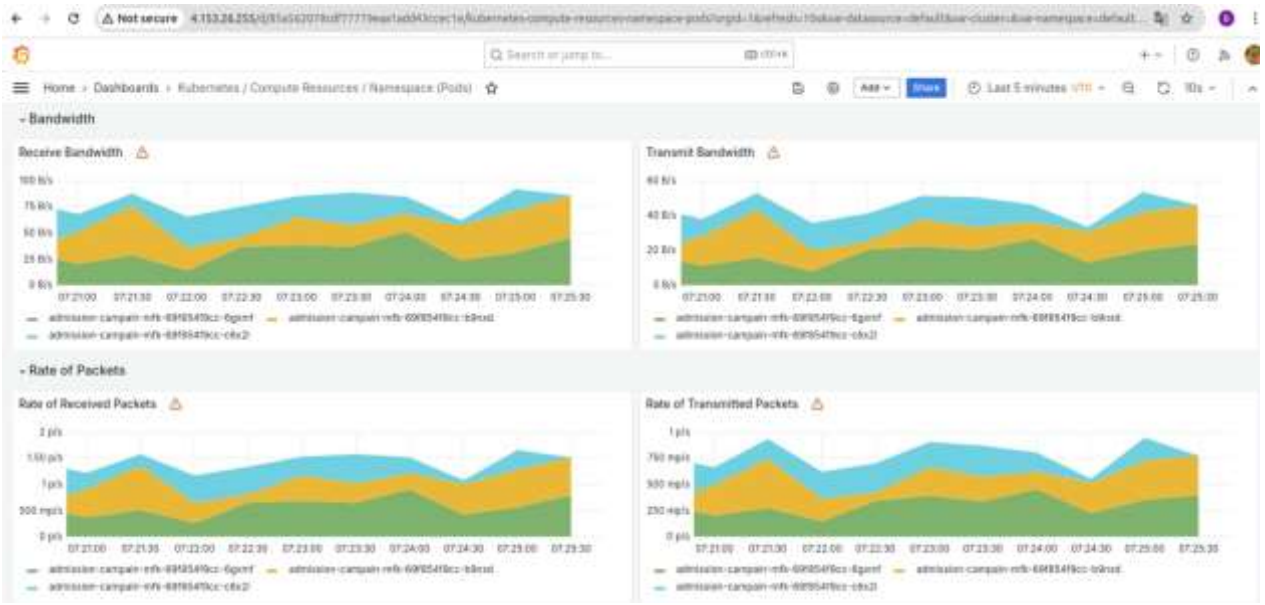


Рис. 3.25. Інформаційна панель «Kubernetes / Compute Resources / Namespace (Pods)» у Grafana

Даний дашборд забезпечує комплексний моніторинг використання ресурсів у межах визначеного простору імен Kubernetes, що дозволяє своєчасно виявляти потенційні проблеми та ефективно керувати ресурсами кластера.

Окрім розглянутих панелей, Grafana підтримує широкий набір інших дашбордів, які відображають різноманітні аспекти функціонування системи. Вони охоплюють метрики продуктивності, використання обчислювальних ресурсів, стан сервісів, мережеву активність, рівень навантаження, а також показники безпеки. Такі інструменти забезпечують візуалізацію даних у режимі реального часу або за визначений період, що підвищує ефективність аналізу стану інфраструктури.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи бакалавра здійснено комплексне дослідження сучасних підходів до управління контейнеризованими застосунками в Kubernetes-кластерах. Обґрунтовано актуальність використання контейнеризації в сучасних інформаційних системах, а також проаналізовано основні методи оркестрації та адміністрування таких середовищ.

Під час виконання роботи було реалізовано такі основні завдання:

1. Проаналізовано сучасні технології та інструменти контейнеризації, оркестрації, моніторингу та автоматизованого розгортання інфраструктури.
2. Розроблено інфраструктуру у вигляді коду (Infrastructure as Code), що дозволило автоматизувати процеси створення, налаштування та розгортання середовищ і забезпечити їх відтворюваність у будь-який момент часу.
3. Виконано інтеграцію Docker як базової технології для контейнеризації застосунків та їхнього розгортання.
4. Налаштовано та використано Kubernetes як оркестратор для керування життєвим циклом контейнеризованих застосунків.
5. Реалізовано інтеграцію системи моніторингу на базі Prometheus та Grafana, що забезпечує безперервний контроль стану системи та аналіз використання ресурсів у контейнеризованому середовищі.
6. Проведено аналіз результатів розгортання з метою перевірки коректності роботи інфраструктури, її продуктивності та поведінки під навантаженням.

Отже, можна дійти висновку, що розроблена система автоматизованого розгортання та керування контейнеризованими застосунками в Kubernetes-кластері є ефективним, масштабованим та перспективним рішенням для сучасних інформаційних систем, що потребують високої доступності та стабільності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What to choose? Virtualization or Containerization? [Електроний ресурс] - URL: https://dev.to/yogini16/what-to-choose-virtualization-or-containerization-1mjm (дата звернення: 10.03.2026).

2. Container Orchestration? [Електроний ресурс] - URL: https://www.wallarm.com/what/what-is-container-orchestration-7-benefits-and-4-best-tools (дата звернення: 11.02.2026).

3. What is Infrastructure as Code? [Електроний ресурс] - URL: https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code (дата звернення: 14.04.2026).

4. Data Engineering with Infrastructure as Code. [Електроний ресурс] - URL: https://www.linkedin.com/posts/ankitrathi_data-engineering-with-infrastructure-as-code-activity-7155447053467148288-noDQ/ (дата звернення: 22.03.2026).

5. What is infrastructure monitoring? [Електроний ресурс] - URL: https://www.ibm.com/topics/infrastructure-monitoring (дата звернення: 18.04.2024).

6. What is Cloud Computing? Types of Cloud-based Technologies and Services. [Електроний ресурс] - URL: https://blog.servercore.com/what-is-cloud-computing-types-of-cloud-based-technologies-and-services-9e0d0a49ae3b (дата звернення: 11.04.2026).

7. Docker. [Електроний ресурс] - URL: https://www.docker.com/ (дата звернення: 27.03.2026).

8. What is Podman? [Електроний ресурс] - URL: https://podman.io/ (дата звернення: 20.03.2026).

9. Docker Hub. [Електроний ресурс] - URL: https://hub.docker.com/ (дата звернення: 02.05.2026).

10. Docker Workflow. [Електроний ресурс] - URL: https://medium.com/@augustineozor/docker-workflow-b9fe71d32184 (дата звернення: 04.04.2026).

11. Docker: Up & Running. Turnbull J. *Docker: Up & Running: Shipping*

Reliable Containers in Production. 2rd ed. Sebastopol: O'Reilly Media, 2018. 352 p.

12. Kubernetes. [Електроний ресурс] - URL: https://kubernetes.io/docs/concepts/overview/ (дата звернення: 10.03.2026).

13. Kubernetes: Up and Running. Burns B., Beda J., Hightower K. *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. 3rd ed. Sebastopol: O'Reilly Media, 2022. 320 p.

14. OpenShift. [Електроний ресурс] - URL: https://www.redhat.com/en/technologies/cloud-computing/openshift (дата звернення: 11.03.2026).

15. Rancher. [Електроний ресурс] - URL: https://rancher.com/docs/rancher/latest/zh/ (дата звернення: 15.02.2026).

16. Amazon Web Services. [Електроний ресурс] - URL: https://aws.amazon.com/ (дата звернення: 12.03.2026).

17. Azure. [Електроний ресурс] - URL: https://azure.microsoft.com/ (дата звернення: 02.01.2026).

18. Google Cloud. [Електроний ресурс] - URL: https://cloud.google.com/ (дата звернення: 24.05.2026).

19. Architecture overview Kubernetes [Електроний ресурс] - URL: https://medium.com/@augustineozor/docker-workflow-b9fe71d32184 (дата звернення: 23.02.2026).

20. What is Jenkins? Architecture, Installation, and Setup Explained. [Електроний ресурс] - URL: https://www.lambdatest.com/blog/what-is-jenkins/ (дата звернення: 10.04.2026).

21. What is GitLab and How to Use It? [Електроний ресурс] - URL: https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab (дата звернення: 05.05.2026).

22. What Is GitHub and Why Should You Use It? [Електроний ресурс] - URL: https://www.coursera.org/articles/what-is-git (дата звернення: 08.05.2026).

23. What is IBM Cloud? Services Offered, Features & Pricing. [Електроний ресурс] - URL: https://www.datamation.com/cloud/ibm-cloud/ (дата звернення: 07.03.2026).

24. Getting started with Red Hat CodeReady Containers. [Електроний

ресурс] - URL: https://access.redhat.com/documentation/en-us/red_hat_codeready_containers/1.0/html/getting_started_guide/getting-started-with-codeready-containers_gsg (дата звернення: 11.03.2026).

25. Learning Kubernetes. Mouat A. *Learning Kubernetes*. 2nd ed. Sebastopol: O'Reilly Media, 2021. 366 p.

26. Operator pattern. [Електроний ресурс] - URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ (дата звернення: 11.03.2026).

27. Kubernetes Service Mesh Definition. [Електроний ресурс] - URL: https://avinetworks.com/glossary/kubernetes-service-mesh/ (дата звернення: 18.02.2026).

28. Serverless on Kubernetes. [Електроний ресурс] - URL: https://medium.com/appvia/serverless-on-kubernetes-63b49aeaf4ef (дата звернення: 22.03.2026).

29. Kubernetes extension. [Електроний ресурс] - URL: https://quarkus.io/guides/deploying-to-kubernetes (дата звернення: 22.03.2026).

30. CNCF. [Електроний ресурс] - URL: https://www.cncf.io/ (дата звернення: 26.04.2026).

31. Terraform. [Електроний ресурс] - URL: https://www.terraform.io/ (дата звернення: 30.03.2026).

32. HashiCorp. [Електроний ресурс] - URL: https://developer.hashicorp.com/terraform/docs (дата звернення: 13.05.2026).

33. Creating a Highly Available Architecture with Terraform. [Електроний ресурс] - URL: https://www.hashicorp.com/ (дата звернення: 05.05.2024).

34. Mozilla Public License. [Електроний ресурс] - URL: https://www.mozilla.org/en-US/MPL/ (дата звернення: 09.02.2026).

35. BSL stands for Business Source License. [Електроний ресурс] - URL: https://fossa.com/blog/business-source-license-requirements-provisions-history/ (дата звернення: 11.04.2026).

36. Pulumi. [Електроний ресурс] - URL: https://byjus.com/chemistry/hydrochloric-acid/ (дата звернення: 15.03.2026).

37. Python. [Електроний ресурс] - URL: https://www.python.org/ (дата звернення: 14.03.2026).

38. JavaScript. [Електроний ресурс] - URL: https://uk.javascript.info/ (дата звернення: 22.03.2026).

39. TypeScript. [Електроний ресурс] - URL: https://www.typescripttutorial.net/ (дата звернення: 24.02.2026).

40. Go. [Електроний ресурс] - URL: https://go.dev/ (дата звернення: 25.04.2026).

41. C#. [Електроний ресурс] - URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/ (дата звернення: 29.04.2026).

42. Java. [Електроний ресурс] - URL: https://www.java.com/en/ (дата звернення: 03.05.2026).

43. Kubernetes Pulumi. [Електроний ресурс] - URL: https://www.pulumi.com/registry/packages/kubernetes/ (дата звернення: 10.02.2026).

44. Monitoring Tools in Kubernetes: An Overview of Available Options and Integration. [Електроний ресурс] - URL: https://medium.com/@prateek.malhotra004/monitoring-tools-in-kubernetes-an-overview-of-available-options-and-integration-812c4c64b8b6 (дата звернення: 09.04.2026).

45. 11 Top Cloud Service Providers Globally In 2023. [Електроний ресурс] - URL: https://www.cloudzero.com/blog/cloud-service-providers (дата звернення: 12.04.2026).

Додаток А

Dockerfile

```
FROM debian:stable

RUN apt-get update && apt-get install -y \
    apache2 \
    php \
    libapache2-mod-php \
    php-mysql \
    php-cli \
    php-pdo

RUN a2enmod rewrite
RUN service apache2 restart

WORKDIR /var/www/html/
COPY . .

EXPOSE 80

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Рис. 1. Код Dockerfile для контейнеризації застосунку

Додаток Б

index.ts × Pulumi.yaml

index.ts > admission_campain_mfkDeployment

```
import * as k8s from "@pulumi/kubernetes";

const labels = { app: "admission-campain-mfk" };

const admission_campain_mfkDeployment = new k8s.apps.v1.Deployment("admission_campain_mfkDeployment", {
    apiVersion: "apps/v1",
    kind: "Deployment",
    metadata: { name: labels.app },
    spec: {
        replicas: 3,
        selector: { matchLabels: labels },
        template: {
            metadata: { labels },
            spec: {
                containers: [{
                    name: labels.app,
                    image: "helga09/admission-campain-mfk:0.0.1",
                    ports: [{ containerPort: 80 }],
                    env: ["DB_HOST", "DB_USER", "DB_PASSWORD", "DB_DATABASE"].map(name => ({
                        name,
                        valueFrom: { secretKeyRef: { name: "db-secret", key: name } },
                    })),
                }],
            },
        },
    },
});

const defaultAdmission_campain_mfkService = new k8s.core.v1.Service("defaultAdmission_campain_mfkService", {
    apiVersion: "v1",
    kind: "Service",
    metadata: { name: labels.app, namespace: "default" },
    spec: {
        allocateLoadBalancerNodePorts: true,
        ipFamilies: ["IPv4"],
        ipFamilyPolicy: "SingleStack",
        ports: [{ port: 80, protocol: "TCP", targetPort: 80 }],
        selector: labels,
        sessionAffinity: "None",
        type: "LoadBalancer",
```

Рис. 1. Код розгортання інфраструктури за допомогою Pulumi