Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів та технологій автоматизації CI\CD процесів для хмарних інформаційних систем
(тема)

Виконав:
студент 2 курсу, групи ІУСТм-22-1
Сергій КОТВИЦЬКИЙ
(Власне ім'я ПРІЗВИЩЕ)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні управляючі системи та технології
(повна назва освітньої програми)

Керівник доцент каф. ІУС Олег КОБИЛІН
(посада, власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту

Зав. кафедри _____ Костянтин ПЕТРОВ
(підпис) (Власне ім'я ПРІЗВИЩЕ)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні управляючі системи та технології
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«_20_» _листопада_ 20 23_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Котвицькому Сергію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів та технологій автоматизації CI\CD процесів для хмарних інформаційних систем
затверджена наказом університету від 16 листопада 2023 р. № 1359Ст
2. Термін подання студентом роботи до екзаменаційної комісії 14 01 2024 р.
3. Вихідні дані до роботи технології побудови хмарних інформаційних систем, технології побудови CI\CD процесів для хмарних інформаційних систем, сучасні рішення автоматизації CI\CD процесів для хмарних інформаційних систем
4. Перелік питань, що потрібно опрацювати в роботі виконати аналіз проблем автоматизації CI\CD процесів для хмарних інформаційних систем, провести дослідження провідних методів та технологій побудови CI\CD процесів для хмарних інформаційних систем, визначити існуючі проблеми та задачі автоматизації CI\CD процесів для хмарних інформаційних систем, сформувати рольові вимоги автоматизації CI\CD процесів для хмарних інформаційних систем, сформувати загальні вимоги автоматизації CI\CD процесів для хмарних інформаційних систем, розробити метод автоматизації CI\CD процесів для хмарних інформаційних систем, виконати апробацію розробленого методу автоматизації CI\CD процесів для хмарних інформаційних систем

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на дипломне проектування	20.11.2023	виконано
2	Аналіз завдання, літератури та аналогів з теми дипломної роботи	21.11.2023 – 02.12.2023	виконано
3	Постановка задачі дослідження	03.12.2023 – 05.12.2023	виконано
4	Аналіз проблем автоматизації CI\CD процесів для хмарних інформаційних систем	06.12.2023 – 15.12.2023	виконано
5	Дослідження методів та технологій побудови CI\CD процесів для хмарних інформаційних систем	16.12.2023 – 21.12.2023	виконано
6	Розробка методу автоматизації CI\CD процесів для хмарних інформаційних систем	22.12.2023 – 27.12.2023	виконано
7	Апробація методу автоматизації CI\CD процесів для хмарних інформаційних систем	28.12.2023 – 31.12.2023	виконано
8	Оформлення пояснювальної записки	01.01.2024 – 08.01.2024	виконано
9	Оформлення графічної частини та презентаційних матеріалів захисту	09.01.2024 – 11.01.2024	виконано
10	Представлення на рецензування	12.01.2024	виконано
11	Представлення кваліфікаційної роботи до ЕК	14.01.2024	виконано
12	Захист	16.01.2024	виконано

Дата видачі завдання 20 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доцент каф. ІУС, Олег КОБИЛІН
(підпис) (посада, власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи містить: 110 с., 3 розділи, 35 рис., 8 табл., 21 джерел, 2 додатки.

АВТОМАТИЗАЦІЯ, АНАЛІЗ, БЕЗПЕРЕРВНА ІНТЕГРАЦІЯ, БЕЗПЕРЕРВНЕ РОЗГОРТАННЯ, КОНФІГУРАЦІЯ, МАНІФЕСТ, МЕТОДИ, ПРОЦЕС, СКРИПТ, ХМАРНА СИСТЕМА.

У роботі виконано огляд методів та технологій автоматизації CI\CD процесів для хмарних інформаційних систем. На підставі проведеного аналізу запропоновано метод автоматизації CI\CD процесів для хмарних інформаційних систем.

Об’єктом дослідження в рамках магістерської кваліфікаційної роботи є CI\CD процеси для хмарних інформаційних систем.

Предмет дослідження: методи та технології автоматизації CI\CD процесів для хмарних інформаційних систем.

В результаті роботи було:

− виконано аналіз проблем автоматизації CI\CD процесів для хмарних інформаційних систем;

− досліджені методи та технології побудови CI\CD процесів для хмарних інформаційних систем;

− сформовано загальні та рольові вимоги автоматизації CI\CD процесів для хмарних інформаційних систем;

− розроблено метод автоматизації CI\CD процесів для хмарних інформаційних систем;

− виконано апробацію розробленого методу автоматизації CI\CD процесів для хмарних інформаційних систем.

ABSTRACT

The explanatory note to the qualifying work contains: 110 pages, 3 sections, 35 figures, 8 tables, 21 sources, 2 appendices.

ANALYSIS, AUTOMATION, CLOUD SYSTEM, CONFIGURATION, CONTINUOUS DEPLOYMENT, CONTINUOUS INTEGRATION, MANIFEST, METHODS, PROCESS, SCRIPT.

The work includes an overview of methods and technologies for automating CI\CD processes for cloud-based information systems. Based on the analysis, a method for automating CI\CD processes for cloud-based information systems is proposed.

The object of research within the master's qualifying work is CI\CD processes for cloud-based information systems.

The subject of the research: methods and technologies of automating CI\CD processes for cloud-based information systems.

The work results:

− Analysis of the problems of automating CI\CD processes for cloud-based information systems has been performed.

− Methods and technologies for building CI\CD processes for cloud information systems have been researched.

− General and role-based requirements for automating CI\CD processes for cloud information systems have been defined;

− The method for automating CI\CD processes for cloud-based information systems has been developed;

− The developed method for automating CI\CD processes for cloud information systems has been approved.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ІТ – інформаційні технології

ІС – інформаційна система

ХІС – хмарна інформаційна система

ПЗ – програмне забезпечення

ОС – операційна система

IaC – інфраструктура як код (англ., Infrastructure as Code)

CI – безперервна інтеграція (англ., Continuous Integration)

CD – безперервна доставка (англ., Continuous Delivery)

CD – безперервне розгортання (англ. Continuous Deployment)

IaaS – інфраструктура як послуга (англ. Infrastructure as a Service)

PaaS – платформа як послуга (англ. Platform as a Service)

SaaS – програмне забезпечення як послуга (англ. Software as a Service)

API – програмний інтерфейс (англ. Application Programming Interface)

VM – віртуальна машина (англ. Virtual Machine)

GCP – Google Cloud Platform

AWS – Amazon Web Services

Azure – Microsoft Azure

OCI – Oracle Cloud Infrastructure

GKE – Google Kubernetes Engine

ВСТУП

На сучасному етапі розвитку інформаційних технологій все більшої популярності набувають гнучкі підходи до розробки та впровадження програмного забезпечення (ПЗ). Це пов’язано зі зростаючими вимогами бізнесу щодо швидкості виходу на ринок нових продуктів та функцій [1]. Одним з таких підходів є концепція безперервної інтеграції та безперервної доставки (CI\CD).

Безперервна інтеграція (CI) автоматизує збірку, тестування та інтеграцію змін коду у спільному репозиторії, що сприяє виявленню та вирішенню інтеграційних проблем. Безперервна доставка (CD) полягає у автоматичній доставці змін коду в аналогічні виробничому середовища, де вони очікують остаточного схвалення. Безперервне розгортання, також CD, автоматично розгортає зміни коду безпосередньо в виробничому середовищі, усуваючи потребу в ручному схваленні кожної зміни.

Застосування CI\CD значно пришвидшує розробку нових версій продукту, дозволяє скоротити час виходу оновлень на ринок, підвищити якість коду завдяки автоматизованому тестуванню. Однак побудова ефективних автоматизованих CI\CD конвеєрів є непростим завданням.

Незважаючи на наявність значної кількості наукових праць, присвячених питанням автоматизації процесів CI\CD досі не запропоновано універсальних методів, які б повною мірою враховували специфіку різних проектів та технологій. Недостатньо вивчені аспекти адаптації процесів CI\CD під особливості та потреби конкретних ІС, зокрема, хмарних. Тому розробка нових методів автоматизації CI\CD процесів є актуальною задачею, що має важливе наукове та прикладне значення.

Кваліфікаційна робота присвячена дослідженню методів та технологій автоматизації CI\CD процесів для хмарних ІС.

Актуальність роботи визначається потребою в розробці ефективних методів автоматизації CI\CD процесів для хмарних ІС. Автоматизація CI\CD має забезпечити швидке внесення змін в систему з мінімальними витратами ресурсів.

В ході роботи необхідно провести аналіз предметної області, огляд сучасних підходів до автоматизації CI\CD процесів, виявити проблеми, які існують. Дослідити технології побудови CI\CD конвеєрів, сформулювати вимоги до розроблюваного методу та виконати опис постановки задачі дослідження. Розробити новий метод автоматизації CI\CD процесів для хмарних ІС, реалізувати його з використанням сучасних технологій та провести апробацію для оцінки ефективності.

Предмет дослідження: методи та технології автоматизації CI\CD процесів для хмарних ІС.

Мета проведення досліджень: пошук та розробка ефективного методу автоматизації процесів CI\CD для хмарних ІС.

Наукова новизна: метод автоматизації CI\CD процесів для хмарних ІС.

Практична цінність: розроблений метод дозволяє підвищити продуктивність CI\CD процесів для хмарних ІС, шляхом інтеграції різних інструментів та автоматизації кроків, що призводить до скорочення часу розробки і забезпечує ефективне впровадження нових функцій або виправлення помилок в ІС.

Перелік задач: проведення аналізу проблем автоматизації CI\CD процесів для хмарних ІС, проведення дослідження використання сучасних методів автоматизації CI\CD процесів, проведення дослідження використання передових технологій та інструментальних засобів для автоматизації CI\CD процесів, визначення структури автоматизації CI\CD процесів для хмарних систем, формування критеріїв вирішення задач автоматизації CI\CD процесів згідно обраних цілей, розробка методу автоматизації CI\CD процесів для хмарних ІС, розробка технології для автоматизації CI\CD процесів та виконання апробації розробленого методу.

1 АНАЛІЗ ПРОБЛЕМИ АВТОМАТИЗАЦІЇ CI\CD ПРОЦЕСІВ ДЛЯ ХМАРНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Огляд сучасних концепцій автоматизації CI\CD процесів для хмарних інформаційних систем

Автоматизація CI\CD процесів змінює підходи до розробки та впровадження програмного забезпечення, пропонуючи нові можливості для підвищення ефективності, зниження ризиків та оптимізації ресурсів. У роботі, особлива увага приділяється аналізу впливу цих процесів на швидкість, надійність та безперервність роботи хмарних систем, а також розглядаються виклики та можливості, які вони створюють для розвитку технологічних компаній.

Аналіз практик впровадження CI\CD процесів у технологічних компаніях, які користуються хмарними платформами, виявив, що процеси інтеграції та доставки програмного забезпечення часто налагоджують і підтримують ручними методами, або за допомогою базових скриптів. Подібний підхід призводить до низки проблем: відсутності єдиного централізованого CI\CD конвеєру, вразливості до помилок оператора, складнощів підтримки та оновлення скриптів, затримок у внесенні змін до хмарного середовища, неможливості реалізації автоматичного тестування та швидкого відкату змін [2].

Для подолання зазначених обмежень, перспективною є розробка та впровадження комплексного підходу до автоматизації CI\CD шляхом створення єдиного конвеєру збірки, тестування та постачання ПЗ у хмару. Ключовими процесами при цьому є розробка скриптів автоматизації, налаштування CI\CD середовища, інтеграція з хмарними сервісами, конфігурування конвеєру та моніторинг його ефективності. Необхідними функціями системи автоматизації є можливість автоматичного збирання коду, безперервного тестування, розгортання у хмарі, моніторингу та

генерування звітності. Застосування такого комплексного рішення дозволить суттєво збільшити швидкість, надійність і безпеку CI\CD процесів для хмарних інформаційних систем [3].

Щоб побудувати таке рішення, необхідно мати чітке розуміння архітектури хмарних інформаційних систем, адже кожен їхній компонент може вимагати свого особливого підходу до реалізації безперервної інтеграції та доставки оновлень.

1.2 Аналіз архітектури хмарних інформаційних систем

Хмарні інформаційні системи (ХІС) відіграють критичну роль у сучасному цифровому просторі, забезпечуючи гнучкість, масштабованість та ефективність обчислювальних ресурсів. Ці системи дають можливість користувачам зберігати, обробляти та управляти даними в інтернет-середовищі, відмовляючись від необхідності власної інфраструктури та обладнання.

Хмарні ІС базуються на концепції "як послуга" (as-a-Service), що включає основні моделі, такі як інфраструктура як послуга (IaaS), платформа як послуга (PaaS), програмне забезпечення як послуга (SaaS), та інші [4]. Кожна з цих моделей пропонує різні рівні контролю і гнучкості, дозволяючи адаптацію до унікальних вимог бізнесу.

Розглянемо ключові характеристики архітектури для ХІС.

Мульти-орендованість (Multi-tenancy) – дозволяє багатьом користувачам (орендарям) використовувати спільну інфраструктуру та ресурси, забезпечуючи при цьому ізоляцію даних та конфігурацій.

Еластичність (Elasticity) – дозволяє динамічно збільшувати або зменшувати використання ресурсів відповідно до зміни запитів користувачів, забезпечуючи оптимізацію витрат та продуктивність.

Самообслуговування за запитом (On-demand self-service) – користувачі можуть самостійно керувати ресурсами без безпосередньої взаємодії з постачальником послуги.

Широкий доступ до мережі (Broad network access) – послуги доступні через стандартні механізми, що використовуються на всіх платформах (наприклад, мобільні, настільні чи інші пристрої).

Для глибшого розуміння специфіки хмарних ІС, розглянемо детально типові архітектури основних моделей надання послуг. Ці моделі є структурною основою для розгортання та управління різноманітними застосунками та сервісами у хмарному середовищі.

На рисунку 1.1 наведено типову архітектуру IaaS, яка є основою для надання інфраструктурних ресурсів у вигляді сервісу.

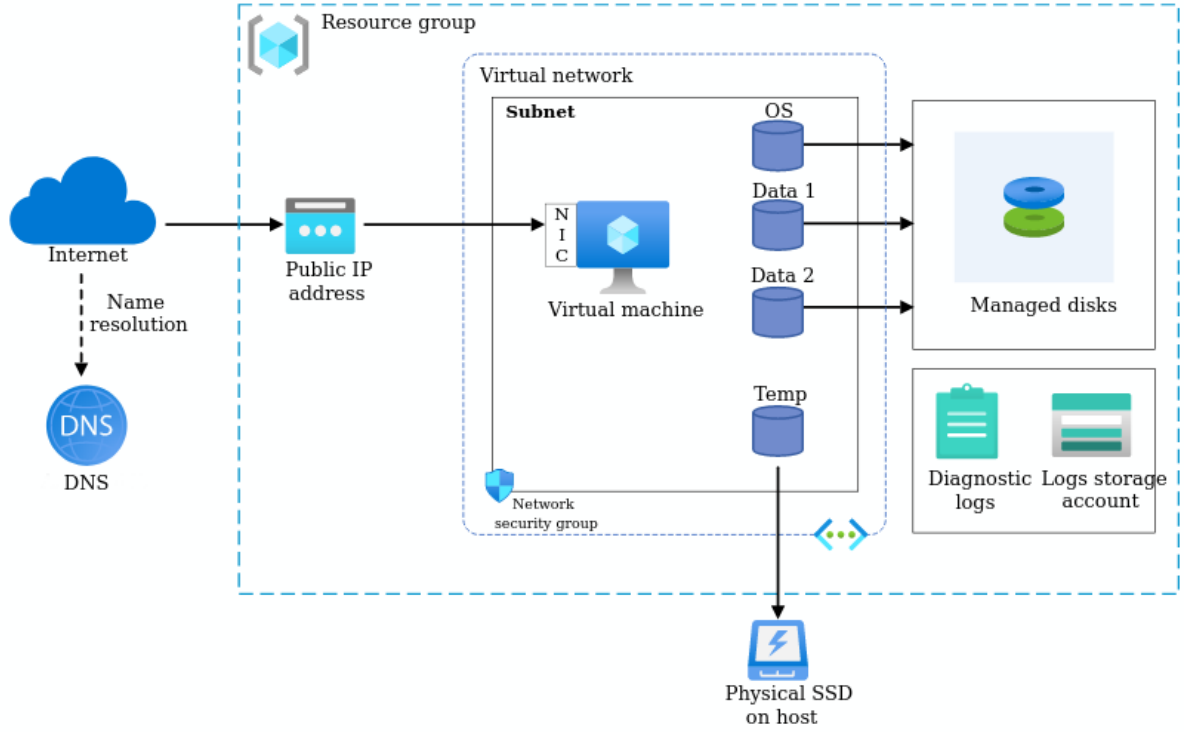

Рисунок 1.1 – Типова архітектура IaaS

Основними компонентами такої системи є Virtual Machines, Network Services, Storage Services, Cloud API, які разом формують надійну основу для запуску та масштабування додатків.

Virtual Machines (VM) – ізольовані серверні сутності, які імітують фізичні комп'ютери та надають користувачам обчислювальні ресурси.

Network Services – сервіси для управління віртуальними мережами, балансуванням навантаження, забезпечення безпеки та доступності мережі.

Storage Services – це спеціалізовані сервіси для зберігання даних, що забезпечують високу надійність та доступність інформації. Вони пропонують різні опції конфігурації в залежності від потреби у швидкості доступу, частоти використання та обсягу зберігання, дозволяючи оптимізувати вартість та ефективність в контексті конкретних бізнес-завдань.

Cloud API – набір протоколів та інструментів, які надають розробникам можливість здійснювати взаємодію з хмарною інфраструктурою. Це дозволяє автоматизувати процеси розгортання, моніторингу, управління та масштабування ресурсів, спрощуючи інтеграцію різноманітних додатків і сервісів у хмарному оточенні.

Типову структуру PaaS архітектури, котра надає розробникам комплексні середовища та інструменти для проектування, тестування та розгортання додатків наведено на рисунку 1.2.

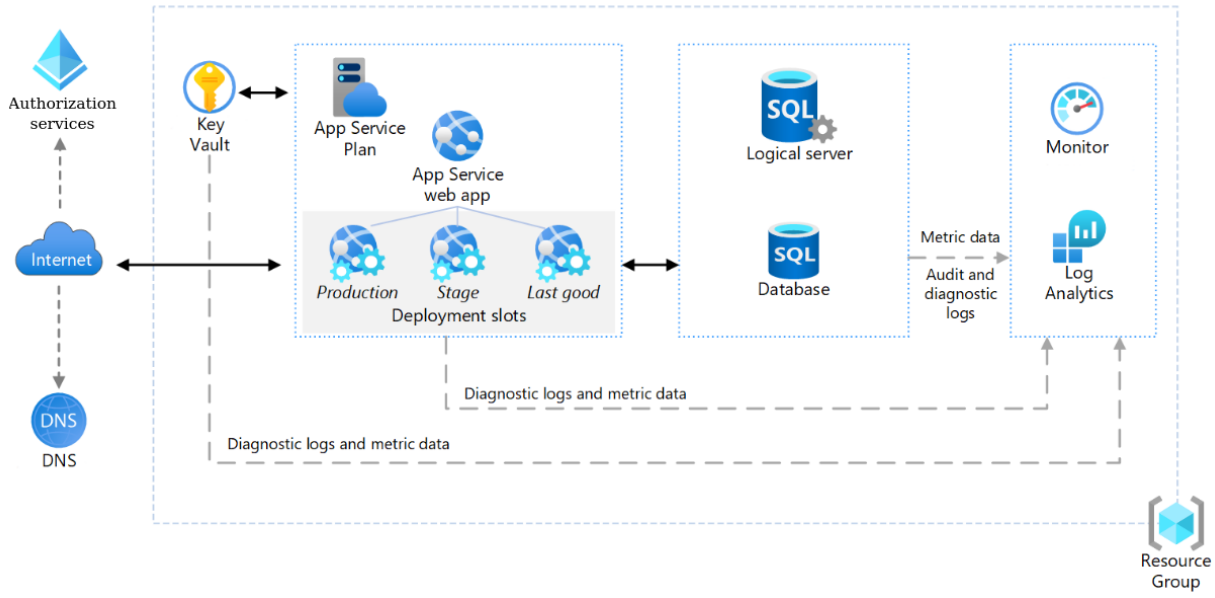

Рисунок 1.2 – Типова архітектура PaaS

Ключовими компонентами такої системи є App Service, DNS, Database, Authorization services, Monitor, Key Vault.

App Service – це повністю керована платформа для створення та розгортання хмарних додатків. Дозволяє визначати набір обчислювальних ресурсів для запуску веб-додатка, розгортати веб-додатки та налаштовувати середовище розгортання.

DNS – хмарний сервіс служби DNS, який забезпечує розрішення імен.

Database – база даних (БД) як сервіс у хмарі.

Authorization services – хмарний сервіс ідентифікації та доступу, який дозволяє користувачам отримувати доступ до хмарної ІС.

Monitor – рішення для збору, аналізу та дій на основі журналів та метрик.

Key Vault – служба для управління секретами, ключами та сертифікатами.

На рисунку 1.3 представлено еталонну архітектуру SaaS у хмарі Microsoft Azure, що пропонує кінцевим користувачам комплексні веб-додатки доступні онлайн. У цій моделі, всі технічні аспекти, включно з управлінням інфраструктурою та платформою, покладаються на провайдера сервісу, що дозволяє користувачам зосередитись на основних бізнес-завданнях без зайвих турбот про технічне обслуговування. Ключовими компонентами системи є Azure Front Door для оптимізації маршрутизації трафіку і забезпечення високої доступності, Microsoft Entra Identity Services для ідентифікації користувачів та управління доступом, Azure DNS для обслуговування доменних імен, Application Gateway для розподілення навантаження внутрішнього трафіку, App Service для ефективного розгортання веб-додатків, Azure Kubernetes Service (AKS) для оркестрування контейнерів, Azure SQL Elastic Pools для гнучкого управління ресурсами баз даних, Azure Cognitive Search для розширених пошукових можливостей із використанням AI та Azure Cache for Redis, що забезпечує високопродуктивний кеш в пам'яті.

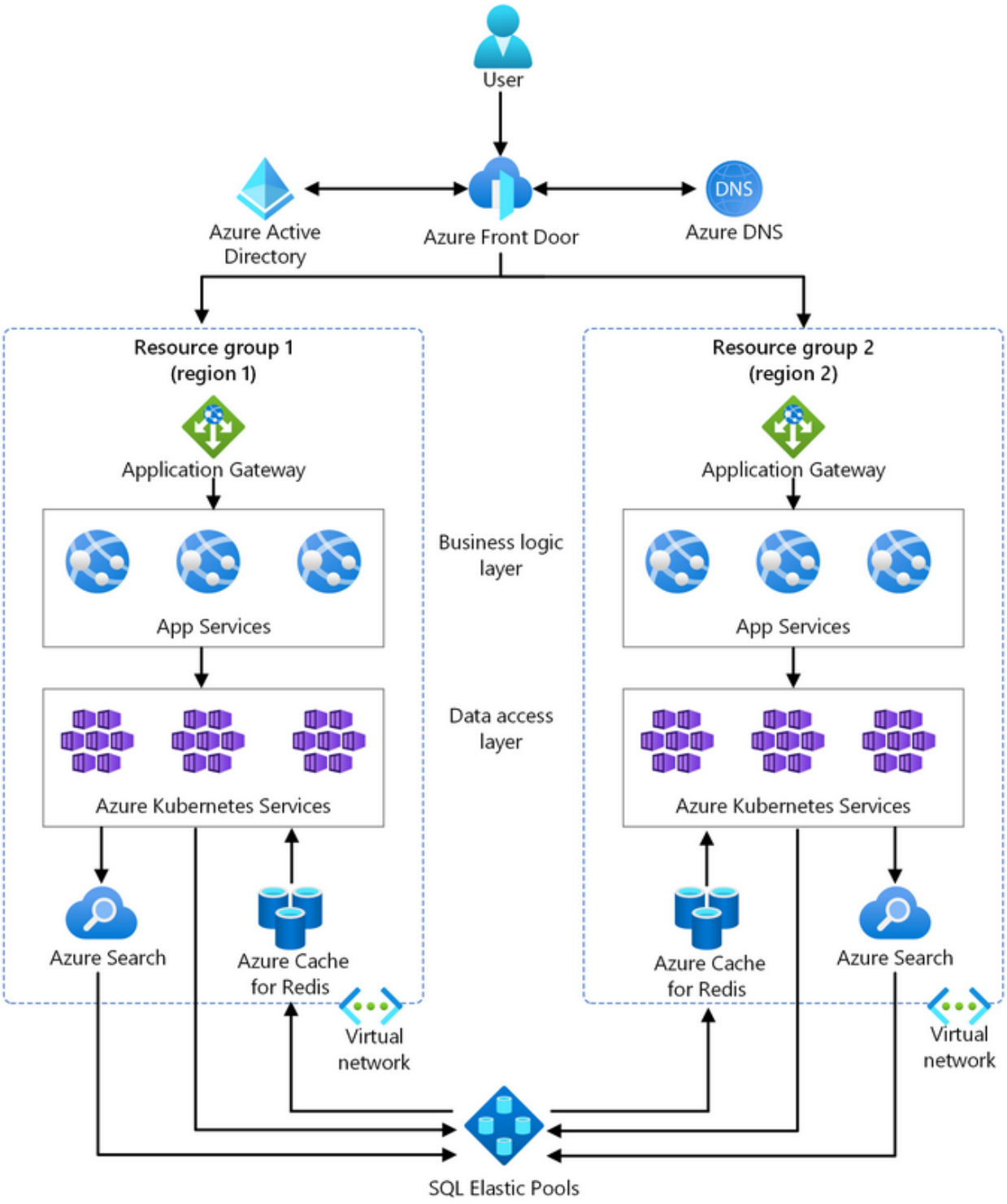

Рисунок 1.3 – Типова архітектура SaaS

Після детального аналізу моделей IaaS, PaaS та SaaS, можна зробити висновок, що хмарні технології загалом значно підвищують ефективність та гнучкість розгортання і управління програмним забезпеченням. Кожна модель має свої переваги залежно від потреб бізнесу. Проте для отримання максимальної віддачі від хмарних рішень, ключовим є впровадження сучасних підходів в розробці ПЗ, зокрема концепції CI\CD. Адже саме CI\CD дозволяє автоматизувати та прискорити розробку, тестування і доставку оновлень для хмарних ІС. Тому, доцільним буде детально проаналізувати сутність концепції CI\CD, її ключові принципи, переваги та особливості впровадження, що дасть змогу сформувати ґрунтовне розуміння методології CI\CD та підходів до її ефективної реалізації у хмарних ІС.

1.3 Аналіз структури та огляд існуючих рішень автоматизації CI\CD процесів для хмарних інформаційних систем

Аналізуючи більшість хмарних інформаційних систем, які розробляються станом на сьогодні, можна відзначити, що велика частина з них базується на розподіленій мікросервісній архітектурі. Такий підхід до структурування системи дозволяє забезпечити гнучкість управління, масштабованість та високу доступність сервісів. Ключовою перевагою мікросервісної архітектури є її здатність до швидкої адаптації під змінні умови бізнесу та технологій, оскільки кожен мікросервіс є незалежним та може бути розроблений, розгорнутий, масштабований та оновлений окремо від інших [5].

Паралельно з розвитком мікросервісної архітектури активно набирає популярності технологія контейнеризації. Контейнери є ідеальним рішенням для мікросервісних систем, оскільки вони забезпечують легкість та ефективність у створенні, розгортанні та управлінні мікросервісами. Завдяки

ізоляції залежностей та оточення, контейнери спрощують процеси CI\CD, роблячи їх більш швидкими та надійними [6].

На рисунку 1.4 представлено структуру типової хмарної інформаційної системи з мікросервісною архітектурою, де основні компоненти системи розміщені у контейнерах для забезпечення ізоляції, ефективного масштабування та оптимального використання ресурсів. Процеси CI\CD, відображені на цьому рисунку, інтегровані з основними модулями системи і виконують критично важливі функції автоматизації інтеграції, тестування, доставки та розгортання мікросервісів. Ця автоматизація відіграє ключову роль у забезпеченні безперервного життєвого циклу розробки, дозволяючи командам розробників здійснювати часті та надійні оновлення продукту без значного простою системи або втручання з боку IT-спеціалістів.

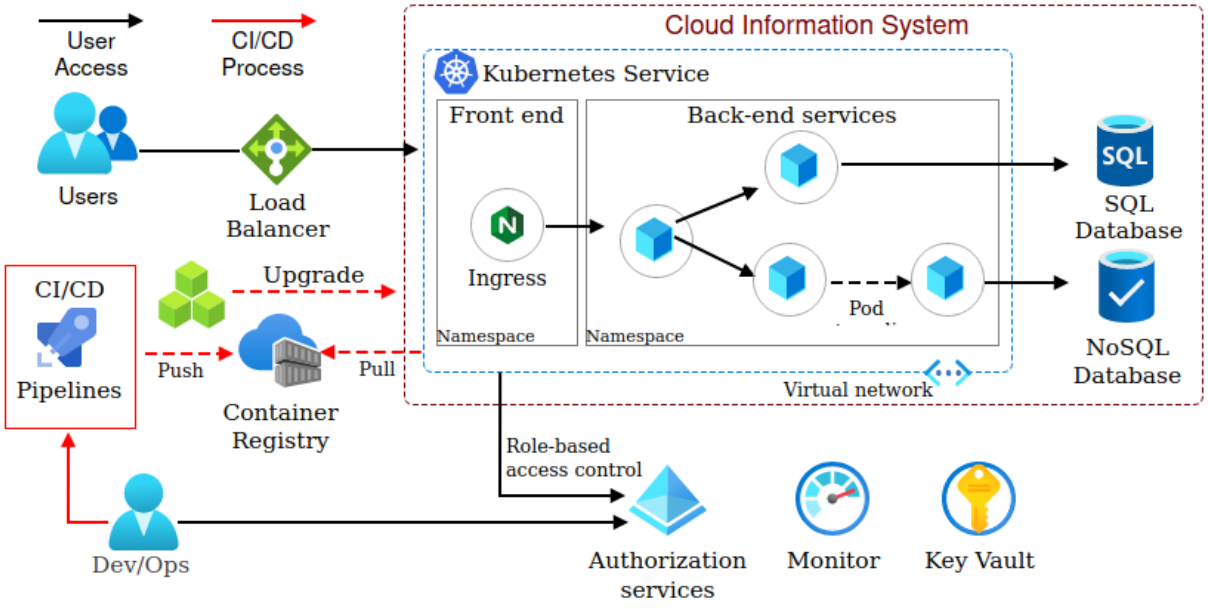

Рисунок 1.4 – Структура типової хмарної інформаційної системи з мікросервісною архітектурою та автоматизованими CI\CD процесами

Для глибшого розуміння ролі та впливу CI\CD процесів на ефективність хмарної інформаційної системи, доцільно детально проаналізувати їх внутрішню структуру. Такий аналіз дозволить з'ясувати, як саме CI\CD інтегруються з основними компонентами системи, та виявити

ключові аспекти, що впливають на швидкість, надійність і стабільність процесів розгортання мікросервісів у хмарному середовищі.

CI\CD конвеєр можна умовно поділити на низку послідовних етапів, починаючи з планування і закінчуючи експлуатацією готової системи. Ці етапи включають: планування, розробку, збірку, тестування, підготовку релізу, розгортання та експлуатацію [7]. Деталізована структура та короткий опис дій для кожного етапу CI\CD процесів наведено на рисунку 1.5.

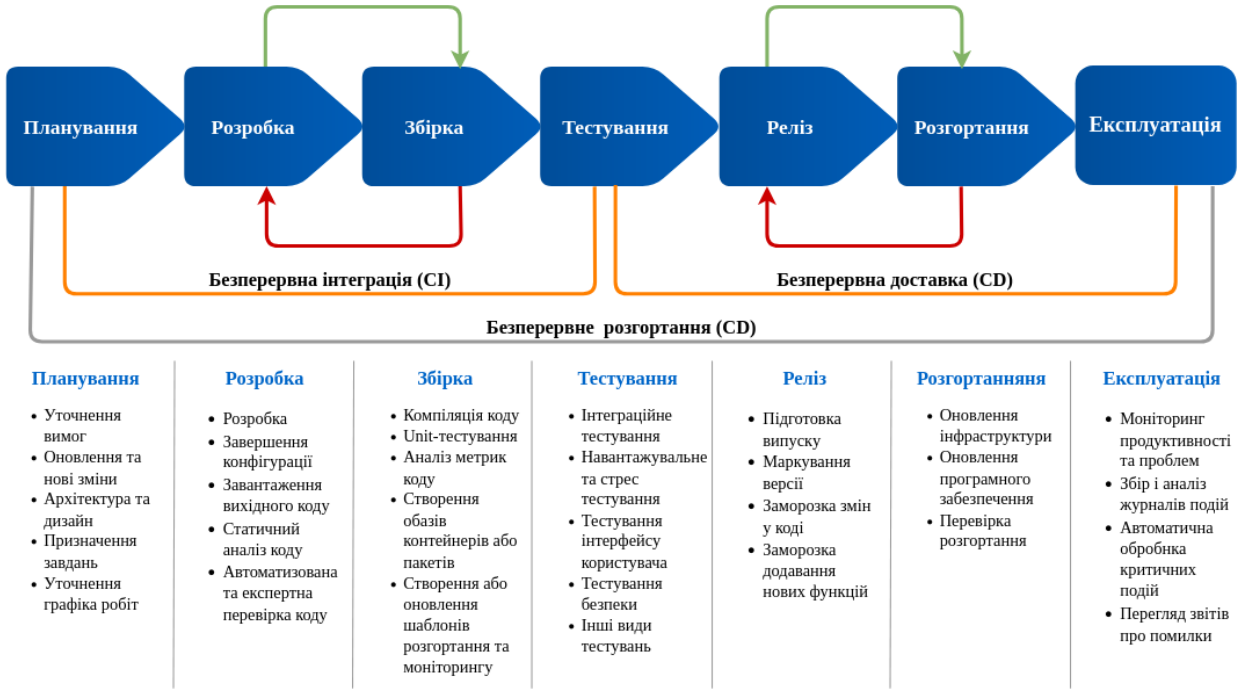

Рисунок 1.5 – Детальна структура CI\CD процесів

Проаналізуємо детальніше зміст та особливості кожного з цих етапів.

Планування – на цьому етапі відбувається уточнення вимог до системи, внесення необхідних змін та додавання нового функціоналу, проектування архітектури та дизайну системи з урахуванням особливостей хмарної інфраструктури, розподіл завдань між учасниками команди, складання графіку робіт з урахуванням процесів CI\CD.

Розробка – на цьому етапі команда виконує безпосередню розробку функціональності системи, завершує її конфігурування, завантажує вихідний код у систему керування версіями, виконує статичний аналіз коду, автоматизовану та експертну перевірку коду.

Збірка – тут відбувається компіляція коду, запуск Unit-тестів, аналіз метрик коду, створення образів контейнерів або дистрибутивних пакетів, генерування чи оновлення шаблонів для автоматизації розгортання та моніторингу в хмарі.

Тестування – на цьому етапі запускаються різні види тестів: інтеграційні, навантажувальні, функціональні, безпеки тощо. Тестування дозволяє пройти верифікацію працездатності системи до її доставки у хмару.

Реліз – тут готується фінальний реліз, "заморожується" код та функціонал, відбувається присвоєння версії релізу та остаточна перевірка готовності до доставки та розгортання.

Розгортання – на цьому етапі виконується оновлення хмарної інфраструктури, розгортання оновленої версії програмного забезпечення, перевірка успішності розгортання у хмарі.

Експлуатація – після розгортання починається моніторинг роботи системи, збір журналів подій, аналіз продуктивності та проблем, автоматичне реагування на критичні події, перегляд звітів про помилки.

Таким чином, CI\CD охоплює весь життєвий цикл хмарної інформаційної системи – від планування до експлуатації.

Аналіз структури CI\CD процесів дозволяє виділити ключові компоненти та етапи, з яких складається конвеєр безперервної інтеграції та безперервної доставки. Щоб детальніше зрозуміти принцип роботи CI\CD, необхідно проаналізувати вхідні та вихідні дані для цих процесів, передумови для ефективної роботи CI\CD конвеєру та обмеження. Результати аналізу вхідних та вихідних даних, передумов та обмежень наведено в таблиці 1.1.

Таблиця 1.1– Результати аналізу вхідних та вихідних даних, передумов та обмежень для CI\CD процесів

Категорія	Опис
Передумови	– Налаштоване середовище розробки, тестів, та експлуатації; – Інструменти автоматизації збірки, тестування та розгортання; – Взаємодія між командами розробки, тестування, DevOps; – Система керування версіями та артефактами; – Правила безпеки та конфіденційності даних.
Вхідні дані	– Вихідний код від розробників; – Конфігураційні файли системи та програм; – Скрипти для баз даних та інсталяції; – Технічна документація та вимоги; – Дані для тестування функціоналу; – Метрики якості попередньої версії.
Вихідні дані	– Зібрані виконувані пакети або контейнери з ПЗ; – Звіти про результати тестування; – Логи процесів та артефакти розгортання; – Оновлена версія ПЗ в хмарі; – Метрики якості нової версії.
Обмеження	– Обмеження хмарної інфраструктури; – Технічні залежності в коді; – Правові обмеження щодо ліцензування ПЗ; – Ресурсні обмеження для CI\CD інфраструктури; – Політики безпеки хмарного провайдера; – Обмеження продуктивності інструментів автоматизації.

Після дослідження вхідних та вихідних даних, передумов та обмежень для CI\CD процесів, доцільно здійснити аналіз наявних методів їх автоматизації. Існуючі на даний час в компаніях підходи до автоматизації CI\CD можна умовно класифікувати наступним чином:

– ручне розгортання та оновлення компонентів ІС, що є трудомістким процесом з високою ймовірністю помилок, браком стандартизації процесів, неузгодженістю конфігурацій, відсутністю аудиту та відстеження змін;

– використання скриптів для часткової автоматизації дозволяє частково автоматизувати виконання окремих рутинних операцій, але вимагає суттєвих ресурсів на подальше супроводження та підтримку самих скриптів;

– використання окремих рішень для автоматизації частини процесу дозволяє частково автоматизувати CI\CD процеси, але призводить до фрагментації цих процесів між різними автоматизованими системами;

– використання складних та ресурсоємних систем автоматизації забезпечує широкі можливості комплексної автоматизації, але потребує значних витрат на впровадження, підтримку та масштабування;

– використання рішень орієнтованих на монолітні системи спрощує автоматизацію в рамках однієї великої системи, але ускладнює її для окремих компонентів, що робить такі підходи непридатними для систем з використанням мікросервісної архітектури.

Такі обмежені підходи не дозволяють повною мірою використати переваги концепції CI\CD, особливо в умовах використання хмарних технологій та гнучкої методології розробки. Зокрема, для сучасної мікросервісної архітектури більшості хмарних ІС потрібні ефективніші методи автоматизації CI\CD з урахуванням їх розподіленої архітектури та динамічності. Обмежені традиційні підходи не забезпечують гнучкого та швидкого процесу безперервної інтеграції і доставки для окремих мікросервісів. Тому, актуальною є розробка нових комплексних рішень автоматизації CI\CD процесів на основі сучасних технологій, таких як контейнеризація та оркестрація, для ефективної підтримки мікросервісної архітектури хмарних ІС. Аналіз інструментальних засобів, що можуть підтримати такі методи, є ключовим для вирішення задачі комплексної автоматизації цих процесів.

1.4 Аналіз інструментальних засобів автоматизації CI\CD процесів для хмарних інформаційних систем

1.4.1 Аналіз інструментальних засобів для контролю версій програмного коду

Серед інструментальних засобів контролю версій програмного коду найбільшого поширення набули системи Git, Subversion (SVN), Mercurial.

Git – розподілена система керування версіями, що стала фактичним стандартом для IT-проектів [8]. Відрізняється високою швидкістю та ефективністю. Має велику екосистему сервісів та інтеграцій.

SVN – поширена централізована система контролю версій. Простіша за Git, але має обмежені можливості розподіленої розробки.

Mercurial – розподілена система керування версіями, альтернатива Git. Використовується в деяких відкритих проектах, але поступається Git за популярністю.

1.4.2 Аналіз інструментальних засобів для хостингу програмного коду

Для організації хостингу та спільної розробки коду широко використовуються хмарні платформи, такі як GitHub, GitLab, Bitbucket, Azure DevOps, AWS CodeCommit, Google Cloud Source Repositories.

GitHub – найпопулярніший та функціональний веб-сервіс для хостингу Git репозиторіїв [9]. Має зручний інтерфейс, широкі можливості для управління проектами та кодом, розвинену екосистему інтеграцій. Основним недоліком є платні тарифи для великих команд, але широкий спектр можливостей та популярність сервісу часто перевищують цю обмеженість.

GitLab – хмарна платформа для хостингу репозиторіїв Git, яка пропонує схожі з GitHub функціональні можливості, але з менш зручним інтерфейсом та менш розвиненою екосистемою інтеграцій. Її перевага полягає у можливості безкоштовного використання, що робить її доступною для широкого кола користувачів.

Bitbucket – веб-сервіс хостингу репозиторіїв Git від компанії Atlassian. Має обмежені можливості порівняно з GitHub та GitLab. Має деякі обмеження у функціональності порівняно з GitHub та GitLab, але пропонує безкоштовний план для невеликих команд, що робить його привабливим для стартапів та малих проектів.

Azure DevOps – хмарний сервіс від компанії Microsoft. Має тісну інтеграцію з екосистемою Azure, що є також і його обмеженням.

AWS CodeCommit – хмарна платформа від компанії Amazon для хостингу Git репозиторіїв. Інтегрована зі службами AWS. Обмеження – прив'язка до хмарної платформи AWS.

Google Cloud Source Repositories – хмарний сервіс від компанії Google, який пропонує безпечний хостинг Git репозиторіїв та інтеграцію з іншими сервісами Google. Обмеження – прив'язка до хмарного середовища Google Cloud.

1.4.3 Аналіз інструментальних засобів автоматизації безперервної інтеграції для хмарних інформаційних систем

Найбільш розповсюдженими інструментальними засобами автоматизації безперервної інтеграції для ХІС є Jenkins, GitHub Actions, GitLabCI, CircleCI, TravisCI, Bamboo, AWS CodePipeline, Azure Pipelines.

Jenkins – зрілий CI інструмент, який забезпечує широкі можливості гнучкого налаштування автоматизації процесів завдяки великій екосистемі

плагінів. Проте складність налаштування через надмірну гнучкість та необхідність написання скриптів на складній мові може бути недоліком для невеликих проектів, які потребують простішого підходу.

GitLab CI – зручний вбудований функціонал для автоматизації в рамках екосистеми GitLab, але має обмежену інтеграцію з зовнішніми інструментами, не пов'язаними з GitLab, що може бути недоліком при використанні інших платформ розробки. Крім того, безкоштовні тарифні плани GitLab передбачають певні обмеження на обчислювальні ресурси та кількість паралельних задач, що необхідно враховувати.

Circle CI – вирізняється швидкодією та зручністю налаштування CI\CD конвеєрів. Проте вартість може бути високою для проектів зі значними обсягами зберігання даних чи паралельних задач.

Travis CI – простий у використанні та інтегрується з GitHub для автоматичного виявлення змін коду, але обмеження безкоштовної версії можуть вплинути на доступність для деяких проектів.

GitHub Actions – має тісну інтеграцію з GitHub та гнучкість налаштування автоматизації прямо в коді репозиторію [10]. Має обмеження на безкоштовні ресурси, але ці обмеження для більшості проектів не є критичними. Підтримує багато різноманітних мов та фреймворків.

AWS CodePipeline – хмарний сервіс від Amazon, тісно інтегрований з іншими сервісами AWS. Зручний для проектів, що розгортаються на AWS. Але має сильну прив'язку до екосистеми AWS.

Azure Pipelines – хмарний сервіс від Microsoft. Добре інтегрується з іншими сервісами Azure. Зручний для проектів на Azure, але обмежений цією хмарною платформою.

1.4.4 Аналіз інструментальних засобів автоматизації збірки та тестування програмного забезпечення для хмарних інформаційних систем

Основними засобами автоматизації збірки та тестування програмного забезпечення виступають Bazel, Maven, Gradle, CMake, MSBuild, Selenium, Appium, Katalon Studio, Azure Test Plans та Google Cloud Test Lab.

Bazel – інструмент від компанії Goggle, який забезпечує швидку, ефективну та масштабовану збірку проектів, оптимізуючи процеси через поетапну збірку та кешування. Підтримує численні мови та платформи, пропонуючи узгодженість і відтворюваність результатів.

Maven – популярний інструмент автоматизації збірки у Java-середовищах, але менш гнучкий для інших мов і залежний від централізованого репозиторію.

Gradle – гнучке рішення автоматизації збірки, але може мати складну конфігурацію та невисоку продуктивність у великих проектах.

CMake – інструмент автоматизації збірки, широко використовується для C/C++ проектів, але його синтаксис може бути складним для розуміння.

MSBuild – рішення автоматизації збірки для .NET проектів, але обмежений підтримкою інших платформ та мов.

Selenium – популярний фреймворк з відкритим вихідним кодом для автоматизації тестування. Підтримує різні мови програмування та має гнучкі можливості налаштування. Потребує технічних навичок для використання.

Appium – рішення відкритим вихідним кодом для автоматизованого тестування мобільних додатків. Дозволяє писати тести на різних мовах програмування. Складність полягає у налаштуванні тестового середовища.

Katalon Studio – комерційний інструмент для автоматизації тестів з графічним інтерфейсом. Простіший у використанні порівняно з Selenium та Appium. Має безкоштовну версію з обмеженим функціоналом.

Azure Test Plans – рішення від Microsoft для автоматизації тестування та управління тестами. Інтегрується з Azure DevOps. Обмеження – прив'язка до платформи Azure.

Google Cloud Test Lab – хмарний сервіс від компанії Google для автоматизованого тестування мобільних додатків та веб-сайтів. Має тісна інтеграція з хмарою Google.

1.4.5 Аналіз інструментальних засобів автоматизації безперервної доставки та розгортання програмного забезпечення для хмарних інформаційних систем

До поширених інструментів автоматизації безперервної доставки та розгортання програмного забезпечення для хмарних інформаційних систем належать Docker, ArgoCD, FluxCD, Spinnaker, Docker Swarm, Nomad, Kubernetes. Їх популярність обумовлена здатністю ефективно керувати складними процесами в середовищі, де використовуються мікросервіси та контейнеризація. Вони забезпечують важливі функції, такі як управління контейнерами, автоматизація розгортання, моніторинг та масштабування, що є ключовими для підтримки надійності, доступності та продуктивності хмарних ІС. Оскільки хмарні інформаційні системи часто залежать від швидкої та гнучкої реакції на зміни в навантаженні та вимогах, ці інструменти надають необхідні можливості для автоматизації цих процесів.

Docker – відкрита платформа для розробки, доставки та запуску додатків, що дозволяє відокремити додатки від інфраструктури та прискорити поставку програмного забезпечення. Завдяки технології контейнеризації Docker надає стандартизований спосіб упаковки та розгортання ПЗ у вигляді легких та переносних контейнерів, які містять всі необхідні бібліотеки та залежності. Дозволяє уніфікувати процеси тестування, відправки коду у

виробництво та розгортання між різними середовищами – від локального для розробників до хмарних платформ.

ArgoCD – декларативний інструмент автоматизації розгортання програмного забезпечення для інформаційних систем у контейнеризованих середовищах [11]. Вирізняється можливістю контролювати та керувати станом додатків, відповідно до конфігурацій, збережених у Git. Забезпечує візуалізацію процесів розгортання та підтримує автоматичну синхронізацію, що робить його особливо корисним для забезпечення консистентності в розгортаннях.

FluxCD – рішення для розгортання та управління додатками на базі Kubernetes. Як і ArgoCD, працює шляхом моніторингу репозиторію з вихідним кодом на предмет змін та автоматичної генерації розгортань для підтримки актуальності додатків. Керує розгортаннями, використовуючи концепцію GitOps, яка передбачає, що бажаний стан додатку визначено у вихідному коді та контролюється за допомогою версій у Git. Такий підхід спрощує відстеження змін у розгортаннях та співпрацю з іншими членами команди.

Spinnaker – відкрита платформа безперервної доставки, яка допомагає впроваджувати зміни в програмне забезпечення швидко та з високим рівнем надійності. Був розроблений з метою оптимізації процесів розгортання в хмарних середовищах, дозволяючи командам розробників ефективно керувати релізами через різні хмарні платформи, такі як AWS, GCP, Azure, а також в приватних хмарах. Підтримує різноманітні стратегії розгортання забезпечуючи гнучкість та надійність при випуску нових версій.

Docker Swarm – потужний інструмент для оркестрування контейнерів, який дозволяє розробникам та системним адміністраторам легко та ефективно створювати та керувати кластерами контейнерів [12]. За допомогою Docker Swarm можна автоматично масштабувати додатки та збалансувати навантаження між різними вузлами кластера. Однак, у порівнянні з Kubernetes та Nomad, Docker Swarm має деякі обмеження. По-

перше, Docker Swarm має менш розвинену екосистему та спільноту, особливо у порівнянні з Kubernetes, який є лідером на ринку оркестрування контейнерів. По-друге, хоча Docker Swarm простіший у налаштуванні та використанні, він не надає такого рівня гнучкості та розширюваності, як Kubernetes, що може бути важливим для складних та багатофункціональних розгортань. Також Docker Swarm має менш розширені можливості управління мережею та безпекою, що є ключовими аспектами для великих та розподілених систем. Архітектуру Docker Swarm наведено на рисунку 1.6.

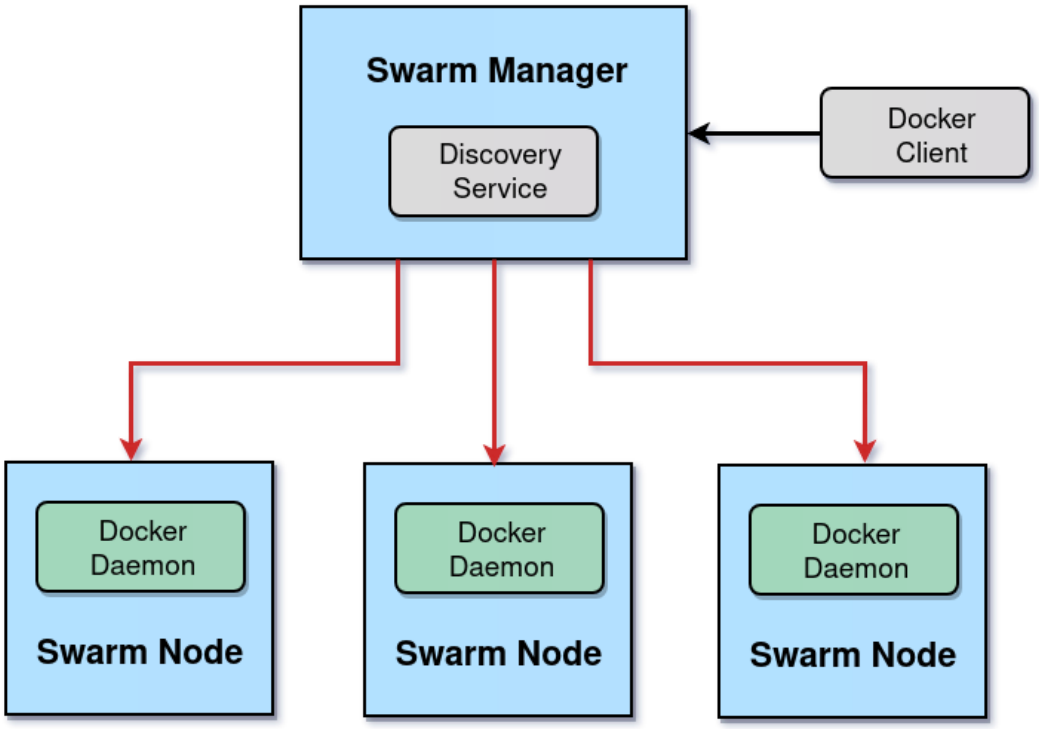

Рисунок 1.6 – Архітектура Docker Swarm

Docker Swarm складається з вузлів (nodes), кожен з яких містить Docker демон та всі вони взаємодіють з Docker API через HTTP. Є два типи вузлів: менеджери (Swarm Manager) та звичайні вузли (Swarm Nodes). Менеджери відповідають за оркестрування та управління кластером, тобто розподіляють задачі між вузлами та контролюють їх виконання. Звичайні вузли отримують задачі від менеджерів, запускають контейнери та звітують про стан їх виконання.

Nomad – інструмент для розгортання додатків від компанії HashiCorp. Підтримує різні типи завдань, включаючи контейнери та не-контейнерні рішення, і вирізняється своєю простотою використання та налаштування, що робить його доступним для різних середовищ [13]. Архітектура Nomad, яку наведено на рисунку 1.7, складається з серверів та клієнтів. Сервери керують станом кластера, розподіляють завдання та координують розгортання, використовуючи алгоритм консенсусу Raft для забезпечення надійності. Клієнти Nomad виконують завдання, призначені серверами, і звітують про їх стан. Nomad підтримує гнучкі стратегії розгортання, забезпечує високу доступність і масштабованість, а також інтегрується з іншими продуктами HashiCorp, такими як Consul для виявлення сервісів та Vault для управління секретами, що робить його ефективним рішенням для різноманітних вимог розгортання. Однак, порівняно з Kubernetes, Nomad має менш розвинену екосистему та спільноту, а також обмежену функціональність у плані складних розгортань і вимог до високої доступності.

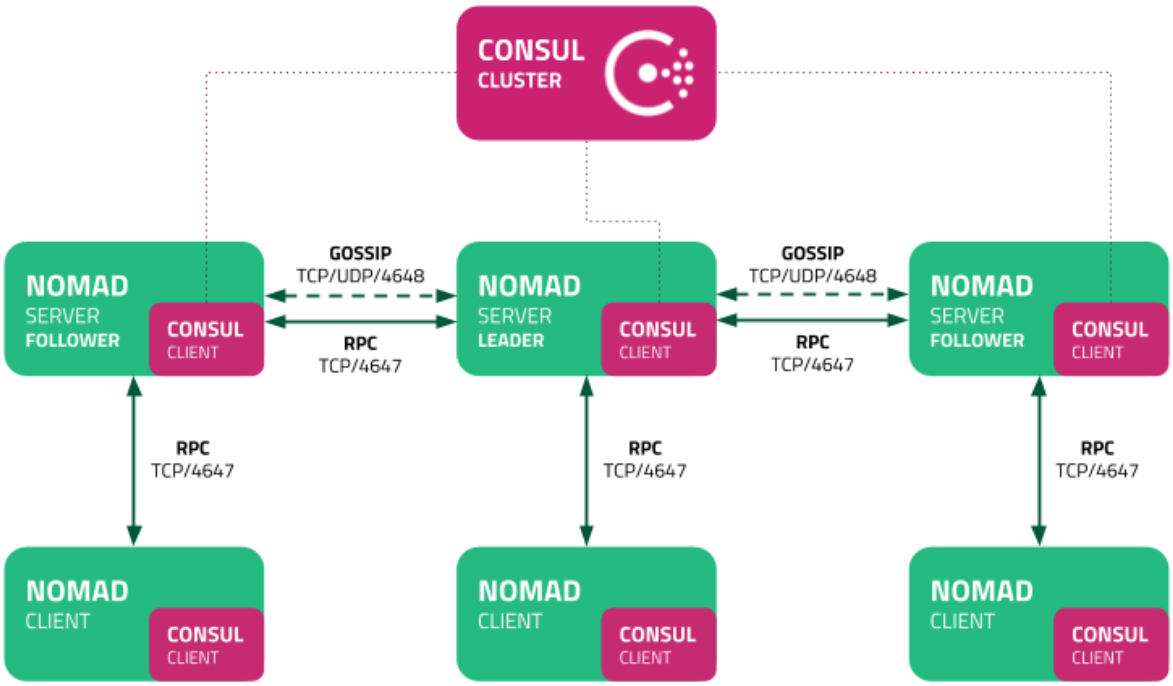

Рисунок 1.7 – Архітектура Nomad

Kubernetes – провідний інструмент для оркестрування контейнерів, який надає надзвичайну гнучкість та масштабованість для розгортання та управління додатками. Безкоштовне, відкрите програмне забезпечення, розроблене Google, яке стало стандартом де-факто для сучасних хмарних додатків і мікросервісної архітектури [14]. Ключовою перевагою Kubernetes є його здатність до самовідновлення, автоматичного масштабування, а також можливість запускати контейнери в різних середовищах, від локальних серверів до великих хмарних провайдерів. Архітектуру Kubernetes наведено на рисунку 1.8

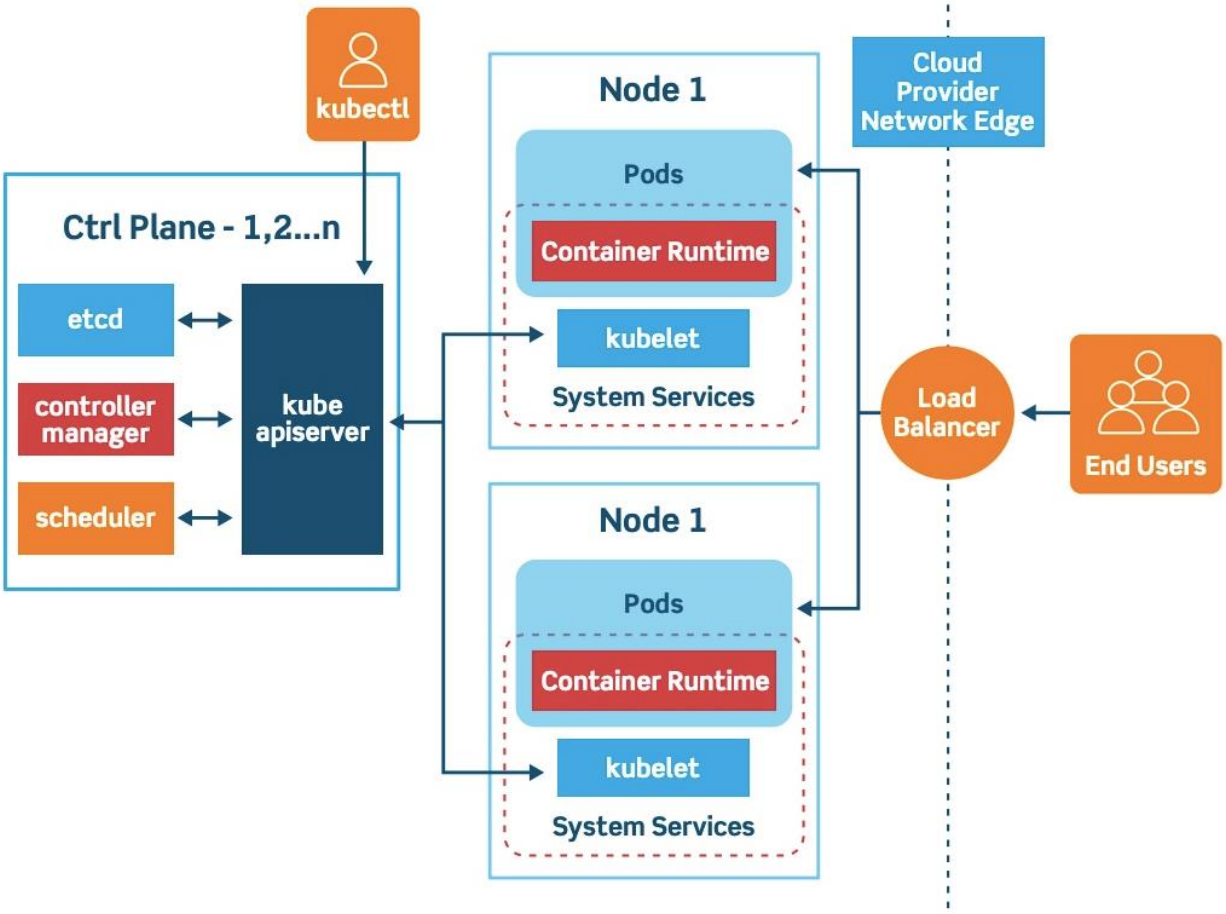

Рисунок 1.8 – Архітектура Kubernetes

Архітектура Kubernetes організована навколо кластера, що складається з контрольних вузлів (control plane nodes) та робочих вузлів (worker nodes). Контрольні вузли відповідають за управління станом кластера, планування запуску додатків, а також за взаємодію між різними компонентами. Вони включають в себе API сервер, планувальник (scheduler), контролери та сховище (etcd) для зберігання стану кластера. Робочі вузли запускають контейнери, в яких працюють додатки, за допомогою Kubelet, який керує життєвим циклом контейнерів. Kubernetes також включає сервіси для забезпечення мережевого зв'язку між контейнерами, включно з внутрішньою DNS-системою, маршрутизацією трафіку та балансуванням навантаження.

1.4.6 Аналіз інструментальних засобів моніторингу хмарних інформаційних систем

До поширених інструментів моніторингу хмарних інформаційних систем належать: Prometheus, Grafana, Elasticsearch, Datadog, Zabbix та New Relic.

Prometheus – відкритий інструмент моніторингу, що набув широкого поширення для спостереження за хмарними системами та інфраструктурою. Вирізняється потужними можливостями збору метрик, ефективною моделлю зберігання та візуалізації даних [15]. Легко масштабується на великі обсяги метрик та підтримує складні запити для аналізу даних.

Grafana – популярний інструмент для візуалізації даних моніторингу. Легко інтегрується з різними джерелами даних і дозволяє створювати наочні інформаційні панелі. Може бути складною в налаштуванні через велику кількість опцій.

Elasticsearch – розподілена система пошуку та аналізу, що часто застосовується для збору та моніторингу логів великих хмарних систем. В поєднанні з Kibana надає потужні можливості аналізу логів.

Datadog – хмарний сервіс моніторингу, який пропонує розширені функції для моніторингу застосунків, мережі та інфраструктури. Datadog включає в себе інтуїтивно зрозумілі інформаційні панелі та має потужні інструменти сповіщень, але його використання може бути відносно дорогим, особливо для великих об'ємів даних.

Zabbix – відкритий інструмент моніторингу, який є універсальним рішенням для відстеження стану мережі, серверів і застосунків. Zabbix підтримує багатий набір типів даних для моніторингу і гнучкі налаштування сповіщень, але його інтерфейс і конфігурація є досить складними для опанування.

New Relic – хмарний сервіс моніторингу, який пропонує детальний аналіз продуктивності застосунків та користувацького досвіду. New Relic підходить для моніторингу складних хмарних застосунків, але може бути дорогим для невеликих організацій або проектів з обмеженим бюджетом.

1.4.7 Аналіз інструментальних засобів для написання інфраструктури у вигляді коду для хмарних інформаційних систем

Найпоширенішими інструментальними засобами для написання інфраструктури у вигляді коду для хмарних інформаційних систем є Terraform, CloudFormation, Pulumi, Crossplane.

Terraform – потужний інструмент для написання, планування та створення інфраструктури як коду (IaC) від компанії HashiCorp [16]. Його сильною стороною є здатність керувати широким спектром ресурсів у багатьох хмарних провайдерах використовуючи декларативну мову для опису

інфраструктури. Terraform підтримує модульність та повторне використання коду. Одне з кращих рішень для опису інфраструктури у вигляді коду.

CloudFormation – інструмент від компанії Amazon для опису та управління інфраструктурою AWS у вигляді коду. CloudFormation дозволяє користувачам моделювати повний набір інфраструктури AWS і управляти нею. Хоча він є потужним та ефективним для користувачів AWS, його обмеження в тому, що він працює лише в екосистемі AWS.

Pulumi – сучасний інструмент IaC, який дозволяє користувачам використовувати загальні мови програмування, такі як JavaScript, TypeScript, Python та Go для опису інфраструктури, що і є його головною перевагою, оскільки розробникам простіше використовувати вже знайомі їм мови програмування для опису інфраструктури. Однак, Pulumi ще відносно новий на ринку, тому його спільнота та екосистема не такі розвинені, як у Terraform.

Crossplane – відкритий інструмент, який дозволяє керувати інфраструктурою у вигляді коду за допомогою Kubernetes API. Це робить Crossplane унікальним, оскільки він інтегрує IaC з Kubernetes, дозволяючи використовувати потужні можливості оркестрування Kubernetes для управління інфраструктурою через різні хмарні провайдери. Головною перевагою Crossplane є його здатність працювати як єдина система управління для багатьох хмарних сервісів та ресурсів, забезпечуючи високу гнучкість та зручність. Однак, через свою залежність від Kubernetes, він може виявитися складним у використанні.

1.4.8 Аналіз інструментальних засобів хмарних систем

Важливим аспектом вибору технологічних рішень для хмарних інформаційних систем є оцінка інструментальних засобів, які пропонують провідні хмарні провайдери. Кожен з них надає унікальний набір сервісів та

інструментів, що істотно впливають на архітектуру та ефективність хмарних рішень. Розглянемо ключові характеристики та можливості, які відрізняють основних гравців на ринку хмарних технологій.

Amazon Web Services (AWS) – найбільший хмарний провайдер, що пропонує широкий спектр інструментів та послуг. Його сильні сторони включають велику надійність, масштабованість та інтеграцію з широким діапазоном сервісів. Недоліками є складність оцінки вартості та прив'язка до технологій AWS.

Microsoft Azure – хмарний провайдер послуг від компанії Microsoft, відомий своєю сильною інтеграцією з Microsoft-орієнтованими рішеннями, що робить його привабливим для компаній, які вже їх використовують. Слабкі сторони включають меншу гнучкість у порівнянні з AWS у деяких сценаріях використання та прив'язку до екосистеми Azure.

Google Cloud Platform (GCP) – хмарна платформа від компанії Google. Вирізняється високою продуктивністю обчислювальних сервісів та сильними можливостями в сфері машинного навчання та великих даних. Високоякісна підтримка Kubernetes у вигляді сервісу. Втім, частка на ринку менша, ніж у AWS та Azure, що може впливати на розмір екосистеми та спільноти.

Oracle Cloud Infrastructure (OCI) – хмарна платформа від компанії Oracle, що надає широкий спектр різноманітних сервісів. Має дата-центри по всьому світу та можливості розгортання гібридної інфраструктури. Має тісну інтеграцію з іншими продуктами Oracle (бази даних, бізнес-додатки), високу продуктивність для роботи з великими даними і навантаженнями та нижчу вартість для великих обсягів даних та обчислень. Менша поширеність, кількість користувачів, додаткових сервісів та можливостей у порівнянні з лідерами ринку.

1.5 Висновки з аналізу інструментальних засобів

Проведений в роботі аналіз дозволяє сформулювати рекомендації щодо вибору компонентів для побудови автоматизації CI\CD процесів для хмарних інформаційних систем. Результати аналізу у вигляді пропозицій оптимальних інструментів за основними напрямками наведено в таблиці 1.2.

Таблиця 1.2 – Рекомендації інструментальних засобів

Призначення	Інструмент	Переваги
Контроль версій	Git	Розподілена робота, ефективне управління, висока швидкість операцій, висока безпека.
Хостинг програмного коду	GitHub	Зручний інтерфейс, широкі можливості для інтеграції з іншими інструментами, ефективне відстеження проблем.
Безперервна інтеграція	GitHub Actions	Інтеграція з GitHub, гнучкість, проста мова сценаріїв, підтримка багатьох платформ.
Автоматизація збірки та тестів	Bazel	Висока швидкість, збірка та тестування лише змінених частин на основі аналізу.
Безперервна доставка	Argo CD	Автоматизація доставки оновлень в Kubernetes. синхронізація конфігурації з Git, візуалізація змін, простота налаштування.
Безперервне розгортання	Kubernetes	Ефективне управління мікросервісами, автоматичне масштабування, незалежність від хмарної платформи, самовідновлення.
Моніторинг	Prometheus, Grafana	Збір та аналіз широкого спектру метрик, ефективність обробки великих обсягів даних, широкі можливості візуалізації.
Хмарний провайдер	GCP	Великий хмарний провайдер, високоякісна підтримка Kubernetes у вигляді сервісу.

1.6 Постановка задачі дослідження

Автоматизацію CI\CD для хмарних інформаційних систем у загальному випадку можливо розділити на декілька задач: задача автоматизації самих CI\CD процесів для хмарної інформаційної системи, задача автоматизованого формування інфраструктури для хмарної інформаційної системи, задача оптимізації CI\CD процесів для хмарної інформаційної системи з метою заощадження коштів та мінімізації витрат часу на CI\CD процеси. На рисунку 1.9 наведений список цих задач та обрані задачі автоматизації.

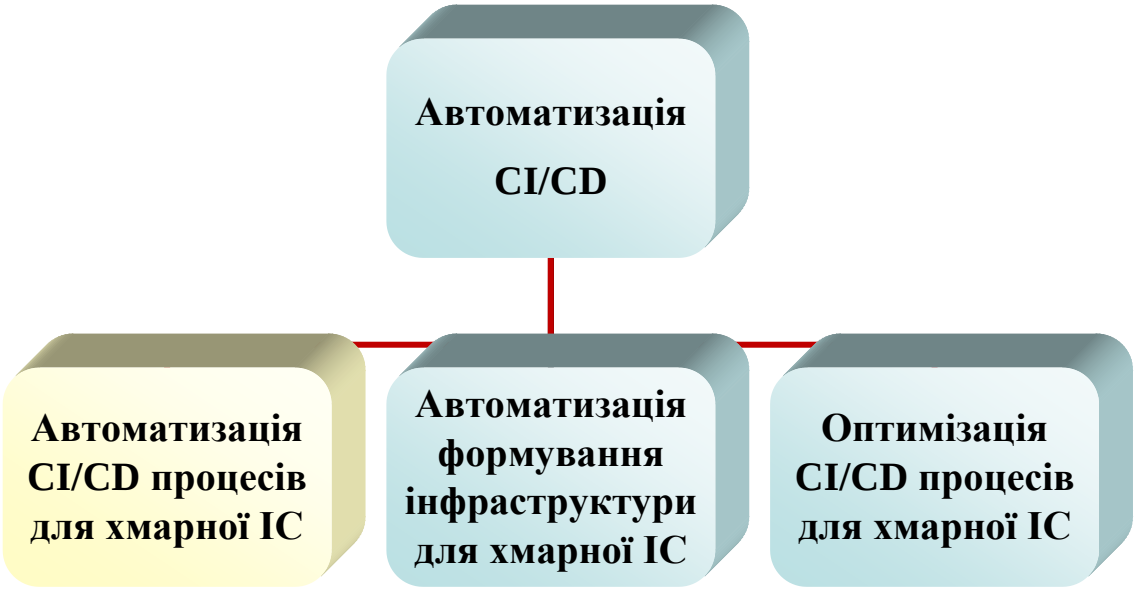

Рисунок 1.9 – Обрана задача автоматизації

Серед сформульованих вище задач, в якості основної для даного дослідження обрано задачу автоматизації CI\CD процесів для хмарної інформаційної системи, на основі якої буде створений метод.

Основний критерій системи – ефективність.

На основі огляду інструментальних засобів, було визначено, що Kubernetes, як середовище для розгортання хмарних ІС, найкраще відповідає

вимогам проекту, забезпечуючи високий рівень надійності та незалежність від конкретного хмарного провайдера.

Вихідним результатом є метод побудови автоматизованого конвеєра CI\CD для забезпечення ефективної автоматизації процесів безперервної інтеграції, тестування, доставки та розгортання оновлень для мікросервісів хмарної інформаційної системи.

Для отримання інформації будуть задіяні кілька баз даних, а саме:

– база даних для керування репозиторіями коду та контролю доступу користувачів;

– база даних для зберігання метрик CI\CD процесів, включаючи результати тестувань та статуси збірок.

При проектуванні інформаційного комплексу важливо передбачити можливість розширення обсягу та зміни структури технологічних процесів у відповідності до потреб конкретної сфери діяльності.

Робота з даною системою вимагає наявності персоналу, який має всебічне розуміння функціонування хмарних технологій.

Інформаційна система повинна бути спроектована з урахуванням можливості легкого масштабування, щоб відповідати зростаючим потребам в обробці даних та кількості користувачів. Це передбачає здатність системи ефективно використовувати додаткові ресурси при їх наявності.

Інформаційна система має забезпечувати стабільну роботу та мати здатність до швидкого відновлення у випадку збоїв.

Інформаційна система повинна бути гнучкою для внесення змін та додавання нових функцій з мінімальними затратами часу та ресурсів, забезпечуючи можливість адаптації під змінні вимоги.

Інформаційна система повинна передбачати різні рівні доступу для користувачів в залежності від їх ролей та обов'язків, запобігаючи несанкціонованому доступу до конфіденційної інформації.

2 РОЗРОБКА МЕТОДУ АВТОМАТИЗАЦІЇ CI\CD ПРОЦЕСІВ ДЛЯ ХМАРНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

2.1 Опис об’єкту дослідження

Об'єктом дослідження в даній роботі є хмарна інформаційна система, розроблена на основі мікросервісної архітектури з CI\CD процесами, які використовуються для автоматизації розгортання та оновлення компонентів цієї системи. У фокусі дослідження знаходяться методи інтеграції, тестування, доставки та розгортання мікросервісів у хмарному середовищі. Структурна схема об’єкту дослідження наведена на рисунку 2.1.

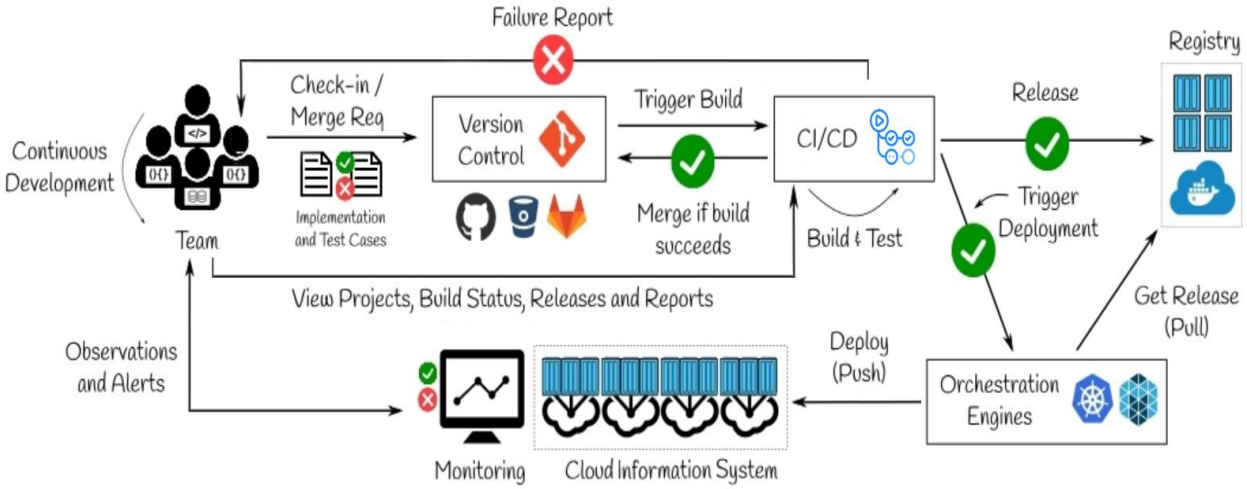

Рисунок 2.1 – Структурна схема об’єкту дослідження

На рисунку 2.1 представлені CI\CD процеси, які ілюструють послідовність автоматизованих кроків – від розробки до розгортання. Ці процеси включають етапи автоматичного тестування, інтеграції, збірки та доставки змін у коді до хмарної ІС. Хмарна ІС може включати сервери, бази даних та інші важливі компоненти, куди доставляються оновлення.

В дослідженні, відповідно до постановки завдання, потрібно реалізувати автоматизацію CI\CD процесів для хмарних інформаційних систем, згідно наведеної схеми на рисунку 2.2.

Рисунок 2.2 – Схема автоматизації CI\CD процесів

Для оцінки ефективності рішення з автоматизації CI\CD процесів було сформовано наступний перелік критеріїв, що відповідають поставленим цілям автоматизації:

Tdep → min – зменшення тривалості процесу розгортання та внесення змін у хмарну інформаційну систему, зменшення ймовірності виникнення помилок при впровадженні та скорочення часу необхідного для їх виправлення.

Cpr → min – спрощення процесів для команди розробки та тестування;

Ssp → max – підвищення швидкості масштабування та зміни конфігурації системи.

Реалізація проекту з автоматизації CI\CD процесів потребуватиме наступних витрат:

– витрати на дослідження існуючих процесів (вивчення особливостей поточних CI\CD процесів, їх недоліків, аналіз можливостей оптимізації);

– витрати на проєктування системи автоматизації (розробка архітектури, вибір інструментів, планування етапів впровадження);

– витрати на розгортання хмарної інфраструктури (налаштування хмарних сервісів та інтеграція їх в систему автоматизації);

– витрати на розробку програмного забезпечення (створення необхідного коду та скриптів автоматизації);

– витрати на тестування та налагодження (перевірка та виправлення помилок в реалізованій системі);

– витрати на впровадження та навчання (запуск системи, навчання користувачів).

2.2 Користувачі системи дослідження

Ефективне функціонування будь-якої інформаційної системи, в тому числі розгорнутої у хмарному середовищі, можливе лише за умови злагодженої взаємодії між усіма задіяними користувачами. Саме тому визначення та аналіз цільових груп користувачів є важливим етапом в проектуванні та впровадженні системи.

Користувачів інформаційної системи можна умовно поділити на дві великі групи: функціональних та технічних. До функціональних відносять безпосередніх користувачів системи, які задовольняють свої бізнес-потреби за допомогою її можливостей. Технічні користувачі забезпечують налаштування, підтримку та розвиток інформаційної системи.

Основні групи користувачів CI\CD конвеєру, що забезпечує безперервну доставку змін у хмарну інформаційну систему, як і групи користувачів самого хмарного інформаційного рішення зображено на рисунку 2.3. До цих груп належать розробники (developers), тестувальники (testers), DevOps інженери (DevOps), менеджери проекту (project managers), власник продукту (owner) та кінцеві користувачі хмарних сервісів (users).

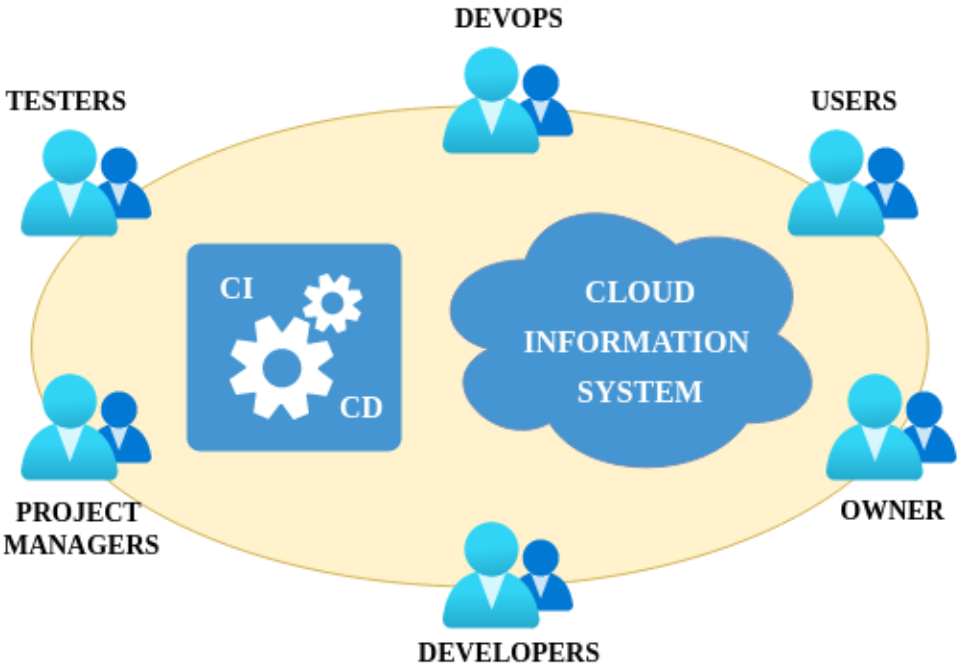

Рисунок 2.3 – Користувачі системи дослідження

Розробники є ключовими учасниками CI\CD процесів. Вони пишуть код, зберігають його у репозиторії, ініціюють автоматизовані процеси збірки та тестування та відстежують результати для виявлення та виправлення помилок.

Тестери зосереджені на забезпеченні якості. Використовують автоматизовані інструменти CI\CD для проведення різних видів тестів, включаючи функціональні, інтеграційні, навантажувальні та інші.

DevOps інженери відіграють центральну роль у налаштуванні та управлінні CI\CD конвеєрами. Вони забезпечують інтеграцію різних інструментів, створюють скрипти для автоматизації, налаштовують середовища для розгортання та виконують моніторинг продуктивності і безпеки систем.

Менеджери проекту фокусуються на плануванні, координації та відстеженні прогресу проектів. У контексті CI\CD, менеджери проекту стежать, щоб оновлення ПЗ хмарної ІС відбувалося згідно з графіком та відповідало якісним стандартам.

Власники продукту фокусуються на визначенні вимог до продукту та пріоритетів. Тісно співпрацюють з розробниками та менеджерами проекту для забезпечення відповідності кінцевого продукту бізнес-цілям.

Кінцеві користувачі хмарних сервісів не беруть безпосередньої участі в процесах CI\CD, але їх зворотний зв'язок є важливим для вдосконалення самого продукту, оскільки вони використовують функціонал та сервіси, які надаються через хмарну інформаційну систему.

Для систематизації та кращого розуміння функцій учасників, доцільно побудувати формалізований опис користувачів у вигляді наборів операцій, що ними виконуються в рамках CI\CD процесів та при роботі з хмарною інформаційною системою. Вирази, що визначають кожну групу учасників через характерні для них дії та задачі можуть бути представлені у вигляді:

Developers (Розробники) = {розробляють програмне забезпечення ІС, ініціюють збірки та тести, відслідковують результати, виправляють помилки}.

Testers (Тестери) = {налаштовують та виконують різні види тестів, створюють звіти про помилки}.

DevOps (DevOps інженери) = {інтегрують інструменти, автоматизують процеси, налаштовують середовища, виконують моніторинг продуктивності та безпеки}.

Project Managers (Менеджери проектів) = {планують, координують, відслідковують прогрес проекту та якість}.

Owner (Власник) = {визначають вимоги, встановлюють пріоритети, забезпечують відповідність цілям}.

Users (Користувачі) = {використовують функціонал, надають зворотний зв'язок про продукт}.

Таким чином, кожна група користувачів відіграє унікальну та важливу роль у CI\CD процесах, вносячи свій вклад у стабільність, надійність та постійне вдосконалення хмарної інформаційної системи.

2.3 Детальний опис об'єкту дослідження

У контексті хмарної інформаційної системи, що базується на мікросервісній архітектурі та реалізується через середовище оркестрування контейнерів, CI\CD процеси організовані навколо двох основних конвеєрів: конвеєра збірки та конвеєра випуску. Конвеєр збірки відіграє вирішальну роль у процесі безперервної інтеграції, запускаючи збірку коду та формуючи артефакти, які необхідні для розгортання. В мікросервісній архітектурі ці артефакти зазвичай включають образи контейнерів та конфігураційні файли, кожен з яких представляє окремий мікросервіс або його компонент.

Процес починається з моменту внесення змін до коду, що автоматично ініціює ряд операцій у конвеєрі. Першим кроком є верифікація коду, яка включає статичний аналіз коду та інші перевірки якості для виявлення помилок або вразливостей. Після цього відбувається компіляція коду, що перетворює вихідний код на виконувані програми або бібліотеки.

Наступним етапом є запуск автоматизованих тестів, які включають unit-тести для перевірки окремих компонентів. Успішне проходження цих тестів є ключовим для переходу до наступного кроку – створення артефактів. Артефакти представляють собою образи контейнерів, готові до розгортання в середовищі оркестрування контейнерів, а також необхідні конфігураційні файли для цього.

Конвеєр випуску в CI\CD процесах для хмарної ІС з мікросервісною архітектурою відповідає за безперервну доставку, забезпечуючи ефективне розгортання артефактів, які були створені та протестовані на етапі конвеєра збірки, у відповідні системи оркестрування контейнерів. Процес розгортання охоплює не тільки фізичне розміщення мікросервісів, а й критично важливе управління такими аспектами, як масштабування, балансування навантаження та забезпечення стабільної роботи та доступності сервісів.

Важливою частиною цього процесу є інтеграційні тести для перевірки взаємодії між різними частинами системи, системні тести для перевірки поведінки всієї системи, управління змінами та версіями, включаючи відстеження різних версій артефактів та їх конфігурацій, а також управління різними середовищами для розгортання (тестового, виробничого та інших).

Комплексний CI\CD процес для хмарної ІС з мікросервісною архітектурою та середовищем оркестрування контейнерів, можна подати у вигляді схеми, яка наведена на рисунку 2.4. Ця схема демонструє ключові компоненти та етапи обох конвеєрів, від ініціації змін у коді, через процеси верифікації, компіляції, тестування, до створення артефактів і їх розгортання у тестовому та виробничому середовищах. Ця візуалізація відображає взаємозв'язки та комплексність процесів CI\CD, ілюструючи повний цикл розвитку та доставки програмного продукту.

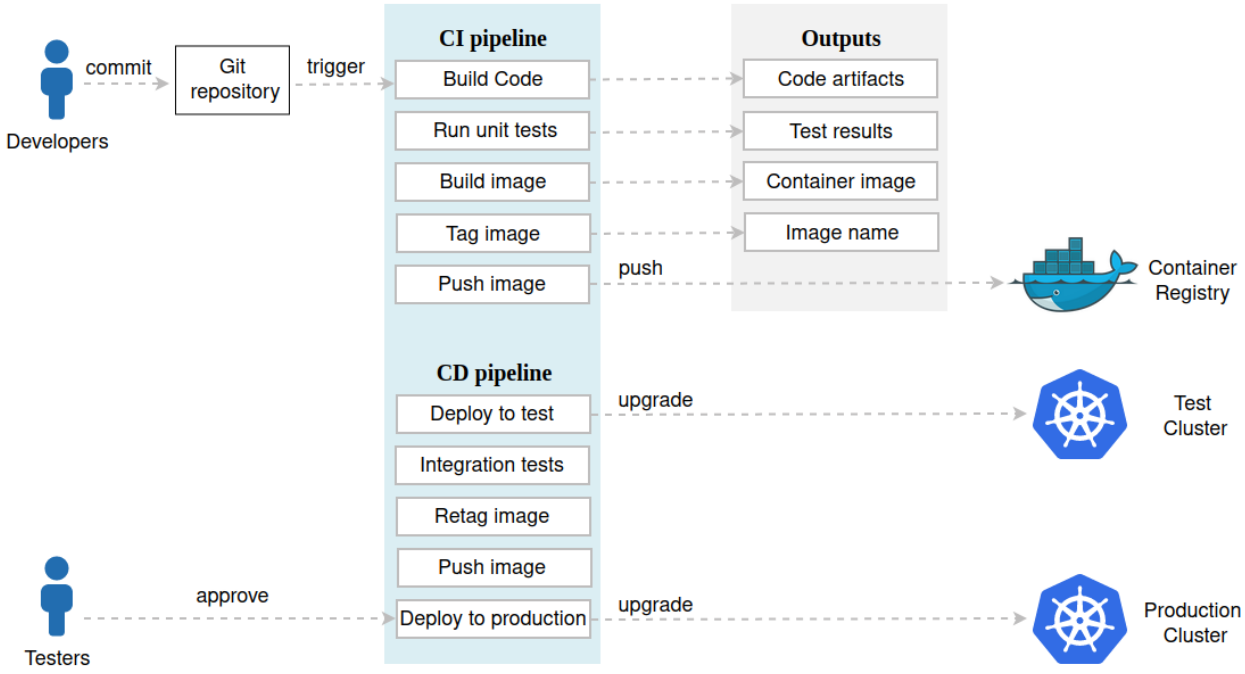

Рисунок 2.4 – Комплексний CI\CD процес для хмарної ІС

Після розгортання артефактів, система переходить до етапу моніторингу. Моніторинг забезпечує постійне спостереження за станом мікросервісів, включаючи їх продуктивність, доступність та виявлення помилок. Системи логування збирають дані про роботу мікросервісів, що дозволяє аналізувати причини збоїв та вдосконалювати систему.

Управління ресурсами хмарної інфраструктури, включаючи розподіл та масштабування ресурсів залежно від потреб системи, забезпечує гнучке реагування на зміни навантаження, забезпечуючи високу доступність сервісів та оптимальне використання хмарних ресурсів.

Завершальним етапом у цьому безперервному циклі є фаза аналізу та вдосконалення, де збираються дані про ефективність роботи системи, задоволеність користувачів та інші ключові показники. Ці дані використовуються для планування подальших ітерацій розвитку системи, включаючи внесення змін у функціональність мікросервісів, оптимізацію процесів CI\CD, а також планування розгортання нових версій. Цей аналітичний підхід забезпечує не тільки виявлення проблемних областей та їх виправлення, але й виявлення можливостей для вдосконалення, дозволяючи системі постійно адаптуватися до змінних вимог та умов.

Взаємодія та інтеграція між конвеєрами збірки та випуску, а також між різними компонентами процесу CI\CD, є фундаментальною для досягнення ефективності та гнучкості хмарної інформаційної системи. Ця інтеграція забезпечує гладкий і безперервний рух коду від розробки до впровадження, дозволяючи швидко реагувати на зміни, вдосконалювати продукт та забезпечувати високий рівень задоволеності користувачів.

Отже, CI\CD процеси для хмарної ІС на базі мікросервісної архітектури формують динамічну та інтерактивну систему, яка не тільки підтримує постійний потік розробки та впровадження програмного забезпечення, але й забезпечує надійність, продуктивність та відповідність до поточних та майбутніх вимог бізнесу та користувачів.

2.4 Функціональні вимоги до автоматизації CI\CD процесів для хмарних інформаційних систем

Для наочного моделювання та аналізу CI\CD процесів використано діаграми UML (Unified Modeling Language)[17]. Зокрема, діаграми варіантів використання (use case diagrams) дозволяють описати функціональні вимоги та взаємозв’язки учасників цих процесів. Детальне відображення цих взаємозв'язків та функціональних вимог представлено на рисунку 2.5, де наведено діаграму варіантів використання автоматизованих CI\CD процесів для XIC, ілюструючи комплексний підхід до моделювання цих процесів.

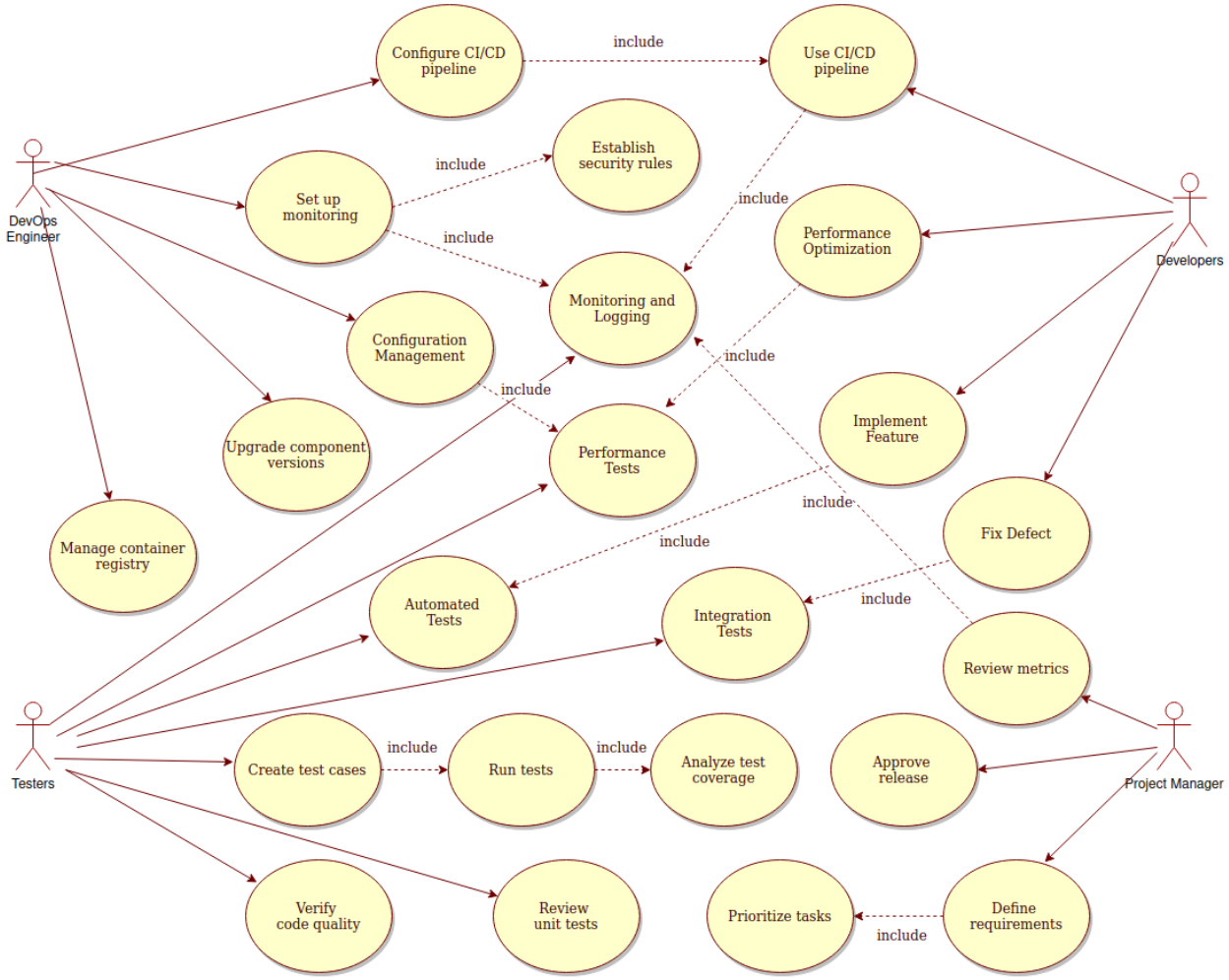

Рисунок 2.5 – Діаграма варіантів використання автоматизованих CI\CD процесів для хмарних інформаційних систем

Для доповнення діаграми варіантів використання, у таблиці 2.1 наведено користувацькі сценарії (User Stories) для кожного з ключових акторів [18]. Ці User Stories конкретизують функціональні вимоги та очікувані результати, сприяючи глибшому розумінню взаємодії між різними ролями в процесі автоматизації CI\CD.

Таблиця 2.1 – User stories автоматизації CI\CD процесів для хмарних інформаційних систем

Номер User Story	Опис
1	DevOps інженерам необхідно мати можливість налаштовувати і підтримувати CI\CD інфраструктуру, щоб забезпечити надійність і ефективність процесів автоматизації.
2	DevOps інженерам необхідно мати можливість швидко виявляти і усувати проблеми в процесах CI\CD, щоб мінімізувати простої в роботі і забезпечити стабільність функціонування системи. Умови: реалізована user story 1.
3	Інженерам з якості ПЗ необхідно, щоб автоматизовані тести запускались після кожного оновлення коду в основному репозиторії, для забезпечення високого рівня якості продукту. Умови: реалізована user story 1.
4	Інженерам з якості ПЗ необхідно мати можливість легко налаштовувати тестові сценарії та інтегрувати їх у CI\CD процеси, щоб ефективно перевіряти різні аспекти системи. Умови: реалізована user story 1.
5	Розробникам необхідно отримувати зворотний зв'язок від системи CI\CD про будь-які помилки або проблеми з їх кодом якомога швидше, щоб ефективно виправити їх. Умови: реалізовані user story 1,4.

Кінець таблиці 2.1

Номер User Story	Опис
6	Розробникам необхідно мати змогу легко інтегрувати код в спільний репозиторій і бачити результати автоматизованих тестів, щоб забезпечити якість і сумісність їх коду з іншими частинами проекту. Умови: реалізовані user story 3.
7	Менеджерам проекту необхідно мати можливість переглядати звіти про стан розробки та випуску програмного забезпечення, щоб забезпечити вчасне і якісне виконання проекту. Умови: реалізована user story 1.
8	Менеджерам проекту необхідно отримувати сповіщення про будь-які проблеми або затримки в CI\CD процесах, щоб своєчасно спрямовувати ресурси на їх усунення. Умови: реалізована user story 2.
9	Менеджерам проекту необхідно мати можливість відслідковувати прогрес реалізації функціоналу в рамках спринту, щоб забезпечити дотримання планів проекту. Умови: реалізована user story 1,5,6.
10	DevOps інженерам необхідно мати можливість масштабування інфраструктури в залежності від навантаження, щоб оптимізувати використання ресурсів. Умови: реалізована user story 1.

Діаграму класів автоматизованого конвеєру CI\CD процесів для хмарних ІС, яка детально показує класи системи, їхні атрибути та операції, а також відображає взаємозв'язки між ними, наведено на рисунку 2.6. Опис класів представлено у таблиці 2.2.

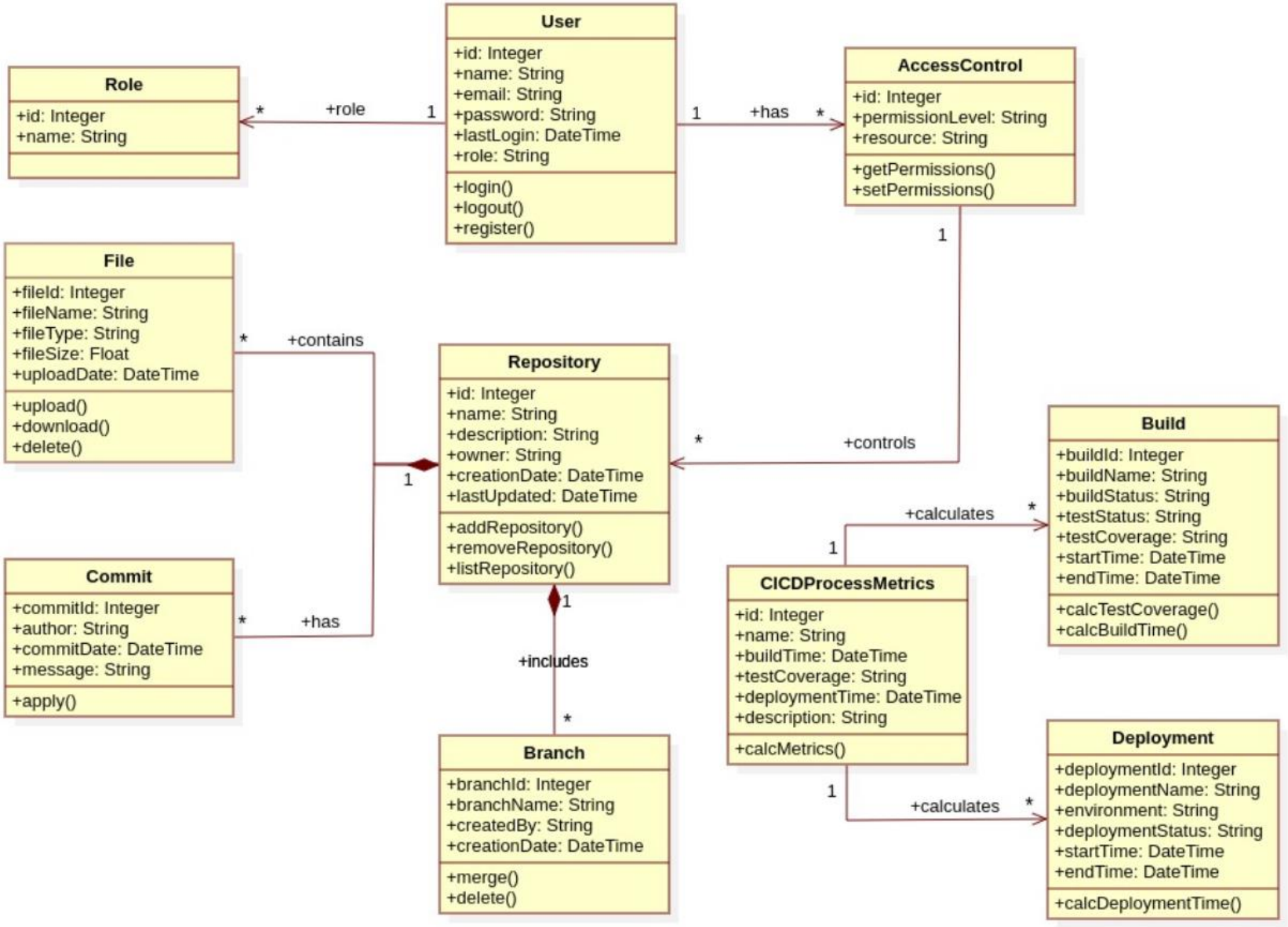

Рисунок 2.6 – Діаграма класів автоматизованого конвеєру CI\CD процесів для хмарних інформаційних систем

Таблиця 2.2 – Опис класів автоматизованого конвеєру CI\CD процесів для хмарних інформаційних систем

Клас	Опис
User	Описує користувача системи.
Role	Описує роль користувача.
AccessControl	Описує систему контролю доступу, яка визначає, які користувачі мають доступ до яких ресурсів.
Repository	Описує сховище коду, де зберігаються всі версії проекту.
Branch	Описує гілку у репозиторії, яка дозволяє розробникам працювати над окремими функціями або виправленнями.
Commit	Описує фіксацію змін у репозиторії, яка містить деталі про автора змін та опис змін.
File	Описує файл у репозиторії, який може містити код, документацію або інші ресурси проекту.
CICDProcessMetrics	Описує метрики, пов'язані з процесами безперервної інтеграції, доставки та розгортання програмного забезпечення для хмарної інформаційної системи.
Build	Описує деталі процесу збірки проекту, включаючи ідентифікатор збірки, статус, час початку та завершення збірки, а також інші метрики, які допомагають оцінити ефективність процесу збірки.
Deployment	Описує процес розгортання ПЗ. Включає інформацію про ідентифікатор розгортання, цільове середовище, статус розгортання, час початку та завершення розгортання для подальшої оцінки ефективності процесу розгортання.

2.5 Загальні вимоги автоматизації CI\CD процесів для хмарних інформаційних систем

Загальні вимоги автоматизації CI\CD процесів для хмарних інформаційних систем включають розробку комплексу скриптів і маніфестів та їх налаштування. Цей комплекс складається з передумов, налаштувань інтеграції з системою контролю версій Git та сховищем GitHub, скрипту GitHub Actions для запуску автоматизації CI\CD процесів, скриптів Bazel для підвищення продуктивності CI процесів, маніфестів ArgoCD для керування CD процесами, конфігураційних файлів Kubernetes для автоматизації розгортання сервісів та маніфестів Prometheus і Grafana для налаштування моніторингу.

1. Передумови. S1 = {PQa <Cloud акаунт>, PQb <GitHub акаунт>, PQc <Container Registry акаунт>, PQd <Kubernetes кластер>}.

PQa <Cloud акаунт> – створення та налаштування хмарного акаунту для розгортання ресурсів.

PQb <GitHub акаунт> – створення акаунту GitHub та ініціалізація репозиторію для зберігання коду, скриптів, маніфестів та інших конфігураційних файлів.

PQc <Container Registry акаунт> – створення акаунту у хмарному реєстрі контейнерів для зберігання образів контейнерів.

PQd <Kubernetes кластер> – розгортання кластеру Kubernetes у хмарі та налаштування доступу.

2. Конфігурація Git та GitHub. S2 = {GTa, GTb, GTc, GTd, GTe, GTf }.

GTa – створення локального репозиторію проекту та його прив'язки до віддаленого у GitHub.

GTb – встановлення імені та поштової адреси користувачів Git для ідентифікації авторів майбутніх завантажень.

GTc – генерування SSH ключів для аутентифікації користувачів при взаємодії з віддаленими GIT репозиторієм по протоколу SSH.

GTe – реєстрація публічної частини SSH ключа в секретах акаунта GitHub для надання доступу до віддаленого репозиторію по SSH.

GTf – перевірка зв'язку клієнта Git з серверами віддаленого репозиторію для верифікації налаштувань аутентифікації.

3. Конфігурація Bazel. S3 = {BZa, BZb, BZc, BZd}.

BZa = {BZa1, BZa2, BZa3} – визначення залежності та правил збірки додатку.

BZa1 – імпорт зовнішніх бібліотек та фреймворків, необхідних для компіляції та пакування коду додатку.

BZa2 – створення опису логічної структури додатку з визначенням окремих його компонентів (бібліотек, сервісів, утиліт) та залежності між цими компонентами, для визначення порядку та правил побудови додатку при автоматичній збірці.

BZa3 – встановлення та конфігурування параметрів для основних етапів збірки додатку, а саме: компіляції коду (компілятори, опції, стандарти), пакування скомпільованих модулів у статичні та динамічні бібліотеки, архіви, файли, контейнери (формати, стиснення, структури).

BZb = {BZb1, BZb2, BZb3} – визначення конфігурації тестування.

BZb1 – розробка та опис наборів unit-тестів для верифікації коректності окремих ключових компонент додатку з подальшим їх інтегруванням в процес автоматизованої збірки.

BZb2 – конфігурування наборів end-to-end тестів, що перевіряють працездатність комплексу мікросервісів додатку при їх сумісній роботі в рамках єдиної системи і подальшою інтеграцією даних тестів в CI систему.

BZb3 – налаштування експорту звітів про проходження тестів та збір детальної статистики щодо статусу проходження окремих тест-кейсів з подальшим включенням згенерованих файлів тестової аналітики в артефакти CI\CD для передачі на наступні етапи циклу релізу.

BZc = {BZc1, BZc2} – налаштування модульної збірки.

BZc1 – налаштування механізму кешування проміжних результатів збірки, отриманих на попередніх етапах конвеєру безперервної інтеграції, для повторного використання об'єктів у кеші при наступних його запусках, що прискорює роботу конвеєру за рахунок часткової збірки лише змінених модулів, а не всього додатку.

BZc2 – налаштування режиму збірки на базі кешованих проміжних результатів попередніх запусків.

BZd = {BZd1, BZd2} – експорт результатів збірки та тестів.

BZd1 – опис та експорт бінарних файлів додатку (виконуваних модулів, бібліотек, архівів, скриптів), згенерованих Bazel на основі вихідного коду та конфігурацій збірки, для подальшого використання результатів збірки на наступних етапах CI\CD процесу.

BZd2 – експорт звітів щодо статусу проходження тестів, а також іншої вихідної документації та аналітики, згенерованої Bazel на етапі автоматизованої збірки та тестування коду для аналізу якості збірки та прийняття рішень в CI\CD процесі.

4. Скрипт GitHub Actions. S4 = {GAa, GAb, GAc}.

GAa – опис змінних для налаштування робочого процесу (workflow) з підключенням та авторизацією до віддаленого реєстру контейнерів (Container Registry) у вигляді URL розташування, логіна та пароля доступу для можливості автоматичного завантаження створених образів контейнерів з мікросервісами додатку в ході роботи автоматизованого CI\CD конвеєру.

GAb = { GAb1, GAb2} – налаштування автоматизованої збірки образів контейнерів з мікросервісами додатку.

GAb1 – опис завдання, що на першому етапі завантажує артефакти у вигляді бінарних файлів та інших результатів роботи системи збірки Bazel для їх подальшого використання при побудові образів контейнерів мікросервісів додатку в автоматизованому CI\CD конвеєрі.

GAb2 – опис завдання, що автоматизує створення образів контейнерів з мікросервісами додатку використовуючи завантажені артефакти Bazel збірки та конфігураційні файли (Dockerfiles), в яких описані інструкції, які будуть застосовані під час створення образів контейнерів та їх запуску з подальшим призначенням номерів версії згідно встановленої політики маркування версій.

GAc = { GAc1, GAc2} – створення тригерів для запуску автоматизації.

GAc1 – налаштування тригера-події при запиті на внесення змін у головну гілку репозиторію (pull request) з метою автоматизації процесів збірки та тестування коду перед його основним злиттям в головну гілку репозиторію для реалізації практик DevOps культури безперервної інтеграції.

GAc2 – налаштування тригера-події при внесенні змін у головну гілку репозиторію (main/master) для автоматичного запуску конвеєра при кожному оновленні коду додатку, що дозволить реалізувати безперервну інтеграцію з автоматичною збіркою, тестуванням та завантаженням нових версій програмного забезпечення до реєстру контейнерів.

5. Маніфести Argo CD. S5 = {ACa, ACb, ACc, ACd, ACe }.

ACa = {ACa1, ACa2, ACa3, ACa4} – інсталяція та налаштування параметрів підключення.

ACa1 – встановлення екземпляру Argo CD в окремий простір імен кожного кластеру за допомогою відповідних Kubernetes маніфестів.

ACa2 – генерування SSH ключа для аутентифікації та отримання доступу до приватного репозиторію у GitHub.

ACa3 – реєстрація публічної частини SSH ключа в налаштуваннях GitHub акаунту з метою надання конкретному екземпляру ArgoCD доступу до репозиторію з маніфестами для розгортання додатку у кластері Kubernetes.

ACa4 – перевірка підключення до приватного GitHub репозиторію з метою верифікації коректності налаштувань конкретного екземпляру ArgoCD.

ACb – налаштування синхронізації конкретного екземпляру Argo CD, розгорнутого в окремому Kubernetes кластері (dev, stage, prod) з відповідною цьому середовищу папкою з маніфестами у GitHub репозиторії.

ACc – розробка та збереження у відповідні папки у GitHub репозиторії конфігураційних файлів (маніфестів) Argo CD для кожного середовища розгортання (dev, stage, prod).

ACd — встановлення політики автоматичного оновлення додатку розгорнутого в Kubernetes кластері середовища розробки (dev), що дозволить розробникам швидко тестувати результати роботи без необхідності ручного втручання.

ACe – налаштування стратегії оновлення додатку у Kubernetes кластерах на основі ручного підтвердження змін людиною з метою впровадження додаткового контролю та верифікації оновлень перед впровадженням їх у середовища близькі до реальної, або реальної експлуатації (stage, prod).

6. Маніфести Kubernetes. S6 = {KNa, KNb}.

KNa – розробка Kubernetes маніфестів у форматі YAML з описом розгортання окремих компонентів та мікросервісів додатку для відповідних середовищ, включаючи deployment, service, ingress та інші необхідні об’єкти для запуску та взаємодії сервісів додатку.

KNb – налаштування автоматизованого оновлення компонентів додатку у відповідному Kubernetes кластері при змінах в маніфестах розгортання відповідного середовища.

7. Конфігурація Prometheus і Grafana. S7 = {PGa, PGb}.

PGa – встановлення Prometheus та налаштування джерел метрик, конфігурування правил агрегацій цих метрик на основі вихідних даних з експортерів для генерування узагальнених показників по ресурсам кластеру для обчислення середніх значень, налаштування збереження метрик у сховище даних.

PGb – розробка та конфігурування інформаційної панелі в Grafana на основі даних з Prometheus для візуалізації метрик роботи Kubernetes кластеру, а саме: утилізації ресурсів, роботи контролерів Kubernetes, статусів та продуктивності розгортань і роботи додатку.

2.6 Метод автоматизації CI\CD процесів для хмарних інформаційних систем

Базуючись на загальних принципах та вимогах до автоматизації CI\CD процесів для хмарних інформаційних систем, можна побудувати узагальнений опис методу автоматизації цих процесів.

Розроблений метод автоматизації CI\CD процесів для хмарних інформаційних систем представлено у вигляді послідовності кроків: S1→S2→S3→S4→S5→S6→S7.

Крок 1. Створення необхідних облікових записів та підготовка компонентів інфраструктури для подальшого розміщення ресурсів. Реєстрація акаунту GitHub та ініціалізація GIT-репозиторію для збереження вихідного коду та конфігурацій, створення реєстру контейнерів для зберігання образів контейнерів з компонентами ІС, реєстрація акаунту у хмарі та розгортання Kubernetes кластерів відповідно до обраних середовищ експлуатації ІС. S1 = {PQa <Cloud акаунт>, PQb <GitHub акаунт>, PQc <Container Registry>, PQd <Kubernetes кластер>}.

Крок 2. Конфігурація засобів колективної розробки Git та GitHub. Встановлення та налаштування системи контролю Git для подальшої колективної розробки, встановлення зв'язку між локальним репозиторієм та віддаленим у GitHub, генерація SSH ключів для аутентифікації користувачів при взаємодії з віддаленими GIT репозиторієм по SSH-протоколу, перевірка зв'язку клієнта Git з серверами віддаленого репозиторію для верифікації налаштувань. S2 = {GTa, GTb, GTc, GTd, GTe, GTf}. Виконується за допомогою конфігурації Git та GitHub.

Крок 3. Розробка правил Bazel автоматизації збірки, тестування та пакування коду компонентів ІС, інтеграція основних видів тестів для автоматичного їх запуску при кожній новій збірці, налаштування механізму кешування проміжних результатів збірки для повторного їх використання та

експорту у конвеєр CI\CD. S3 = {BZa, BZb, BZc, BZd}. Виконується за допомогою конфігурації Bazel.

Крок 4. Створення завдання GitHub Actions Workflow у відповідному репозиторії для побудови автоматизованого CI\CD конвеєру, додавання змінних у GitHub середовище з параметрами доступу до хмарної платформи, налаштування завдання для збірки образів контейнерів з компонентами ІС на основі артефактів компіляції експортованих з Bazel, призначення образам номерів версій згідно встановленої політики маркування з подальшим їх завантаження у реєстр контейнерів, налаштування тригерів автоматичного запуску CI\CD конвеєру при оновленні коду у відповідних гілках репозиторію. S4 = {GAa, GAb, GAc}. Виконується за допомогою скриптів GitHub Actions.

Крок 5. Встановлення системи доставки оновлень ArgoCD у Kubernetes кластер для кожного із середовищ експлуатації ІС, конфігурування екземплярів ArgoCD на відстеження змін у відповідних гілках Git репозиторію, налаштування автоматичного або ручного оновлення компонентів ІС у відповідних кластерах. S5 = {ACa, ACb, ACc}. Виконується за допомогою маніфестів ArgoCD.

Крок 6. Розробка конфігурацій Kubernetes у форматі YAML-файлів для опису розгортання компонентів ІС, групування та завантаження цих файлів у відповідні гілки Git репозиторію для подальшого відстеження в них змін ArgoCD операторами і автоматичного запуску розгортання згідно встановленої політики розгортань. S6 = {KNa, KNb}. Виконується за допомогою маніфестів Kubernetes.

Крок 7. Встановлення і налаштування системи моніторингу на базі Prometheus та Grafana для збору метрик роботи кластеру та працюючої в ньому ІС, налаштування збереження зібраних метрик у сховище даних та подальшої їх візуалізації. S7 = {PGa, PGb}. Виконується за допомогою конфігурацій Prometheus та Grafana.

2.7 Пояснення до методу автоматизації CI\CD процесів для хмарних інформаційних систем

Розроблений метод автоматизації дозволяє комплексно охопити всі етапи сучасних CI\CD процесів для хмарних ІС. Для кращого розуміння запропонованого підходу було побудовано схему, яка демонструє потоки даних між основними компонентами автоматизованого CI\CD конвеєру (рисунок 2.7).

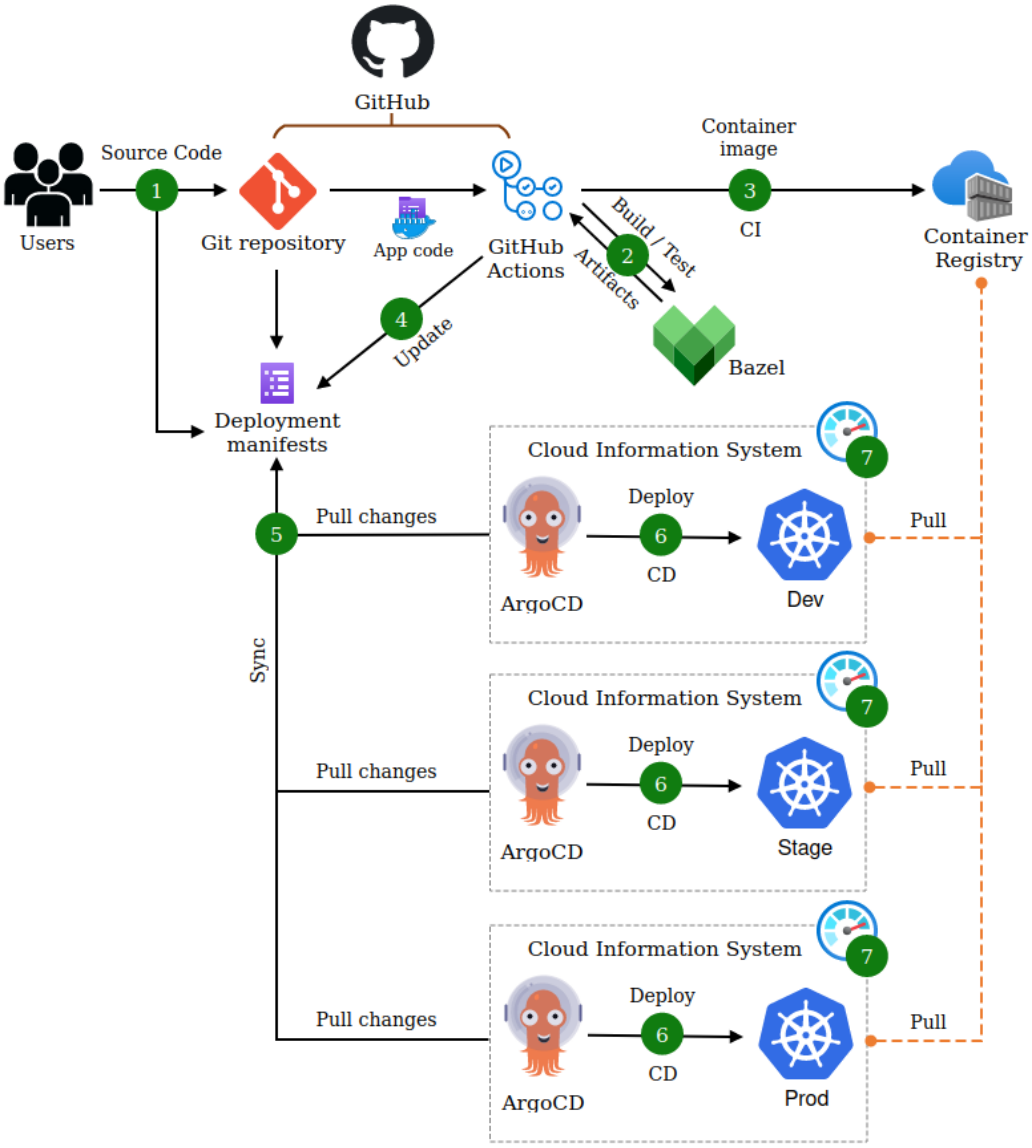

Рисунок 2.7 – Схема роботи автоматизованого конвеєру CI\CD процесів для хмарних інформаційних систем

На діаграмі візуалізовано рух даних між ключовими компонентами розробленого рішення (код, артефакти, конфігурації, метрики). Крім того вона дає уявлення про зв'язки та інформаційний обмін між системами керування версіями, збірки, тестування, розгортання та моніторингу.

Дані проходять через сценарій наступним чином:

– створення вихідного коду компонентів ІС та маніфестів розгортання з подальшим завантаженням їх у локальний, а потім у віддалений репозиторій на GitHub;

– сценарій GitHub Actions запускає виконання автоматизованого CI\CD конвеєру – збірка, тестування коду (Bazel) та створення контейнерних образів з компонентами інформаційної системи;

– відбувається завантаження створених контейнерних образів з компонентами ІС у реєстр контейнерів, для подальшого їх розгортання у Kubernetes кластері;

– сценарій GitHub Actions автоматично оновлює запис про версію образів контейнерів з компонентами у маніфестах розгортання відповідних гілок репозиторію;

– екземпляри ArgoCD, налаштовані на відстеження змін у відповідних гілках репозиторію, синхронізують стан Kubernetes кластерів, в яких працює ІС, згідно опису в маніфестах, внаслідок чого завантажуються нові версії образів компонентів ІС з реєстру контейнерів;

– відбувається розгортання оновлених версій компонентів ІС у Kubernetes кластері згідно визначеної політики розгортання;

– система моніторингу Prometheus разом з Grafana здійснює збір та візуалізацію метрик роботи кластеру та компонентів інформаційної системи.

Таким чином, після реалізації семи кроків розробленого методу, який охоплює всі необхідні етапи – від написання коду до його тестування, збірки, розгортання та моніторингу після впровадження, досягається комплексна автоматизація CI\CD процесів для хмарних інформаційних систем за допомогою інтеграції передових DevOps інструментів та платформ.

Деталізація процесу розгортання компонентів хмарної інформаційної системи у Kubernetes кластері, який виступає для неї хмарною інфраструктурою наведено на діаграмі послідовності (рисунок 2.8),

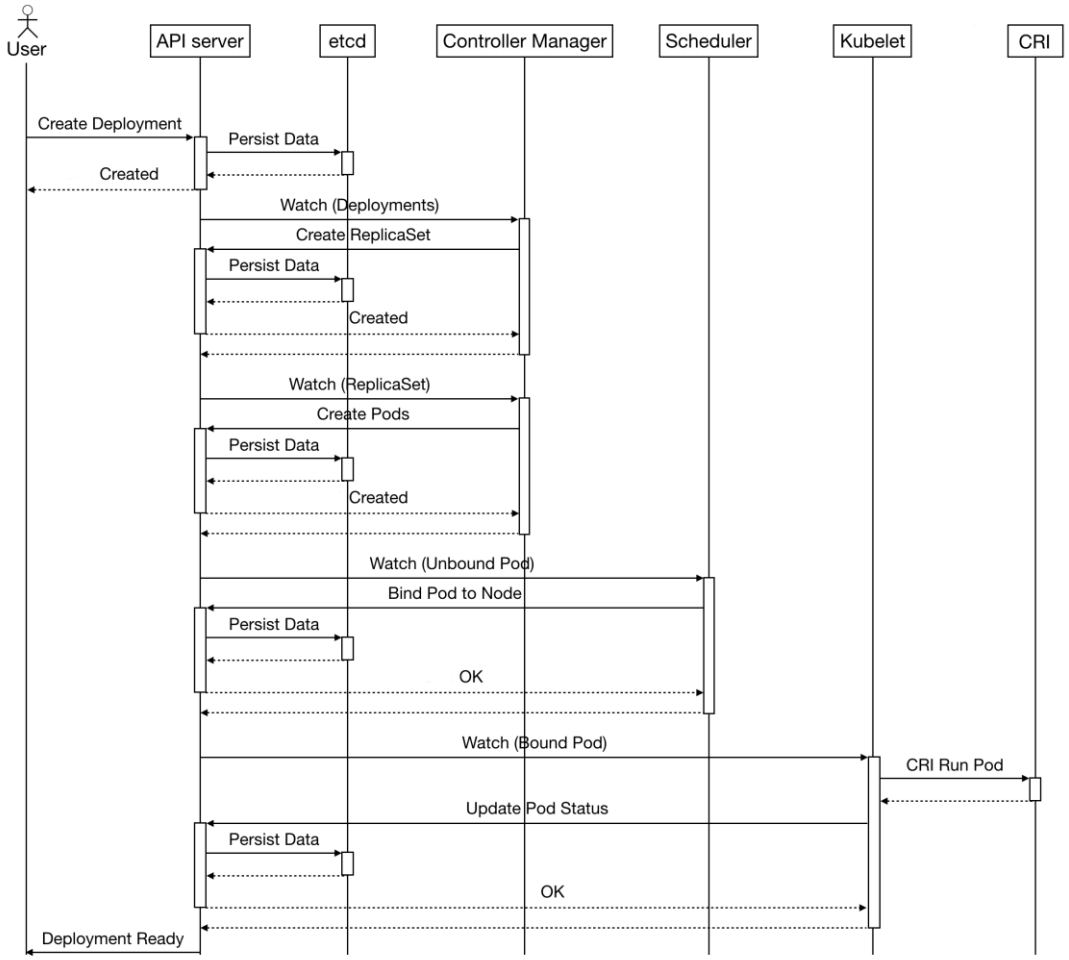

Рисунок 2.8 – Діаграма послідовності розгортання компонентів ІС у кластері

На вході Kubernetes отримує бажаний стан у вигляді об'єкту розгортання (Deployment). Далі надсилається запит на API сервер (API server) для запуску групи контейнерів (Pods) компонентів ІС на основі вказаних образів. API сервер зберігає об'єкт Deployment у сховищі ключ-значення (etcd). Після цього, Controller Manager створює об'єкт ReplicaSet для підтримки заданої кількості реплікацій подів компонентів. На основі цих даних створюються необхідні поди компонентів ІС. Планувальник (Scheduler) визначає розміщення подів на вузлах кластера, а агенти kubelet запускають контейнери розгортаючи компоненти інформаційної системи.

3 АПРОБАЦІЯ МЕТОДУ АВТОМАТИЗАЦІЇ CI\CD ПРОЦЕСІВ ДЛЯ ХМАРНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

3.1 Підготовка компонентів інфраструктури та створення необхідних облікових записів

В якості хмарної платформи для розгортання інформаційної системи обрано Google Cloud Platform (GCP). Для управління сервісами ІС у контейнерах в GCP використовується служба Google Kubernetes Engine (GKE), яка надає можливість розгортати Kubernetes кластери.

Як приклад інформаційної системи, для автоматизації CI\CD процесів якої застосовується розроблений метод, взято хмарну платформу для створення інтернет-магазинів. Така система складається з веб-додатку, який взаємодіє з базою даних та різними сервісами, необхідними для правильної роботи ІС.

Метод. Крок 1. На рисунках 3.1–3.5 продемонстровано створення необхідних облікових записів та базових інфраструктурних компонентів для подальшої автоматизації CI\CD процесів.

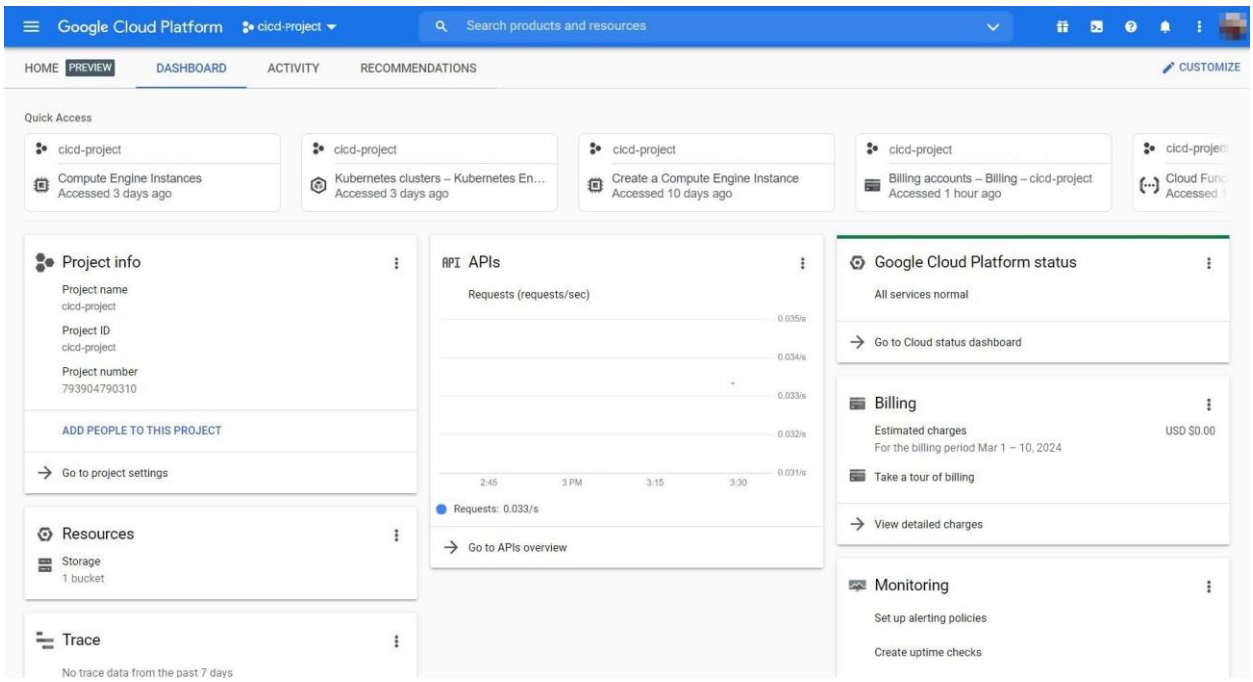

Рисунок 3.1 – Створений та налаштований обліковий запис у хмарі Google

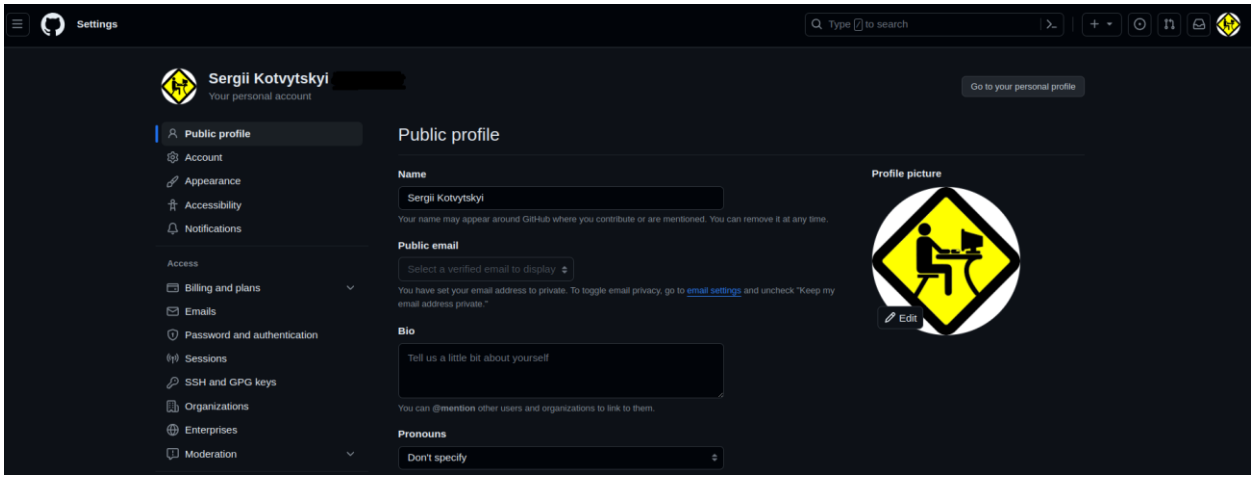

Рисунок 3.2 – Зареєстрований та налаштований акаунт у GitHub

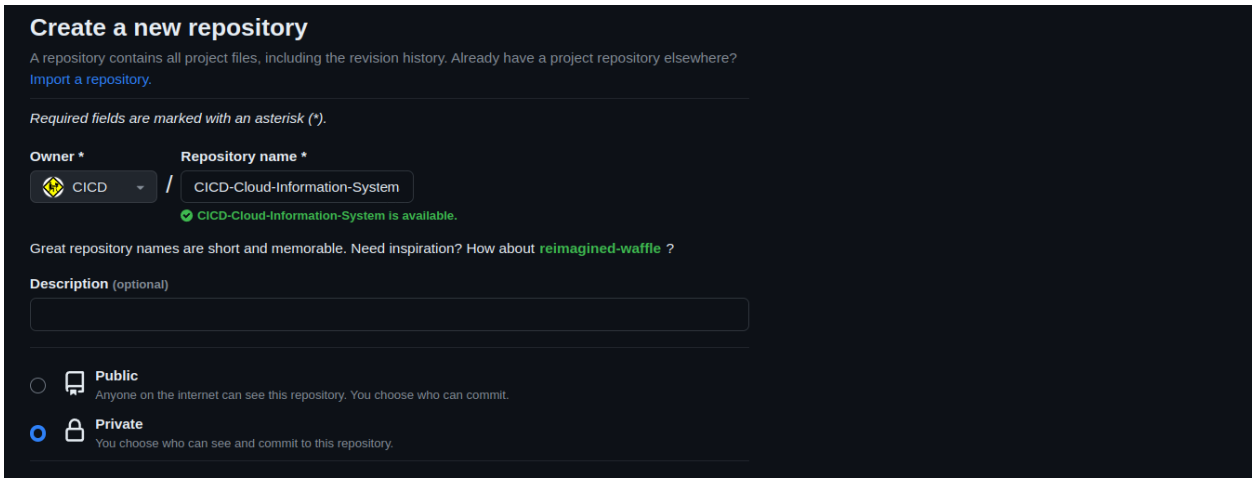

Рисунок 3.3 – Створений новий GIT-репозиторій для збереження вихідного коду, конфігурацій та маніфестів

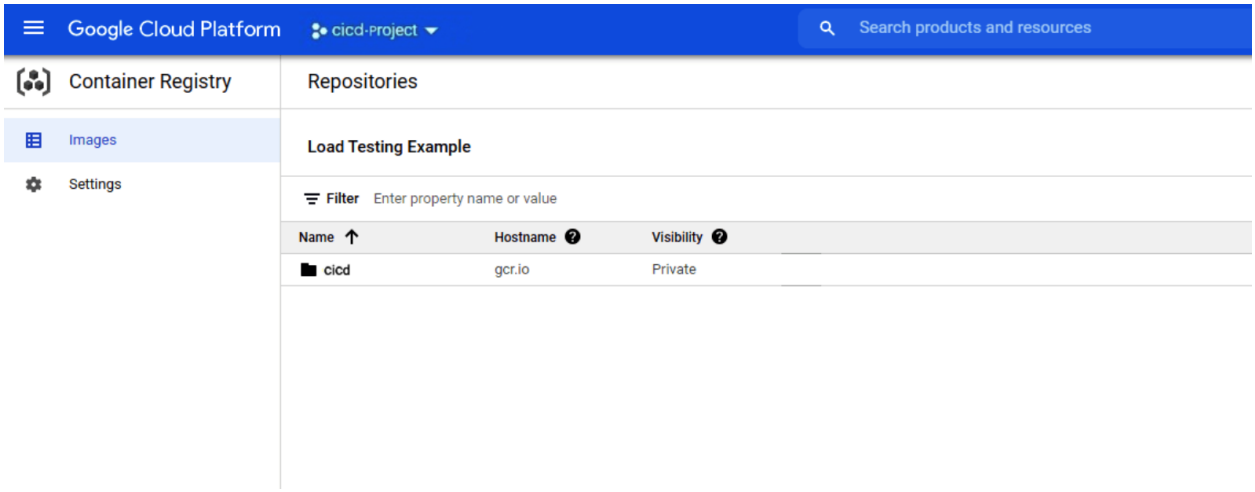

Рисунок 3.4 – Створений реєстр контейнерів у хмарі Google

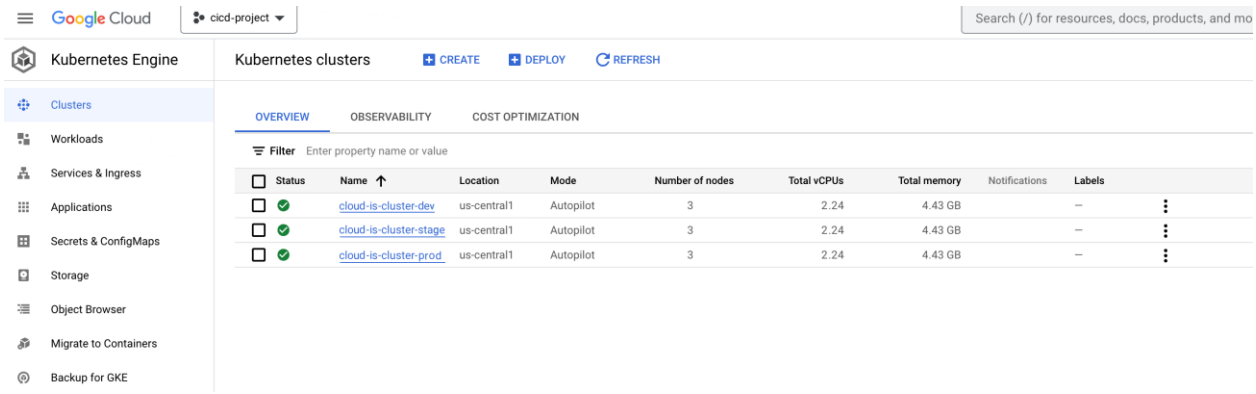

Рисунок 3.5 – Розгорнуті Kubernetes кластери у хмарі Google

У GCP було розгорнуто три окремих Kubernetes кластери для трьох середовищ функціонування інформаційної системи:

– cloud-is-cluster-dev – кластер для середовища розробки;

– cloud-is-cluster-stage – кластер для тестового середовища;

– cloud-is-cluster-prod – кластер для середовища експлуатації.

Така конфігурація дозволяє розмежувати процеси розробки, тестування та експлуатації системи. Розробники працюють у середовищі для розробки. Після тестування зміни розгортаються у текстовому середовищі. І вже перевірені оновлення системи розгортаються у середовищі експлуатації ІС.

Така стратегія дозволяє мінімізувати ризики та помилки при оновленні працюючої системи.

3.2 Налаштування засобів колективної розробки для організації CI\CD

Метод. Крок 2. Як систему керування версіями коду було обрано Git – найпопулярніший інструмент для організації командної розробки програмного забезпечення. Для розміщення віддаленого Git репозиторію, згідно кроку 1 розробленого методу використовується платформа GitHub, яка надає можливості для координації команди розробників та управління життєвим циклом ПЗ. Розроблений метод передбачає налаштування

локального середовища розробки з інтеграцією зі спільним віддаленим репозиторієм для організації ефективної командної взаємодії та контролю версій, що є ключовим для подальшої автоматизації CI\CD процесів.

На рисунках 3.6 – 3.8 наведено основні дії зі встановлення та налаштування Git, генерації SSH ключів для аутентифікації користувачів, налаштування роботи з віддаленим репозиторієм у GitHub та перевірка реплікації коду між репозиторіями.

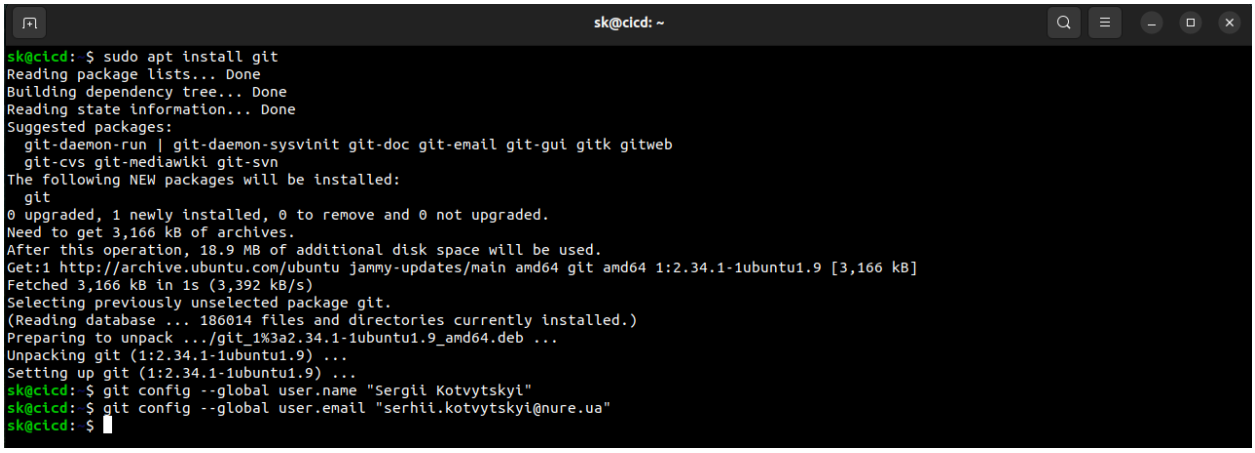

Рисунок 3.6 – Встановлений та налаштований локально Git для подальшої колективної розробки

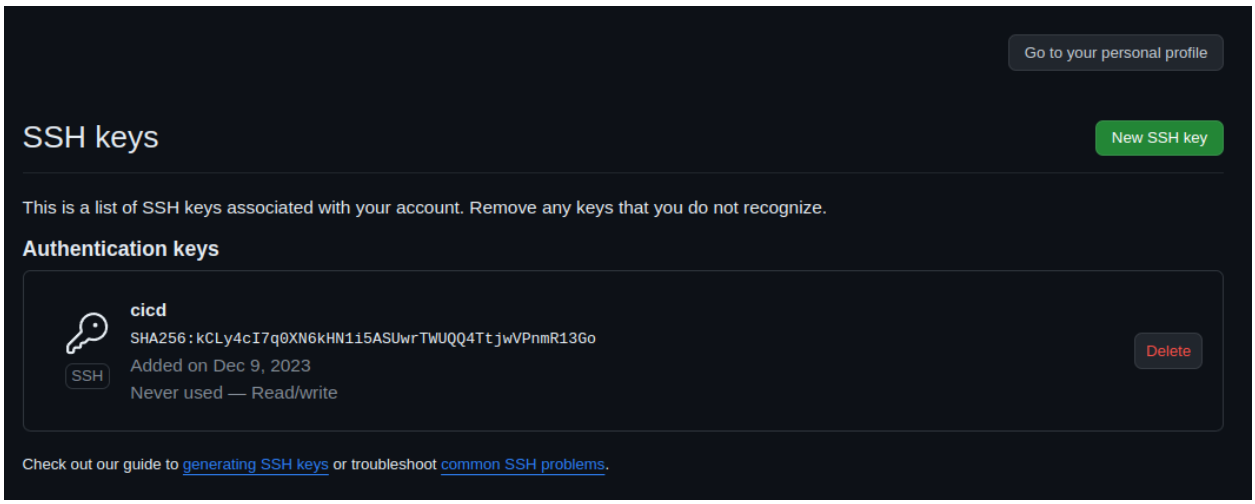

Рисунок 3.7 – Згенеровані SSH ключі для аутентифікації користувачів у GitHub

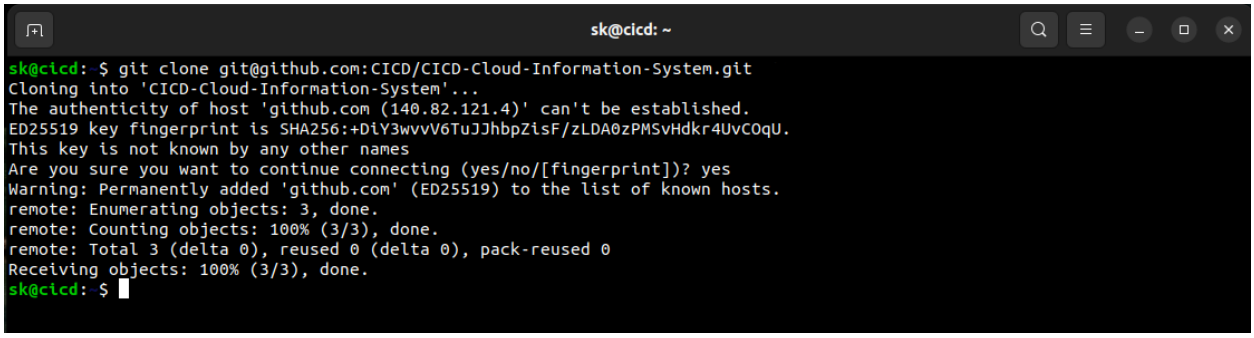

Рисунок 3.8 – Налаштований зв’язок між локальним та віддаленим репозиторіями з виконанням реплікації коду

3.3 Налаштування Bazel для автоматизації процесів збірки та тестування програмного забезпечення інформаційної системи

Метод. Крок 3. Розроблений метод автоматизації CI\CD процесів передбачає створення конфігураційних правил Bazel для автоматизації процесів тестування, збірки та упаковки коду сервісів інформаційної системи на основі їх компонентів, з налаштуванням механізму кешування проміжних результатів для подальшого їх використання.

В таблиці 3.1 представлено основні правила Bazel для автоматизації збірки та тестування програмного забезпечення обраної інформаційної системи, яке розроблене з використанням мови програмування Python та фреймворку Flask. За цими правилами визначаються залежності між компонентами системи, створення артефактів та конфігурацій, запуск процесів тестування при кожній збірці коду, а також експорт готових артефактів збірки для подальшого використання у CI\CD процесах.

Правила Bazel інтегровані безпосередньо у репозиторій основного коду, що надає можливість налаштувати автоматичне тестування та збірку лише змінених частин при кожному їх оновленні за допомогою механізму webhook у GitHub.

Таблиця 3.1 – Метод. Крок 3. Конфігурація Bazel для автоматизації збірки і тестування програмного забезпечення ІС

load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive") http_archive (name = "rules_python", build_file = "@//:third_party/python/BUILD", sha256 = "e85ae30de33625a63eca7fc40a94fea845 e641888e52f32b6beea91e8b1b2793", strip_prefix = "rules_python-0.27.1", url = "https://github.com/bazelbuild/rules_python/releases /download/0.27.1/rules_python-0.27.1.tar.gz",) load("@rules_python//python:repositories.bzl", "py_repositories") py_repositories() ========== ========== load("@rules_python//python:pip.bzl", "pip_repositories") pip_repositories() load("@rules_python//python:pip.bzl", "pip_import") pip_import(name = " thirdparty ", requirements = "//thirdparty/python:requirements.txt",) load("@thirdparty //:requirements.bzl", "pip_install") pip_install() ========== ========== load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive", "http_file") load("//toolchains/docker:toolchain.bzl", docker_toolchain_configure = "toolchain_configure",) docker_toolchain_configure(name = "docker_config", docker_path = "/usr/bin/docker",) load("//repositories:repositories.bzl", container_repositories = "repositories",) container_repositories() ========== ==========	Блок коду для опису зовнішніх залежностей: вказуються URL архіву (можна використовувати кілька дзеркал) та його sha256 хеш-сума, набір патчів та build_file, в якому описуються правила автоматизованої збірки. Блок коду для інтеграції з Python. Виконується ініціалізація репозиторіїв, імпортування Python-пакетів згідно з вимогами, визначеними у файлі requirements.txt та їх подальша інсталяція. Блок коду для інтеграції платформи управління контейнерами та визначення необхідних репозиторіїв для забезпечення доступності всіх зовнішніх ресурсів та інструментів, які використовуються у проекті	load("@rules_python//python:defs.bzl", "py_library") load("@thirdparty//:requirements.bzl", "requirement") package(default_visibility = ["//visibility:public"]) py_library(name = "online-shop", deps = [requirement("Flask"), requirement("Jinja2"), requirement("Werkzeug"), requirement("itsdangerous"), requirement("click"), requirement("MarkupSafe"),],) ========== ======== py_binary(name = "main", srcs = ["main.py"], deps = ["//requirement("online-shop")],) ========== ======== load("@rules_python//python:defs.bzl", "py_test") py_test(name = "test_services", srcs = ["test_services.py"], deps = ["//thirdparty/python:pytest", "//src/python",],) ========== ========	Блок коду для створення бібліотеки Python (py_library), яка визначає залежності для проекту. Використовується для збору залежностей разом, забезпечуючи їх доступність для інших частин проекту. Блок коду який визначає основний скрипт для старту проекту. Блок коду визначення конфігурації для автоматизації тестування, щоб перевірити коректність роботи компонентів додатку, перед його розгортанням.

3.4 Побудова автоматизованого CI\CD конвеєру на базі GitHub Actions

Метод. Крок 4. Створення завдання GitHub Actions workflow для побудови автоматизованого CI\CD конвеєру на основі попередньо налаштованих компонентів наведено на рисунку 3.9, а процес його поетапного виконання продемонстровано на рисунку 3.11.

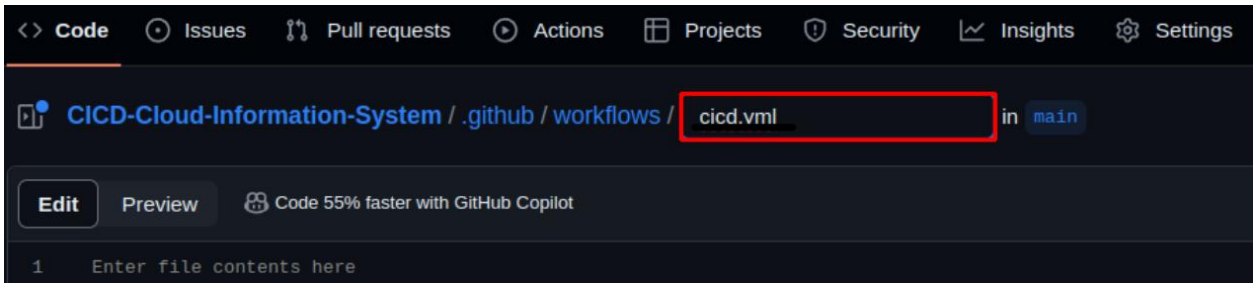

Рисунок 3.9 – Створене завдання GitHub Actions workflow

Створення змінних GitHub середовища з параметрами доступу до хмарної платформи наведено на рисунку 3.10.

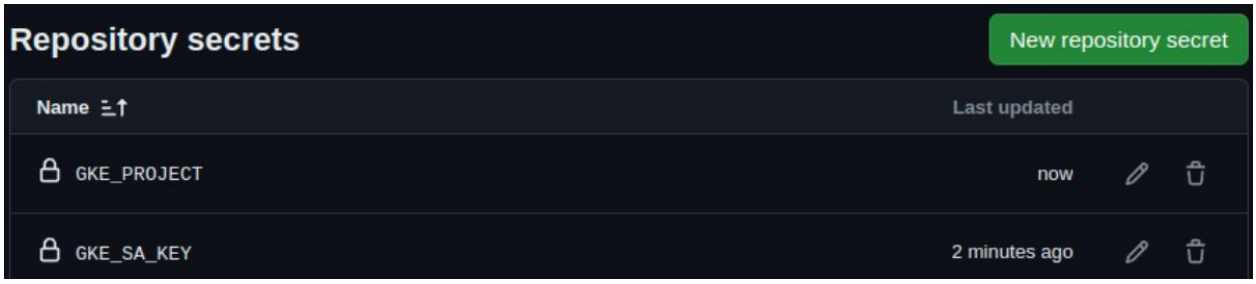

Рисунок 3.10 – Створені змінні GitHub середовища з параметрами доступу до хмарної платформи

Текст скрипту для автоматичного запуску CI\CD конвеєру при оновленні коду у репозиторії, зі збіркою контейнерних образів сервісів на основі артефактів експортованих з Bazel, призначенням образам номерів версій з подальшим їх завантаженням у реєстр контейнерів, наведено у таблиці 3.2. Тригерами запуску слугують події push та pull_request для гілки main.

Таблиця 3.2 – Метод. Крок 4. Скрипт GitHub Actions Workflow для побудови автоматизованого CI\CD конвеєру

on:
  push:
  branches: [main]
  pull_request:
  branches: [main]
========== ==========
env:
  PROJECT_ID: ${{ secrets.GKE_PROJECT }}
  GKE_CLUSTER: ${{ secrets.GKE_CLUSTER }}
  GKE_ZONE: ${{ secrets.GKE_ZONE }}
  DEPLOYMENT: ${{ secrets.DEPLOYMENT}}
  IMAGE: ${{ secrets.IMAGE }}
========== ==========
jobs:
  cicd:
  runs-on: ubuntu-latest
  environment: production
steps:
  - name: Checkout
  uses: actions/checkout@v4
  - uses: google-github-actions/setup-gcloud@1bee7de035d65ec5da40a31f8589e240eba8fde5
  with:
  service_account_key: ${{ secrets.GKE_SA_KEY }}
  project_id: ${{ secrets.GKE_PROJECT }}
========== ==========
  - name Login to Container Registry
  run gcloud --quiet auth configure-docker
  uses: google-github-actions/get-gke-credentials@db150f2cc60d1716e61922b832
  with:
  cluster_name: ${{ env.GKE_CLUSTER }}
  location: ${{ env.GKE_ZONE }}
  credentials: ${{ secrets.GKE_SA_KEY }}
========== ==========
	Визначення тригерів запуску робочого процесу. Тригери: push та pull_request для гілки main.
	
	Визначення змінних середовища, які використовуються у робочому процесі.
	
	Створення робочого процесу, під назвою cicd, який забезпечує отримання вихідного коду з репозиторію та отримання доступу до ресурсів у хмарі GCP, за рахунок токену, що зберігається як секрет GitHub.
	
	Налаштування доступу Docker у реєстр контейнерів (GCR) та отримання облікових даних для взаємодії з Kubernetes кластером (GKE) у хмарі Google.
		- name: Bazel
		  id: bazel-cache
		  uses: actions/cache@v2
		  env:
		  version: ${{ vars.BASEL_VERSION }}
		  with:
		  path: | ~/.cache/bazel
		  key: ${{ runner.os }}-{{ env.version }}
		  -bazel-cache
		  run: bazel build
		  run: bazel test
		========== ==========
		- name: Build
		  run: |-
		  docker build \
		--tag "$PROJECT_ID/$IMAGE:$GITHUB_SHA" \
		--build-arg GITHUB_SHA="$GITHUB_SHA" \
		--build-arg GITHUB_REF="$GITHUB_REF"\
		========== ==========
		- name: Delivery
		  run: |-
		  docker push "$PROJECT_ID/$IMAGE:$GITHUB_SHA"
		  run: |-
		  curl -sfLo kustomize https://github.com/kubernetes-sigs/kustomize/releases/download/v3.1.0/kustomize_3.1.0_linux_amd64
		  chmod u+x ./kustomize
		  run: |-
		  ./kustomize edit set image PROJECT_ID/IMAGE:TAG=$PROJECT_ID/$IMAGE:$GITHUB_SHA
		  ./kustomize build .
		========== ==========
			Налаштування кешу для повторного використання проміжних результатів, збірка та тестування артефактів за допомогою Bazel
			
			Автоматизована збірка Docker контейнерних образів з сервісами та присвоює їм унікальних тегів.
			
			Завантаження створених контейнерних образів сервісів в реєстр GCR, встановлення утиліти kustomize, яка виконує автоматичне оновлення записів про версії сервісів у маніфестах розгортання.

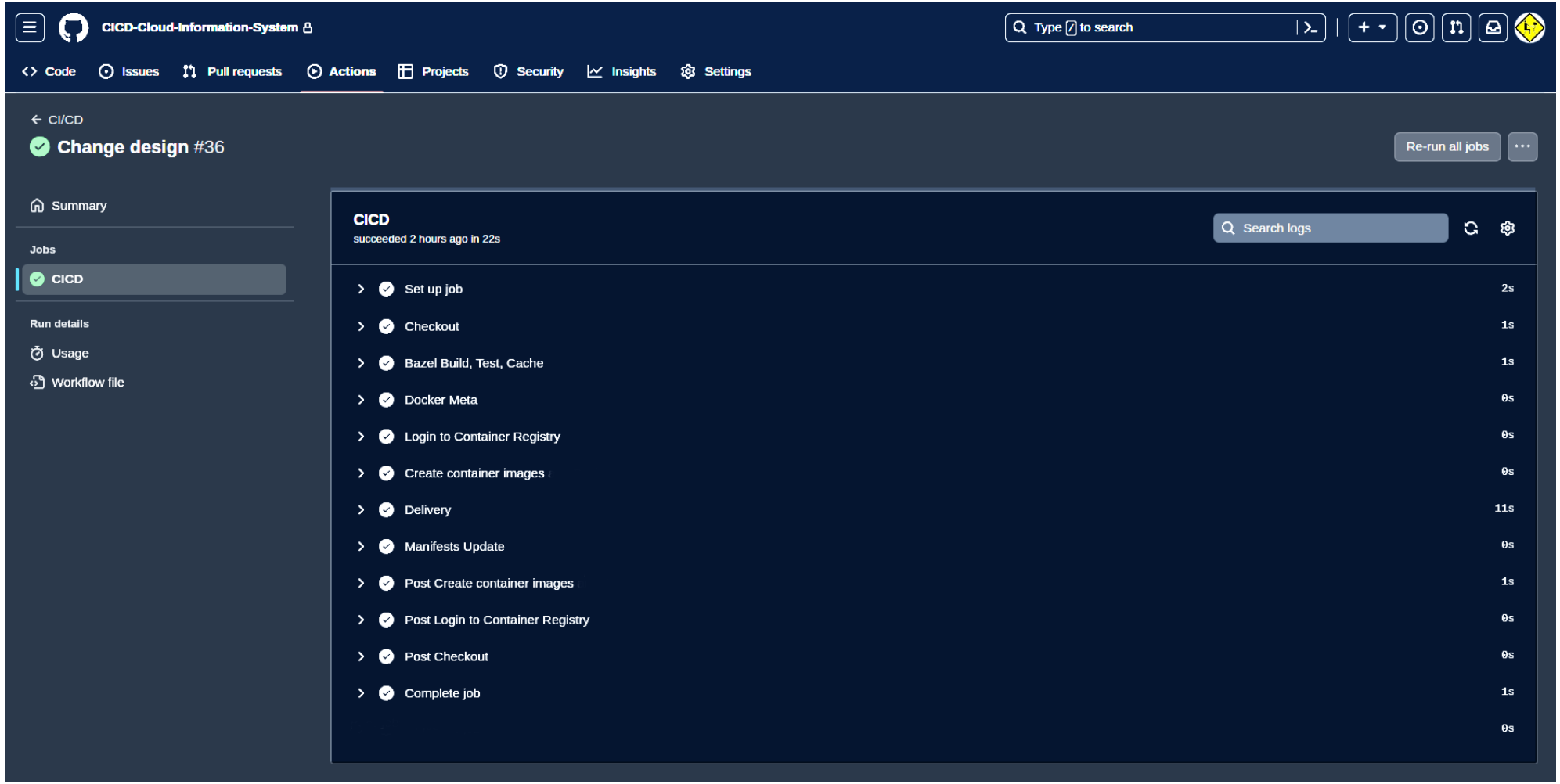

Рисунок 3.11 – Виконане завдання GitHub Actions Workflow методу автоматизації CI\CD процесів для хмарних інформаційних систем

3.5 Встановлення та налаштування системи розгортання ArgoCD

Метод. Крок 5. Розгортання системи ArgoCD у Kubernetes кластері для автоматизації доставки оновлень інформаційної системи у хмарне середовище наведено на рисунку 3.12.

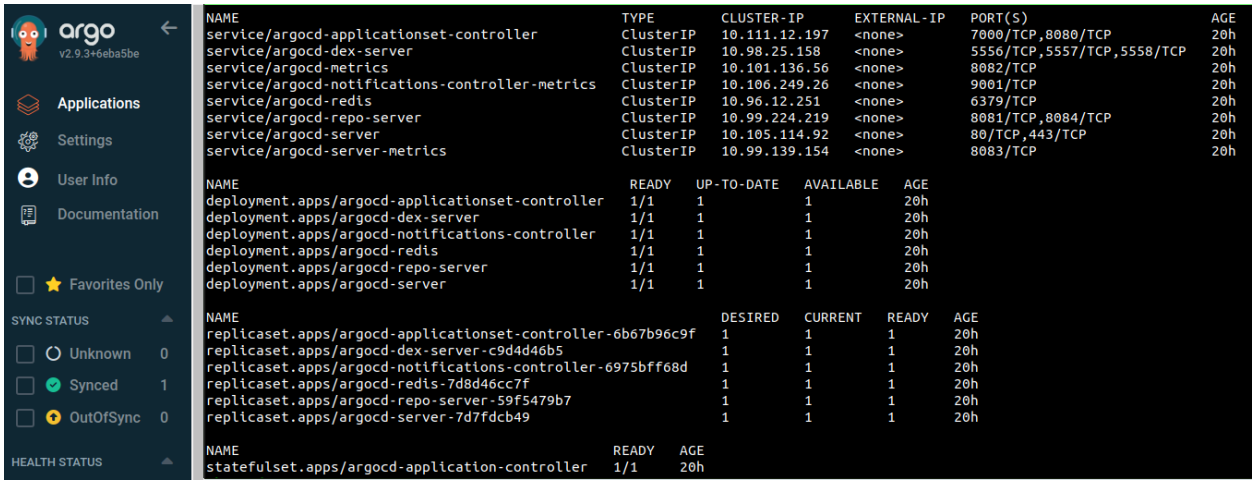

Рисунок 3.12 – Розгорнута система ArgoCD у Kubernetes кластері

Налаштування окремого екземпляру ArgoCD на відстеження змін у відповідній гілці репозиторію для середовища в якому експлуатується інформаційна система наведено на рисунку 3.13. Текст маніфестів конфігурації ArgoCD представлено у таблиці 3.3.

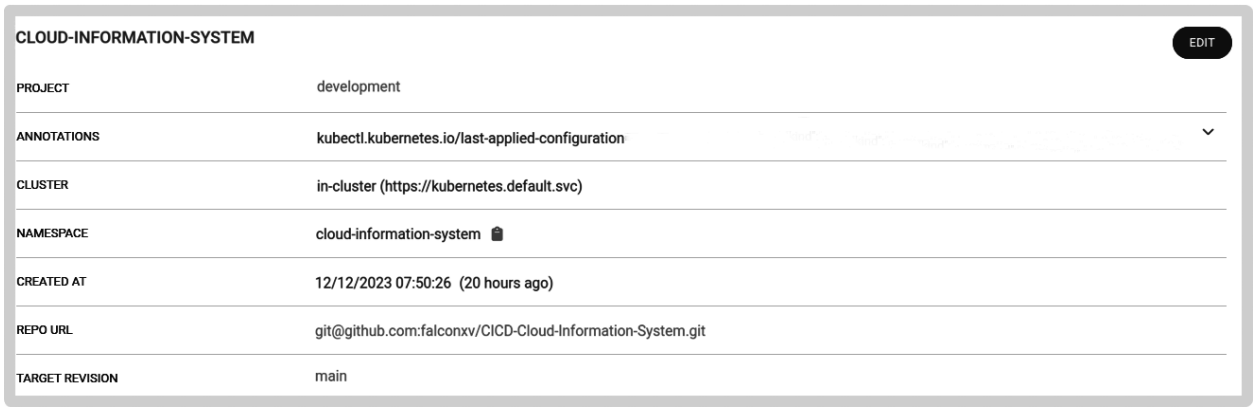

Рисунок 3.13 – Налаштований ArgoCD на відстеження змін у гілці main

Таблиця 3.3 – Метод. Крок 5. Маніфести ArgoCD для налаштування автоматизації доставки оновлень ІС

apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: cloud-information-system
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io
spec:
  project: development
  source:
  repoURL: git@github.com:falconxv/
  CICD-Cloud- Information-System.git
  targetRevision: main
  path: kubernetes/
destination:
  server: https://kubernetes.default.svc
  namespace: cloud-information-system
syncPolicy:
  automated:
  prune: true
  selfHeal: true
  syncOptions:
  - CreateNamespace=true
	Блок коду ArgoCD для автоматизації процесу безперервної доставки оновлень інформаційної системи у Kubernetes. Вказується джерело файлів маніфестів розгортання у вигляді Git репозиторію та задається політика автоматичної синхронізації з цим репозиторієм. Коли зміни коду потраплять у задану гілку, Argo CD самостійно завантажить оновлені маніфести та застосує їх у вказаному Kubernetes кластері і просторі імен.	apiVersion: argoproj.io/v1alpha1
		kind: AppProject
		metadata:
		  name: development
		  finalizers:
		  - resources-finalizer.argocd.argoproj.io
		spec:
		  description: Project containing development environment services
		  sourceRepos: '*'
		  destinations:
		  - namespace: '*'
		  server: '*'
		  clusterResourceWhitelist:
		  - group: '*'
		  kind: '*'
			Маніфест опису проекту ArgoCD для розгортання додатків ІС з наданням прав на використання різних репозиторіїв та будь-яких кластерних ресурсів Kubernetes.

apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
  namespace: argocd
  labels:
  app.kubernetes.io/name: argocd-cm
  app.kubernetes.io/part-of: argocd
data:
  timeout.reconciliation: 30s
	Блок коду для визначення відведеного часу операціям приведення поточного стану кластеру до відповідності стану, описаному в маніфестах у Git репозиторії.	apiVersion: argoproj.io/v1alpha1
		kind: AppProject
		metadata:
		  name: infrastructure
		  namespace: argocd
		  finalizers:
		  - resources-finalizer.argocd.argoproj.io
		spec:
		  description: Project with infrastructure related applications
		  sourceRepos: '*'
		  destinations:
		  - namespace: '*'
		  server: '*'
		  clusterResourceWhitelist:
		  - group: '*'
		  kind: '*'
			Маніфест опису проекту ArgoCD для розгортання інфраструктурних додатків системи, з наданням прав на використання різних репозиторіїв та будь-яких кластерних ресурсів Kubernetes.

3.6 Розробка маніфестів для розгортання сервісів у Kubernetes кластері

Метод. Крок 6. Текст розроблених маніфестів для опису розгортання сервісів хмарної інформаційної системи у Kubernetes кластері наведено у таблиці 3.4, а саму розгорнуту інформаційну систему на рисунку 3.15.

На рисунку 3.14 зображено структуру гілок репозиторію для зберігання YAML-файлів середовища розробників, тестового та експлуатаційного.

Така організація дозволяє системі ArgoCD відстежувати зміни конфігурацій в конкретних гілках розробки та автоматично розгортати їх у відповідні середовища експлуатації згідно налаштованої політики синхронізації.

Розмежування конфігурацій по середовищах спрощує адміністрування та дає можливість гнучкого керування процесом доставки змін в хмарну інфраструктуру.

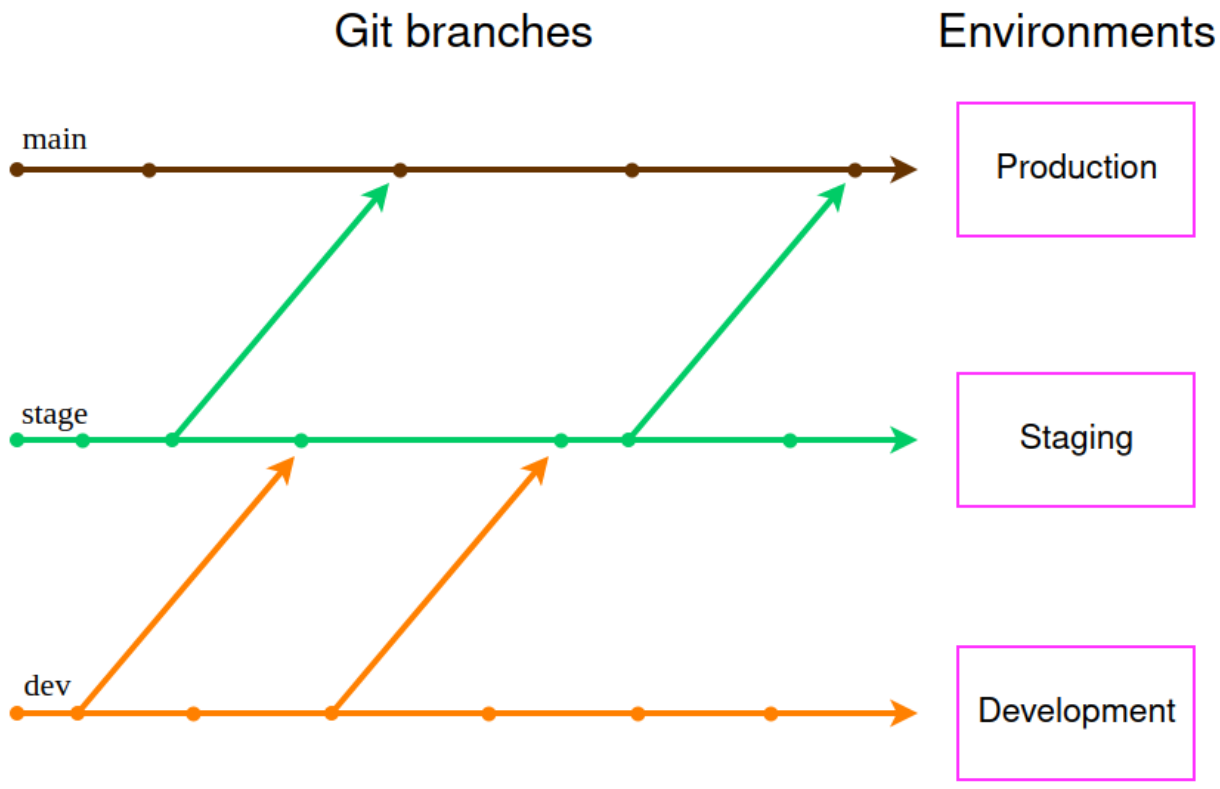

Рисунок 3.14 – Структура гілок репозиторію для зберігання YAML-файлів

Таблиця 3.4 – Метод. Крок 6. Маніфести для налаштування розгортання сервісів ІС у Kubernetes кластері

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: cloud-is-ingress
  namespace: cloud-information-system
  annotations:
  kubernetes.io/ingress.class: nginx
  nginx.ingress.kubernetes.io/ssl-redirect: "false"
  nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - host: cloud.information.systems
  http:
  paths:
  - path: /
  pathType: Prefix
  backend:
  service:
  name: cloud-is-service
  port:
  number: 80
	Маніфест для опису ресурсу ingress, котрий виконує роль точки входу трафіку до всіх сервісів інформаційної системи із зовнішньої мережі. Налаштовує правила маршрутизації запитів на відповідні сервіси всередині кластера по заданим шляхам URI та портам. Таким чином ingress забезпечує єдиний вхід до компонентів ІС за допомогою описаних правил маршрутизації запитів.	apiVersion: apps/v1
		kind: Deployment
		metadata:
		  name: cloud-is-deployment
		  labels:
		  app: cis
		spec:
		  replicas: 5
		  selector:
		  matchLabels:
		  app: http-server
		  template:
		  metadata:
		  labels:
		  app: http-server
		  spec:
		  containers:
		  - name: cloud-is
		  image: falconxv/cloud-information-system:latest
		  ports:
		  - containerPort: 5000
			Маніфест для опису розгортання групи контейнерів з окремим сервісом інформаційної системи. Запускається вказана у параметрі replicas кількість контейнерів з цим сервісом за вказаним у параметрі image образом в реєстрі контейнерів. Платформа Kubernetes забезпечує розгортання компонентів сервісу та їх самовідновлення після збоїв.

apiVersion: v1
kind: Service
metadata:
  name: cloud-is-service
spec:
  selector:
  app: http-server
  ports:
  - protocol: TCP
  port: 80
  targetPort: 5000
  type: LoadBalancer
	Маніфест для опису ресурсу Service, котрий забезпечує внутрішню взаємодію між компонентами ІС в кластері та надає зовнішній доступ до окремого сервісу чи його компонентів.	apiVersion: v1
		kind: Pod
		metadata:
		  name: cis
		spec:
		  containers:
		  - name: cloud-is
		  image: falconxv/cloud-information-system:latest
		  ports:
		  - containerPort: 5000
			Маніфест для опису поду, найпростішого об’єкту у кластері, який представляє собою групу контейнерів. Містить контейнери та всі налаштування потрібні для його роботи.

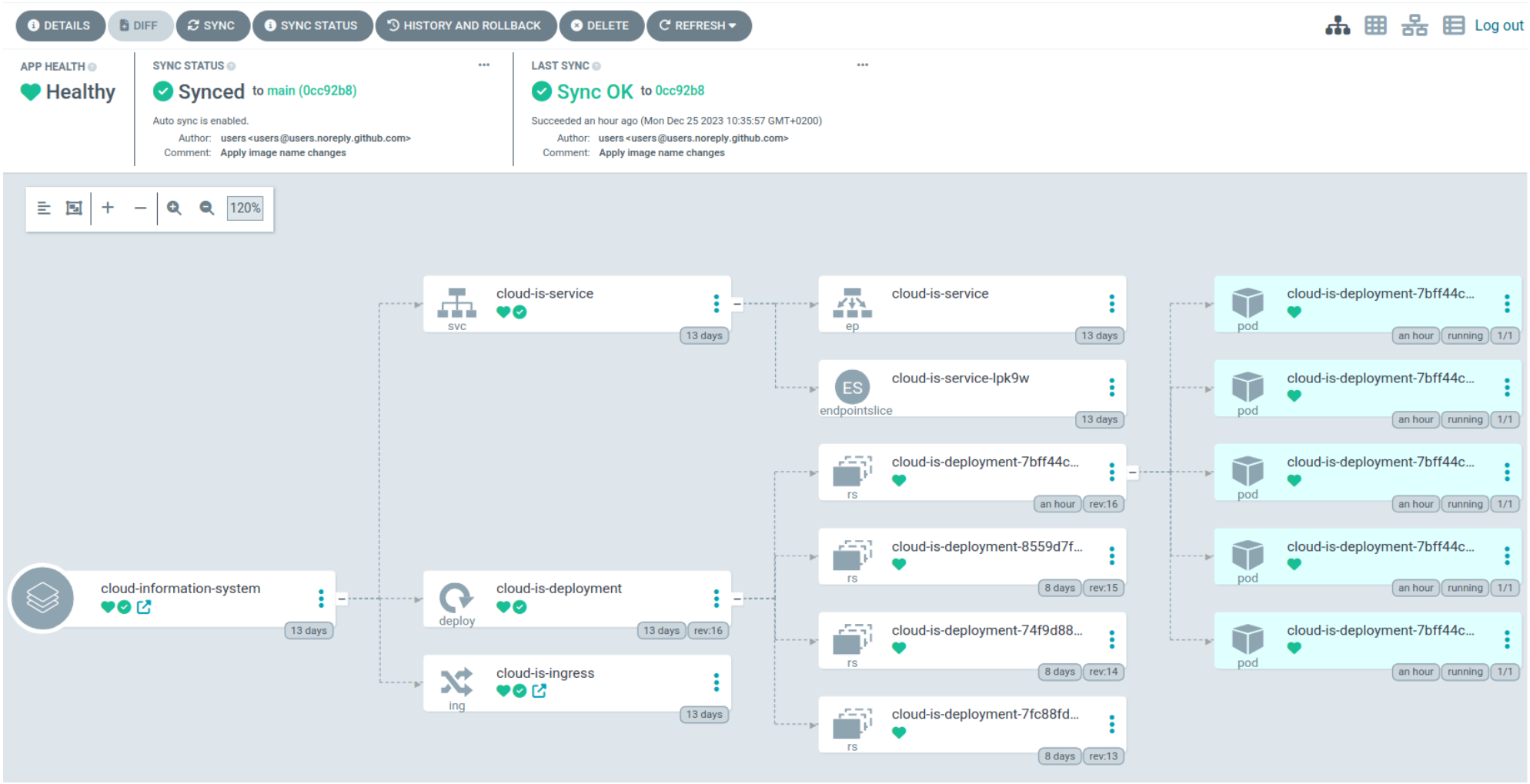

Рисунок 3.15 – Розгорнута хмарна інформаційна система у Kubernetes кластері

3.7 Налаштування системи моніторингу на базі Prometheus та Grafana

Метод. Крок 7. Для реалізації моніторингу роботи Kubernetes кластеру та розгорнутої в ньому ІС було використано стек інструментів Prometheus та Grafana. Їх схему взаємодії наведено на рисунку 3.16.

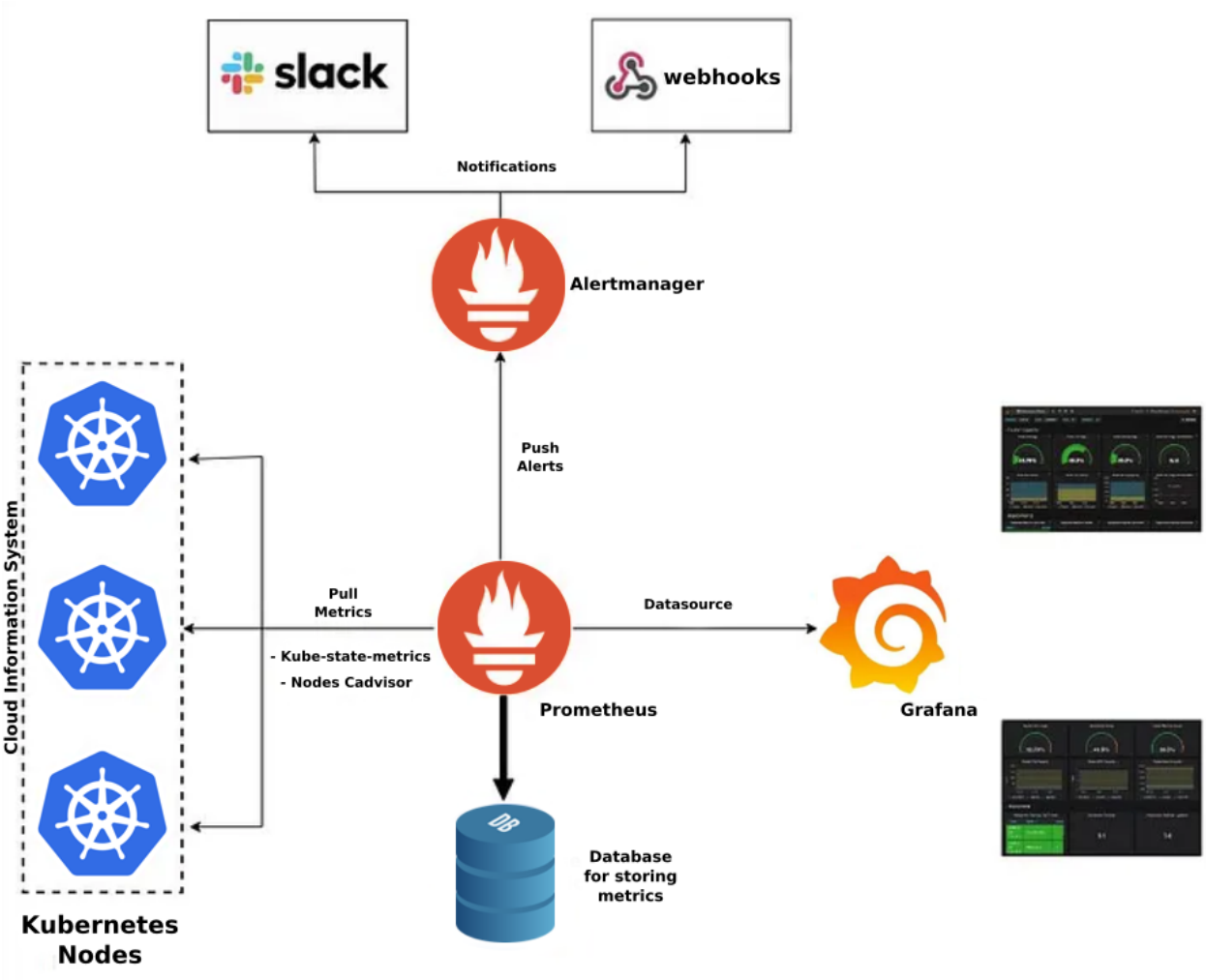

Рисунок 3.16 – Схема взаємодії Prometheus і Grafana

Prometheus відповідає за збір метрик з заданих джерел. Для цього в Kubernetes кластері були розгорнуті експортери метрик node_exporter та cAdvisor, які забезпечують збір показників роботи вузлів та контейнерів відповідно. Також було активовано збір метрик Kubernetes через вбудований API сервер. Зібрані метрики зберігаються у базі даних.

Для візуалізації цих даних використовується Grafana, яка отримує їх напряму через Prometheus API. Інформаційну панель для відображення даних про продуктивність кластера, використання ресурсів, роботу сервісів ІС тощо, наведено на рисунку 3.17.

Рисунок 3.17 – Інформаційна панель моніторингу кластеру та системи

Для встановлення стеку моніторингу був використаний інструмент helm – пакетний менеджер для розгортання сервісів безпосередньо у Kubernetes кластер. За допомогою helm та його репозиторіїв (charts) Prometheus та Grafana були швидко розгорнуті з усіма залежностями та налаштуваннями для збору метрик та їх візуалізації відповідно. Встановлення Prometheus та Grafana за допомогою helm показано на рисунку 3.18.

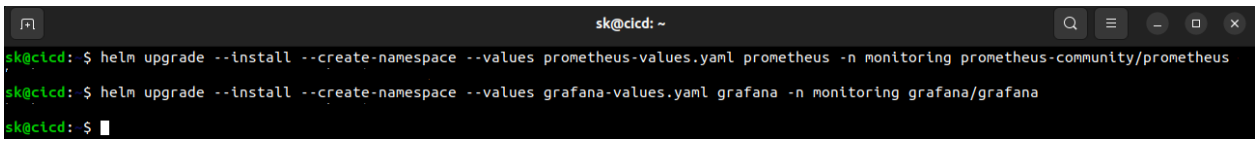

Рисунок 3.18 – Встановлення Prometheus та Grafana за допомогою пакетного менеджера helm

ВИСНОВКИ

У процесі дослідження в рамках дипломної роботи було проаналізовано існуючі підходи до автоматизації CI\CD процесів для хмарних інформаційних систем. Зокрема, досліджено актуальні методи та найбільш поширені технології їх реалізації. Це дозволило виявити ключові недоліки та напрямки для вдосконалення наявних рішень. На основі проведеного аналізу було сформовано уніфіковані вимоги до процесів автоматизації з урахуванням потреб основних груп користувачів. Отримані в ході дослідження результати дозволили сформувати нову комплексну методику автоматизації CI\CD процесів для хмарних інформаційних систем. Звіт складено згідно [19-21].

До основних результатів дослідження відносяться:

– виконано аналіз проблем автоматизації CI\CD процесів для хмарних інформаційних систем;

– проведено дослідження провідних методів та технологій побудови CI\CD процесів для хмарних інформаційних систем;

– визначено структуру автоматизації CI\CD процесів для хмарних інформаційних систем;

– сформовані рольові та загальні вимоги автоматизації CI\CD процесів для хмарних інформаційних систем;

– розроблено метод автоматизації CI\CD процесів для хмарних інформаційних систем;

– виконано апробацію розробленого методу автоматизації CI\CD процесів для хмарних інформаційних систем.

Робота містить вирішення актуальної науково-практичної проблеми автоматизації CI\CD процесів для сучасних хмаро-орієнтованих інформаційних систем. Подальші дослідження можуть бути спрямовані на розширення функціональності розробленого методу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation / J. Humble, D. Farley. - Addison-Wesley Professional, 2010. – 512 p.

2. Kim, G., Debois, P., Willis, J., & Humble, J. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution Press, 2016. – 480 p.

3. Erich, M., Amrit, C., Duvall, P. Continuous Delivery: Patterns and Practices in the Cloud. Addison-Wesley, 2021. – 224 p.

4. Nayan B. Ruparelia, Cloud Computing, revised and updated edition.The MIT Press Essential Knowledge series, 2023. – 304 p.

5. Shivakumar R.Goniwada, Cloud-Native Application Architecture: Microservice Development Best Practice. Springer, 2023. – 399 p.

6. Blokdyk, G., Containers and Microservices Second Edition. 5STARCooks, 2021. – 311 p.

7. Erich, M., Amrit, C., Duvall, P., Continuous Delivery: Patterns and Practices in the Cloud. Addison-Wesley, 2021. – 224 p.

8. Chacon, S., & Straub, B., Pro Git. Apress, 2014. – 440 p.

9. GitHub documentation. URL: https://docs.github.com/ (дата звернення: 25.11.2023).

10. GitHub Actions documentation. URL: https://docs.github.com/en/actions (дата звернення: 26.11.2023).

11. ArgoCD documentation. URL https://argo-cd.readthedocs.io/ (дата звернення: 28.11.2023).

12. Docker Swarm documentation. URL: https://docs.docker.com/ engine/swarm/ (дата звернення: 29.11.2023).

13. Nomad documentation. URL: https://developer.hashicorp.com/ nomad/docs (дата звернення: 30.11.2023).

14. Kubernetes documentation. URL: https://kubernetes.io/docs/home/ (дата звернення: 01.12.2023).

15. Prometheus documentation. URL: https://prometheus.io/docs/ (дата звернення: 02.12.2023).

16. Terraform documentation. URL: https:// /developer.hashicorp.com/ /terraform/docs (дата звернення: 02.12.2023).

17. Rumbaugh, J., Jacobson, I., Booch, G., The Unified Modeling Language Reference Manual. Addison-Wesley, 2nd Edition, 2004. – 742 p.

18. Cohn, M., User Stories Applied: For Agile Software Development, 1st Edition. Addison-Wesley Professional, 2004. – 304 p.

19. Методичні вказівки щодо розробки та оформлення кваліфікаційної роботи (для студентів усіх форм навчання другого (магістерського) рівня програми "Інформаційні управляючі системи та технології") / Упоряд.:Петров К.Е., Левикін В.М., Чалий С.Ф., Євланов М.В., Саєнко В.І., Міхнов Д.К., Міхнова А.В., Чала О.В. – Харків: ХНУРЕ, 2021. – 30с.

20. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлювання. – Чинний від 22.06.2015. – Київ: ДП «УкрНДНЦ», 2016. – 31 с.

21. ДСТУ 8302:2015. Інформація та документація. Бібліографічні посилання. Загальні положення та правила складаня. – Чинний від 04.03.2016. – Київ: ДП «УкрНДНЦ», 2016. – 20 с.