

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет прикладної математики

**Кафедра системного програмування і
спеціалізованих комп'ютерних систем**

«На правах рукопису»
УДК 004.77

До захисту допущено:

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«__» _____ 2024 р.

Магістерська дисертація

**на здобуття ступеня магістра
за освітньо-науковою програмою «Системне програмування та
спеціалізовані комп'ютерні систем»
зі спеціальності 123 «Комп'ютерна інженерія»
на тему: «Метод автоматизації рутинних завдань DevOps»**

Виконав:

студент II курсу, групи KB-21 мн

Гончаров Влас Сергійович

Науковий керівник:

Професор кафедри СПіСКС, докт. техн. наук, професор

Зайцев Володимир Григорович

Рецензент:

Засвідчую, що у цій магістерській
дисертації немає запозичень з
праць інших авторів без
відповідних посилань.

Студент _____

Київ – 2024 року

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Факультет прикладної математики

**Кафедра системного програмування і спеціалізованих
комп'ютерних систем**

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«__» _____ 2022 р.

**ЗАВДАННЯ
на магістерську дисертацію студенту**

Гончарову Власу Сергійовичу

1. Тема дисертації «Метод автоматизації рутинних завдань DevOps», науковий керівник дисертації Зайцев Володимир Григорович докт. техн. наук, професор, затверджені наказом по університету від «27» березня 2024 р. №1478-С

2. Термін подання студентом дисертації «5» червня 2024 р.

3. Об'єкт дослідження: розробка методу автоматизації рутинних завдань DevOps.

4. Вихідні дані: налаштування автоматизації рутинних завдань DevOps.

5. Перелік завдань, які потрібно розробити:

- Проведення аналізу профільної літератури та вже існуючих рішень: ознайомитись з основами DevOps практик та способами автоматизації рутинних завдань; переглянути існуючі методи та інструменти до них; проаналізувати успішне використання DevOps практик.
- Визначення цілей автоматизації рутинних завдань: типові завдання, які можуть бути автоматизовані; критерії оцінки ефективності.
- Розробка методики автоматизації: розробка методики для безпосередньої автоматизації; вибір оптимальних інструментів; опис всього процесу впровадження DevOps практик.

- Реалізація автоматизації: написання скриптів на налаштування відповідних програм під потреби проекту; проведення тестів для визначення переваг використання DevOps практик.
 - Аналіз ефективності: збір даних про ефективність автоматизованих процесів; порівняння ефективності системи *до* та *після* впровадження; визначення переваг та недоліків такого підходу.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу: презентація.
7. Орієнтовний перелік публікацій:
- Теза доповіді “Автоматизація рутинних завдань DevOps для посилення захисту інформації в телекомунікаційних системах”.
8. Дата видачі завдання «1» листопада 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення основних принципів DevOps. Аналіз літератури за темою магістерської дисертації.	11.12.2022	
2.	Розробка та узгодження завдання із керівником.	23.03.2023	
3.	Робота над першим розділом магістерської дисертації.	02.06.2023	
4.	Розробка та тестування програмного забезпечення, що реалізує головну ідею магістерської дисертації.	13.09.2023	
5.	Робота над другим розділом, що включає в себе опис розробки програмного забезпечення.	08.12.2023	
6.	Тестування ефективності впровадженого методу автоматизації рутинних завдань.	11.01.2024	
7.	Робота над третім розділом магістерської дисертації.	10.03.2024	
8.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу.	12.05.2024	
9.	Попередній розгляд магістерської дисертації на кафедрі.	28.05.2024	

Студент

Влас ГОНЧАРОВ

Науковий керівник

Володимир ЗАЙЦЕВ

РЕФЕРАТ

Актуальність теми обумовлена зростаючою складністю програмного забезпечення, необхідністю у швидкій та якісній розробці, регулярних стабільних оновлень та можливості швидко повернутися до попередньої версії продукту. Автоматизація рутинних завдань розробника знижує ризик помилок та може підвищити продуктивність команди розробників, що робить DevOps процеси більш ефективними. Ця робота спрямована на дослідження та практику використання методики DevOps, що є критично важливим для забезпечення конкурентоспроможності сучасних ІТ-компаній на ринку.

Об'єктом дослідження є процес розробки та налаштування виробничого процесу із використанням методології DevOps, що містить такі аспекти, як розробка, тестування та розгортання програмного забезпечення. Особлива увага приділена рутинним процесам, що можуть бути автоматизовані. Швидку та стабільну роботу команди розробників може забезпечити безперервна інтеграція, неперервне розгортання, автоматизоване тестування та моніторинг стану системи.

Предмет дослідження полягає у виборі інструментів автоматизації рутинних завдань в DevOps процесах та відповідних методів їхнього використання. Особлива увага приділена визначенню ефективності розробки програмного забезпечення із використанням DevOps практик та без них.

Мета роботи полягає в тому, щоб дослідити рішення, що існують та запропонувати метод автоматизації рутинних завдань DevOps для поліпшення виробничого процесу команд розробників. Робота має продемонструвати, що використання DevOps практик не є примхою для команди, а є необхідністю для якісного конкурентоспроможного програмного забезпечення.

Методи дослідження. В даному дослідженні можна побачити використання таких методів, як аналіз, моделювання, вимірювання та порівняння.

Наукова новизна полягає в розробці методу автоматизації рутинних завдань у DevOps процесах, що має призвести до підвищення ефективності та якості розробки.

Практична цінність. Ця робота дає можливість використовувати розроблений метод автоматизації рутинних завдань в DevOps процесах для того, щоб підвищити ефективність процесу розробки шляхом вивільнення більшої кількості часу на більш важливі питання, вища кваліфікація виконуваних дій та наявності високої гнучкості виробничого процесу. Все це може надати можливість швидко, якісно та стабільно розробляти продукт, щоб підвищити рівень конкурентоспроможності компанії.

Апробація роботи. Основні положення були викладені та обговорювались на XVII Міжнародній науково-практичній конференції «ІНТЕГРОВАНІ ІНТЕЛЕКТУАЛЬНІ РОБОТОТЕХНІЧНІ КОМПЛЕКСИ» ПРТК-2024 (м. Київ, 21-22 травня 2024 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, 3-х розділів та висновків.

У вступі представлено загальну характеристику роботи, проведена оцінка сучасного стану даної проблеми, обґрунтовано актуальність цього дослідження, визначено мету та завдання роботи, продемонстровано наукову новизну та практичну цінність отриманих результатів.

Перший розділ: розгляд теоретичної інформації (основні принципи, терміни та інструменти) та вже існуючих методів використання DevOps практик; порівняльний аналіз DevOps практик та методологій-попередників.

Другий розділ: вибір оптимальних інструментів DevOps для виконання практичної складової; виконана підготовка (дослідження способу впровадження безперервної інтеграції (CI), неперервного розгортання (CD), автоматизованого тестування та моніторингу системи) до реалізації практик DevOps в третьому розділі.

Третій розділ: імплементація DevOps практик для автоматизації рутинних завдань під час розробки програмного забезпечення; збір інформації про роботу новоствореної системи задля подальшого аналізу.

Висновки: узагальнення отриманих результатів дослідження; формулювання висновків стосовно доцільності та необхідності використання DevOps практик в сучасному виробничому процесі; визначення можливих векторів розвитку цього підходу в найближчому майбутньому.

Робота виконана на 101 аркуші, містить 3 додатки та посилання на список використаних літературних джерел з 26 найменувань. У роботі наведено 46 рисунків та 4 таблиці.

Ключові слова: DevOps практики, CI/CD, моніторинг, Docker, інструменти автоматизації, Google Cloud Platform.

ABSTRACT

The relevance of the topic is due to the growing complexity of software, the need for fast and high-quality development, regular stable updates, and the ability to quickly return to the previous version of the product. Automating routine developer tasks reduces the risk of errors and can increase the productivity of the development team, making DevOps processes more efficient. This work is aimed at researching and practicing the use of the DevOps methodology, which is critical to ensuring the competitiveness of modern IT companies in the market.

The object of the study is the process of developing and customizing the production process using the DevOps methodology, which includes such aspects as software development, testing and deployment. Particular attention is paid to routine processes that can be automated. The fast and stable work of the development team can be ensured by continuous integration, continuous deployment, automated testing, and system health monitoring.

The subject of the research is the selection of tools for automating routine tasks in DevOps processes and appropriate methods for their use. Particular attention is paid to determining the efficiency of software development with and without DevOps practices.

The aim of the work is to investigate existing solutions and develop a method for automating routine DevOps tasks to improve the production process of development teams. The work should demonstrate that the use of DevOps practices is not a whim for the team, but a necessity for high-quality competitive software.

Research methods. In this study, you can see the use of methods such as analysis, modeling, measurement, and comparison.

The scientific novelty lies in the development of a method for automating routine tasks in DevOps processes, which should lead to increased efficiency and quality of development.

Practical value. This work makes it possible to use the developed method of automating routine tasks in DevOps processes in order to increase the efficiency of the development process by freeing up more time for more important issues, higher qualification of the actions performed, and the availability of high flexibility of the production process. All of this can enable fast, high-quality and stable product development to increase the company's competitiveness.

Approbation of the work. The main provisions were presented and discussed at the XVII International Scientific and Practical Conference “INTEGRATED INTELLECTUAL ROBOTOTECHNICAL COMPLEXES” IIRTC-2024 (Kyiv, May 21-22, 2024).

Structure and scope of the work. The master's thesis consists of an introduction, 3 chapters and conclusions.

The Introduction presents a general description of the work, assesses the current state of the problem, substantiates the relevance of this study, defines the purpose and objectives of the work, and demonstrates the scientific novelty and practical value of the results.

Chapter One: review of theoretical information (basic principles, terms and tools) and existing methods of using DevOps practices; comparative analysis of DevOps practices and predecessor methodologies.

Chapter Two: selection of the optimal DevOps tools for the practical part of the course; preparation (research on how to implement continuous integration (CI), continuous deployment (CD), automated testing and system monitoring) for the implementation of DevOps practices in the third section.

Chapter Three: Implementation of DevOps practices to automate routine tasks during software development; collection of information about the operation of the newly created system for further analysis.

Conclusions: generalization of the research results; formulation of conclusions regarding the feasibility and necessity of using DevOps practices in the modern

production process; identification of possible vectors for the development of this approach in the near future.

The work is executed on 101 pages, contains 3 appendices and references to the list of used literature sources with 26 titles. This document contains 46 pictures and 4 tables.

Keywords: DevOps practices, CI/CD, monitoring, Docker, automation tools, Google Cloud Platform.

Зміст

Перелік скорочень та термінів.....	12
ВСТУП.....	15
1. ДОСЛІДЖЕННЯ DEVOPS МЕТОДОЛОГІЇ. АНАЛІЗ ІСНУЮЧИХ ІНСТРУМЕНТІВ ДЛЯ АВТОМАТИЗАЦІЇ РУТИННИХ ЗАВДАНЬ DEVOPS.	17
1.1. Історія виникнення DevOps методології	17
1.2. Основні принципи DevOps методології та їх детальний огляд.....	20
1.3. Необхідні знання для сучасного спеціаліста DevOps	22
1.4. Підходи до впровадження CI/CD в DevOps процесах.....	24
1.5. DevOps моніторинг.....	29
1.5. Використання Docker в DevOps процесах.....	32
1.7. Використання Kubernetes в DevOps процесах	37
1.8. Використання DevOps практик в хмарних середовищах та порівняння їх провайдерів.....	38
1.9. Недоліки, які можна зустріти при використанні методології DevOps	41
Висновки до першого розділу	42
2. ПІДГОТОВКА ДО ВПРОВАДЖЕННЯ АВТОМАТИЗАЦІЇ РУТИННИХ ЗАВДАНЬ НА ПРИКЛАДІ ВЕБ-ЗАСТОСУНКУ	44
2.1. Використання GitHub Actions для впровадження CI/CD	45
2.2. Можливості та способи використання Docker та Kubernetes для ізольованого розташування веб-застосунку.....	49
2.3. Використання хмарного середовища на прикладі Google Cloud Platform (GCP)	55

2.4. Можливості та потреба використання сервісу Google Cloud SQL ..	60
2.5. Моніторинг системи на Google Cloud Platform	63
Висновки до другого розділу	66
3. БЕЗПОСЕРЕДНЯ РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ТА ВПРОВАДЖЕННЯ АВТОМАТИЗАЦІЇ РУТИННИХ ЗАВДАНЬ DEVOPS	68
3.1. Створення веб-застосунку та опис принципу його роботи	69
3.2. Використання Docker для розробки веб-застосунку	74
3.3. Створення та налаштування Google Cloud SQL та GCP Artifact Registry	77
3.4. Імплементація практики CI/CD на основі GitHub Actions в хмарному середовищі Google Cloud Platform	81
Висновки до третього розділу	90
ВИСНОВКИ	92
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	94
ДОДАТОК А	Error! Bookmark not defined.
ДОДАТОК Б	Error! Bookmark not defined.
ДОДАТОК В	Error! Bookmark not defined.

Перелік скорочень та термінів

Agile – методологія управління проектом, яка передбачає ділення проекту на окремі шматки (фази) та фокусується на постійній співпраці та покращенні.

APM (Application Performance Monitoring) – практика відстеження ключових показників продуктивності програмного забезпечення за допомогою моніторингового програмного забезпечення та телеметричних даних.

CD (Continuous Delivery) – це двоскладовий процес, який стосується інтеграції, тестування та доставлення змін коду.

CI (Continuous Integration) – практика автоматичної та частої інтеграції змін коду в спільне сховище вихідного коду.

Cloud – термін належить до серверів, доступ до яких здійснюється через Інтернет, а також програмного забезпечення та баз даних, які працюють на цих серверах.

Cloud Computing – це надання обчислювальних ресурсів, таких як сервери, сховища, бази даних, мережі, програмне забезпечення та аналітика, через Інтернет на вимогу користувача.

Containerization (Контейнеризація) – віртуалізація на рівні операційної системи або віртуалізація на рівні застосунків на кількох мережевих ресурсів, щоб застосунки мали можливість працювати в ізольованих просторах користувача.

Google Compute Engine (GCE) – інфраструктурний сервіс хмарних обчислень, що є складовою Google Cloud Platform (GCP). Цей сервіс надає можливість користувачам запускати віртуальні машини на серверах Google з обраною потужністю та необхідними ресурсами.

DevOps – це набір правил, практик та інструментів, які в сукупності збільшують можливість розробляти продукт швидше за рахунок поєднання розробки (Development) та операцій (Operations).

DevOps Monitoring – відстежування всіх етапів розробки: починаючи з планування і розробки та закінчуючи тестуванням і розгортанням.

Docker – це платформа, яка надає можливість швидко створювати, тестувати та розгортати програми в ізольованих контейнерах, що працюють однаково в незалежності від середовища.

Dockerfile – набір інструкцій, які буде зчитувати Docker для створення певного образу.

Enterprise systems management (ESM) – це практика застосування управління IT-послугами з метою покращення продуктивності, ефективності та якості послуг, що надаються.

GCP (Google Cloud Provider) – набір хмарних сервісів від компанії Google, що надають інструменти для обчислення та зберігання даних. GSP дозволяє керувати та масштабувати застосунки на хмарній інфраструктурі Google.

GitHub – платформа для зберігання та керування своїм кодом на основі технології Git.

GitHub Actions – сервіс від компанії GitHub для легкої автоматизації всіх робочих процесів безпосередньо в самому репозиторії.

IaC (Infrastructure as Code) – основна практика DevOps методології, яка надає можливість керувати ресурсами хмарної інфраструктури (VM, мережі, бази даних та інше).

IDE (Integrated Development Environment) – це програмне середовище для зручної розробки програмного забезпечення із цілим переліком розширень, що спрощують та пришвидшують процес розробки.

Kubernetes (K8s) – застосунок з відкритим вихідним кодом для автоматизації розгортання, масштабування та керування ізольованими програмами, тобто контейнерами.

SDLC (Software development lifecycle) – економічно ефективний процес, який використовують розробники для створення справді якісного програмного забезпечення.

SRE (Site Reliability Engineering) – це підхід розробки програмного забезпечення для IT-операцій. Команди SRE використовують зазвичай програмне забезпечення, як інструмент для керування системами, розв’язання проблем і автоматизації операційних завдань.

SSH (Secure Shell) – метод захищеного надсилання команд на комп’ютер через мережу, яка не є захищеною.

SSL (Secure Sockets Layer) – протокол безпеки, який надає конфіденційність, автентифікацію та цілісність з’єднання через Інтернет.

VM (Virtual Machine) – програма, яка виконує операції схожі на фізичний комп’ютер, але тільки в емульованому стані.

Waterfall – методологія управління проектом, яка побудована на покроковому переході від початку і до кінця розвитку проекту.

Deployment – набір дій, які роблять програмне забезпечення доступ до використання за рахунок його запуску в робочому середовищі.

ВСТУП

У сучасному світі інформаційних технологій неперервне покращення та швидкість розробки програмного забезпечення є важливим фактором конкурентоспроможності для багатьох компаній. В сучасному світі команди розробників мають пристосовуватися та швидко впроваджувати новий функціонал у свій продукт, щоб залишатися привабливим для користувача. З цього погляду методологія DevOps виглядає не лише як підхід до розробки програмного забезпечення, але і як культура, що сприяє командному духу та швидкому постачанню нових технологій для користувача.

DevOps поєднує у собі культуру, практики та інструменти, що спрямовані на автоматизацію рутинних процесів розробки, тестування, розгортання та моніторингу програмного забезпечення. Це дає змогу збільшити швидкість розгортання продукту, зменшити час на виправлення помилок та поліпшити якість продукту.

Рутинні завдання, такі як розгортання, тестування та моніторинг, залишаються затратними по часу завданнями, потребують постійної уваги та втручання від розробників. Це сповільнює процес розробки внаслідок того, що програмісти відволікаються на організаційні питання та зменшується зосередженість над самою розробкою, що призводить до зниження ефективності всієї команди. Тому дослідження та розробка методів автоматизації цих рутинних завдань стає актуальною та вкрай важливою задачею для подальшого розвитку DevOps.

Метою даної магістерської дисертації є дослідження та розробка ефективного методу автоматизації рутинних завдань DevOps. Під цими словами матиметься на увазі аналіз наявних підходів та інструментів, потенційна розробка нових алгоритмів та методів, а також експериментальне впровадження DevOps практик та оцінка їхньої ефективності для подальших висновків необхідності таких дій.

Одним із головних завдань даного дослідження є розгляд більшості аспектів автоматизації рутинних завдань DevOps, в тому числі аналіз наявних підходів, інструментів та методів, а також визначення їх переваг та недоліків.

В рамках практичного дослідження буде проведена імплементація автоматизованих методів та інструментів в тестовий проект. Це дозволить оцінити ефективність різних підходів та їх вплив на робочий процес.

Важливим аспектом дослідження є оцінка зростаючої ефективності за рахунок впровадження автоматизації рутинних завдань.

В кінці результати дослідження будуть проаналізовані та узагальнені для формулювання висновків стосовно необхідності впровадження автоматизації рутинних завдань у DevOps процеси. Це може стати важливим доповненням до наявних практик розробки програмного забезпечення та потенційно допомогти підвищити ефективність робочого процесу.

Мета полягає в дослідженні наявних рішень та побудові методу автоматизації рутинних завдань DevOps, що може допомогти організаціям удосконалити свої DevOps процеси та забезпечити швидше, надійніше та якісніше розгортання нового програмного забезпечення. Це може мати достатньо вагомий вплив на конкурентоспроможність компаній.

1. ДОСЛІДЖЕННЯ DEVOPS МЕТОДОЛОГІЇ. АНАЛІЗ ІСНУЮЧИХ ІНСТРУМЕНТІВ ДЛЯ АВТОМАТИЗАЦІЇ РУТИННИХ ЗАВДАНЬ DEVOPS.

1.1. Історія виникнення DevOps методології

Перед усім необхідно подати чітке визначення методології DevOps. Цей термін є конкатенацією двох скорочень від слів “Development” та “Operations”. Термін був запропонований Патріком Дебуа у 2009 році [1]. DevOps – не є процесом, технологією чи якимось затвердженим стандартом на цей час. Найчастіше термін “DevOps” асоціюють із терміном культура і на те є причини [2]. DevOps це зміни в ІТ-культурі, які сфокусовані на швидкому постачанні ІТ-послуг шляхом впровадження гнучких методів у контексті системно-орієнтованого підходу. DevOps практики зосереджені на тому, щоб покращити співпрацю між операційними командами та когортою розробників.

Розвиток DevOps методології не був раптовістю, а скоріше був теоретичним напрацюванням експертів та ентузіастів, яке стало реальністю. Попередні методології не давали потрібної гнучкості виробничого процесу.

До основних попередників DevOps методології варто віднести [3]:

- Enterprise systems management (ESM). Багато з тих, хто брав участь у першому визначенні терміну DevOps, були системними адміністраторами. Ці спеціалісти з операційних процесів внесли в DevOps ключові практики ESM, такі як управління конфігураціями, моніторинг систем, автоматизоване забезпечення ресурсами та підхід до створення наборів необхідних інструментів (toolchain).
- Agile розробка. DevOps методологію можна розглядати, як продовження Agile методу в якому розробка програмного забезпечення передбачає тісну співпрацю замовників, менеджменту та

розробників для заповнення прогалин і швидкої ітерації до кращого продукту. Надання послуг і те, як взаємодіють додаток та системи, є фундаментальною частиною ціннісної пропозиції для клієнта, тому команда продукту має включати ці аспекти як пріоритетні. З цієї сторони, DevOps методологія просто розширює принципи Agile за межі коду на весь процес розробки.

Таким чином, DevOps з'явився завдяки еволюції наявних практик управління корпоративними системами та Agile розробки, об'єднавши їх для створення більш ефективних і скоординованих ІТ-процесів [4].

Історію започаткування DevOps методології можна розділити на кілька ключових етапів, кожен з яких відзначений важливими розробками та впровадженими інноваціями.

Ось деякі з ключових етапів в історії розвитку DevOps практик [5]:

1. Поява Agile: на початку 2000-х років Agile методології почала стрімко набирати популярності в розробці програмного забезпечення, в якому фокусувались на ітераційній та спільній розробці.
2. Потреба в DevOps: оскільки розробка програмного забезпечення ставала все складнішою та більш багатофакторною, стала очевидною необхідність в ефективніших і спільних процесах між командами розробки та операцій.
3. Ранні практики DevOps: початкові етапи DevOps методології включали постійну інтеграцію, неперервну доставку та автоматизоване тестування.
4. Розвиток хмарних обчислень (Cloud Computing): поява хмарних обчислень у середині 2000-х років дозволила автоматизувати виділення інфраструктури під запит, що ще більше прискорило розвиток DevOps культури.

5. Стандартизація: у 2012 році DevOps культура отримала ще більшої сили після публікації «The Phoenix Project», книги, яка сприяла прийняттю стандартизованих практик у командах розробки.
6. DevOps набирає обертів: до 2014 року DevOps став основним підходом до розробки програмного забезпечення, і великі організації, такі як Amazon, Netflix і Etsy, використовували його для вдосконалення своїх процесів доставки програмного забезпечення.
7. DevSecOps: в останні роки відбувся зсув у бік більш цілісного підходу до розробки програмного забезпечення, відомого як DevSecOps. Цей підхід підкреслює важливість інтеграції практик безпеки в процес DevOps, гарантуючи, що безпека є ключовим фактором із самих ранніх етапів розробки.
8. Cloud-Native DevOps: остання фаза руху DevOps зосереджена на хмарних DevOps практиках, які включають використання ресурсів і служб хмарних обчислень для оптимізації процесу розробки. До цього можна віднести використання архітектури мікросервісів, безсерверних обчислень (Serverless Computing) та інших хмарних технологій.

До використання DevOps практик команда розробників та операцій працювали незалежно одна від одної, що призводило до цілого ряду неефективних рішень у життєвому циклі розробки програмного забезпечення.

Основні проблеми, які виникали для організації до імплементації DevOps практик у виробничий процес, включають:

- Повільна доставка програмного забезпечення низької надійності;
- Низька комунікація між командами;
- Відсутність автоматизації кроків, що можуть бути такими;
- Завищені операційні витрати;
- Низький рівень безпеки програмного забезпечення.

1.2. Основні принципи DevOps методології та їх детальний огляд

Культура DevOps означає, що розробники стають ближчими до користувача, бо краще розуміють їхні вимоги та потреби. Операційні групи беруть участь у процесі розробки та додають вимоги до обслуговування та потреби клієнтів. Це означає дотримання наведених нижче ключових принципів, які допомагають командам DevOps надавати програми та послуги швидше та ліпшої якості, ніж організації, які використовують традиційні (застарілі) моделі розробки програмного забезпечення.

Загалом DevOps методологію варто описати, як неперервний цикл (рис. 1), що складається з таких кроків [6]:

- Plan (планування);
- Build (збірка);
- Test (тестування);
- Deploy (розгортання);
- Operate (експлуатація);
- Observe (спостереження);
- Feedback (зворотній зв'язок);
- Discover (дослідження).

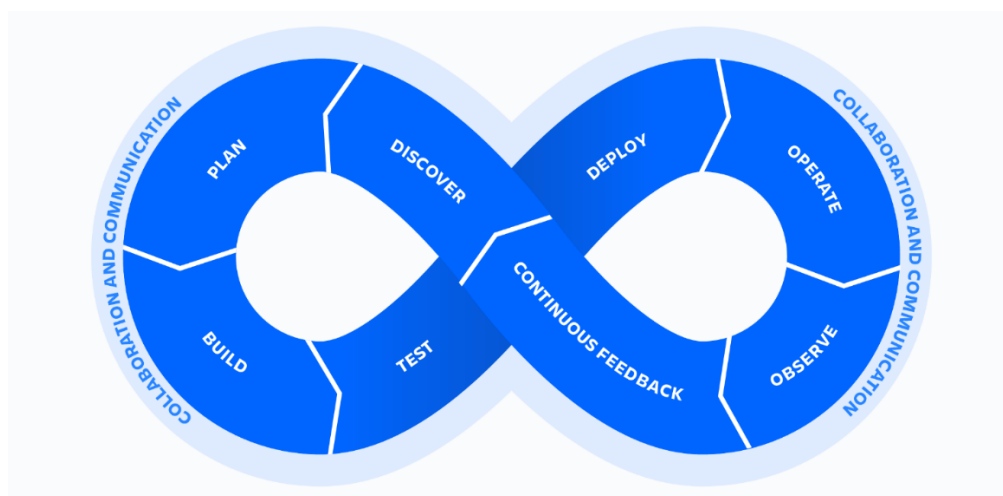


Рис.1.: Виробничий процес DevOps.

До переліку основних принципів DevOps методології [7] можна віднести такі аспекти, як:

- Співпраця (Collaboration): команди розробки та операцій об'єднуються у функціональну команду, яка спілкується, ділиться відгуками та співпрацює протягом усього циклу розробки продукту та його розгортання.
- Автоматизація рутинних завдань: автоматизація є ключовим елементом конвеєра CI/CD і допомагає знизити кількість людських помилок; автоматизація призводить до можливості постійного вдосконалення та зниження простою системи.
- Клієнтоорієнтованість: команди DevOps використовують короткі цикли зворотного зв'язку з клієнтами та кінцевими користувачами для розробки продуктів і послуг, зосереджених на потребах саме користувачів.
- Постійне вдосконалення та мінімізація витрат: використання автоматизації повторюваних завдань та перегляду показників продуктивності дає можливість скоротити час випуску або середній час відновлення; високоефективні команди DevOps регулярно шукають області, які можна було б покращити.

Успішне впровадження DevOps методології дозволяє компаніям значно підвищити ефективність розробки та впровадження програмного забезпечення, зменшити час виходу на ринок та підвищити якість продукту. Основні принципи DevOps, такі як співпраця, автоматизація, безперервна інтеграція та розгортання, інфраструктура як код (IaC)), моніторинг та безпека, забезпечують стабільний і надійний процес розробки. Їхній детальний огляд буде наведено в наступних розділах.

Організації, які впроваджують DevOps, можуть швидше реагувати на потреби ринку, забезпечувати високу якість продукту і задоволення

користувачів, що є ключовими факторами успіху в сучасному конкурентному середовищі.

1.3. Необхідні знання для сучасного спеціаліста DevOps

У сучасному світі інформаційних технологій, роль DevOps інженера стає все більш важливою для успішного функціонування організацій. DevOps інженери є ключовими фігурами, які забезпечують безперервну інтеграцію, розгортання, моніторинг та підтримку програмного забезпечення. Для того, щоб бути успішним у цій позиції, DevOps інженеру необхідно володіти вкрай широким спектром знань та вмінь, які охоплюють як технічні, так і нетехнічні аспекти роботи.

Цей розділ побудований на детальному огляді основних знань та навичок, які необхідні для сучасного DevOps інженера. Розглянемо ключові технічні компетенції, такі як знання з автоматизації, контейнеризації, хмарних технологій та управління конфігураціями. Окрім технічних навичок, важливо також розуміти значення комунікативних та аналітичних здібностей, які сприяють ефективній співпраці в командах та прийняттю обґрунтованих рішень.

Необхідні технічні знання DevOps інженера [8][9]:

1. Системне адміністрування;

Системне адміністрування є основою для будь-якого DevOps інженера. Це включає в себе знання про налаштування, управління та моніторинг серверів та операційних систем. DevOps інженер повинен мати поглиблені знання різних операційних систем (Linux та Windows). Необхідністю є вміння працювати з термінальними програмами, що дозволяє ефективніше автоматизувати рутинні завдання.

2. Автоматизація та написання скриптів;

Автоматизація є основною складовою DevOps культури в якій інженери повинні вміти писати скрипти для автоматизації завдань, використовуючи мову програмування Python та Bash чи PowerShell (в залежності від обраної операційної системи). Використання цих вмінь надає змогу скоротити час на виконання рутинних завдань, зменшити кількість людських помилок та збільшити ефективність процесів.

3. Контейнеризація продуктів та оркестрація;

Використання контейнеризації дозволяє ізолювати застосунки та їх залежності в єдиному цільному середовищі, що робить розгортання більш передбачуваним та зручним для розробника. DevOps інженери мають знати, як працювати з контейнерами за допомогою Docker. Не менше увагу варто приділяти і оркеструванню цих контейнерів з використанням Kubernetes (K8s). Це забезпечує масштабованість та надійність додатків в різних середовищах.

4. Хмарні технології (Cloud Technology);

Необхідно знати ще й хмарні платформи. Лідерами в цій сфері є: Amazon Web Services, Google Cloud, Microsoft Azure. Зазвичай, DevOps інженер обирає один з цих сервісів на свій смак та спеціалізується саме на ньому, бо спеціалізація на одному із сервісів забирає вкрай багато часу. До необхідних знань можна вказати такі аспекти: розуміння хмарних сервісів, управління ресурсами, безпека та налаштування мережі в хмарному середовищі.

5. Управління конфігураціями (Configuration Management);

DevOps інженери повинні вміти використовувати інструменти управління конфігураціями. Серед найпопулярніших можна подати такі продукти: Terraform, Ansible, Puppet, Chef та інші. Ці інструменти дають змогу автоматизувати налаштування інфраструктури та забезпечувати її відповідність потрібним стандартам.

6. Неперервна інтеграція та безперервне розгортання (CI/CD);

Знання принципів та інструментарію для неперервної інтеграції та безперервного розгортання є основою DevOps методології. Такі спеціалісти повинні вміти налаштовувати та використовувати інструменти для CI/CD, наприклад: Jenkins, GitLab CI або Circle CI. Це забезпечує автоматизацію процесу тестування, збірки та розгортання програмного забезпечення, що призводить до швидкого виявлення та виправлення помилок.

1.4. Підходи до впровадження CI/CD в DevOps процесах

Continuous Integration/Continuous Deployment (CI/CD) означає постійну інтеграцію та безперервне постачання/розгортання. Цей процес має на меті оптимізувати та прискорити життєвий цикл розробки програмного забезпечення.

Безперервна інтеграція (CI) стосується практики автоматичної та частоті інтеграції змін коду в спільне сховище вихідного коду. Безперервна доставка та/або розгортання (CD) [10] — це двоскладовий процес, який стосується інтеграції, тестування та доставки змін коду. Безперервна доставка не закінчується автоматичним виробничим розгортанням, тоді як постійне розгортання автоматично випускає оновлення у виробниче середовище.

Загалом ці взаємопов'язані практики часто називають «конвеєром CI/CD» і підтримуються командами розробки та операцій, які працюють разом у гнучкий спосіб за допомогою підходу DevOps. Інколи може використовуватись підхід розробки надійності сайту (SRE).

На рис. 2 зображено діаграму типового “конвеєру” CI/CD [11]. Умовно можна поділити цей процес на шість кроків, а саме:

1. Розробка програмного забезпечення (безпосереднє написання коду);
2. Процес Continuous Integration:
 - 2.1.Етап BUILD: збір проекту в інтегрованому середовищі розробки;
 - 2.2.Етап TEST: автоматичне проведення тесту для вже зібраного коду;
 - 2.3.Етап MERGE: якщо попередні кроки успішно пройдені, то виконується злиття в головну гілку репозиторія.
3. Процес Continuous Delivery: новий код автоматично опубліковується в головному репозиторії за умови, що попередній процес був успішно пройдений;
4. Процес Continuous Deployment: автоматичне розгортання застосунку в середовищі розробки.



Рис.2.: Виробничий процес CI/CD.

Тепер необхідно детальніше поговорити по кожну зі складових цього терміну та зрозуміти необхідність їхнього використання.

«CI» в терміні CI/CD означає безперервну інтеграцію, процес автоматизації для розробників, який сприяє частішому об’єднанню змін коду в основну або спільну гілку. По мірі внесення цих оновлень запускаються кроки автоматизованого тестування, щоб переконатися в надійності змін щойно об’єднаного коду.

У сучасній розробці додатків мета полягає в тому, щоб кілька розробників чи підкоманд одночасно працювали над різним функціоналом однієї і тієї ж програми. Однак, якщо організація налаштована на об'єднання всього розгалуженого вихідного коду в один день ("Merge Day"), то результат може бути наповнений великою кількістю ручної роботи.

Це спричинено тим, що внесені розробником зміни, які працюють ізольовано можуть мати конфлікти зі змінами, які одночасно вносять інші розробники. Ця проблема може бути ще більше ускладнена, якщо кожен розробник налаштував власне локальне інтегроване середовище розробки (IDE), а не команда домовилася про одне хмарне середовище розробки.

СІ можна розглядати як вирішення проблеми паралельної розробки багатьох гілок програми, які можуть конфліктувати одна з одною.

Добре налаштований СІ означає, що коли зміни, внесені розробником, об'єднані, то ці зміни перевіряються шляхом автоматичного створення програми та виконання автоматизованих тестів. Зазвичай виділяють два різновиди тестів: модульні та інтеграційні. Це означає, що буде проведено тестування абсолютно всього: від класів і функцій, до модулів, які утворюють цілу програму. Головна перевага СІ полягає в тому, що автоматичне тестування одразу виявляє конфлікти між новим та вже існуючим кодом.

«CD» у терміні СІ/CD стосується безперервної доставки та/або безперервного розгортання, які є пов'язаними між собою поняттями та іноді використовуються взагалі як синоніми. Обидва поняття стосуються автоматизації подальших етапів нашого "конвеєра". Вибір між безперервним постачанням і безперервним розгортанням залежить від стійкості до ризику та конкретних потреб команд розробників і операційних команд.

Безперервна доставка (CD) автоматизує випуск перевіреного коду в репозиторій після автоматичної збірки та тестування (модульного та інтеграційного) в СІ. Отже, щоб побудувати ефективний CD, важливо, щоб

якісний CI вже був імплементований у конвеєр розробки, бо CD ніби буде надбудовою на нього.

У CD кожен етап, від об'єднання змін коду до поставки зібраних проектів, передбачає автоматизацію тестування та автоматизацію випуску коду. В кінці цього процесу, операційна команда може швидко розгорнути робоче програмне забезпечення нової версії.

CD означає, що зміни, внесені розробником у програму, автоматично перевіряються на наявність помилок і завантажуються в репозиторій (в нашому випадку це буде GitHub), де їх потім може розгорнути операційна команда в робочому середовищі.

На мою думку, також, необхідні поговорити і про безпеку CI/CD. Для забезпечення безпеки в CI/CD використовуються автоматизовані перевірки і тестування, щоб запобігти вразливостям у поставці програмного забезпечення [12]. Впровадження безпеки у конвеєр допомагає захистити код від атак, запобігти витоку даних, відповідати політикам і гарантувати якість.

Для додаткової безпеки в “конвеєрі” CI/CD можна використовувати методи “Shift Left” (“Зсув вліво”) та “Shift Right” (“Зсув вправо”). Перший метод, “Зсув вліво” [13], означає включити тестування безпеки якнайшвидше, щоб знайти вразливості та виправити дефекти якомога раніше в розробці. Другий метод, “Зсув вправо” [14], означає відстежувати поведінку користувача, використання, продуктивність і показників безпеки на етапі виробництва, щоб перевірити працездатність програмного забезпечення.

Не варто нехтувати безпекою процесу розробки, бо за її відсутності неминучими стають такі проблеми:

- Розкриття конфіденційних даних зовнішнім джерелам;
- Використання незахищеного коду або сторонніх компонентів;
- Несанкціонований доступ до сховищ вихідного коду;

- Несанкціонований доступ або перехоплення контролю інструментів для зборки.

Необхідно зробити огляд на основні інструменти, які DevOps інженери використовують для налаштування CI/CD. Деякі інструменти спеціально обробляють саме інтеграцію (CI), деякі керують лише розробкою та розгортанням (CD), тоді як інші спеціалізуються на постійному тестуванні. Ми візьмемо на розгляд такі інструменти, як: Jenkins, GitHub Actions, GitLab CI/CD та Ansible.

Jenkins – найпопулярніший інструмент з відкритим вихідним кодом (open source) для реалізації безперервної інтеграції та безперервного розгортання (CI/CD) [15]. Він вміє автоматизувати процеси збірки, тестування та розгортання програмного забезпечення. Jenkins підтримує величезний перелік розширень, які збільшують його функціональність і дозволяють інтегруватися з різними інструментами та на різних платформах. Цей застосунок має зручний веб-інтерфейс. Його легко можна налаштувати і користуватись ним, що робить його очевидним вибором багатьох компаній та команд розробників.

GitHub Actions – потужна платформа автоматизації, вбудована безпосередньо в GitHub, яка дозволяє розробникам автоматизувати робочі процеси для безперервної інтеграції та безперервного розгортання (CI/CD) [16]. Вона дозволяє створювати, тестувати та розгортати код безпосередньо з GitHub, використовуючи файли типу YAML для визначення робочих процесів.

GitLab CI/CD – це комплексна система для автоматизації процесів розробки, тестування та розгортання програмного забезпечення, вбудована в платформу GitLab [17]. Вона дозволяє компаніям автоматизувати весь цикл розробки, що сприяє швидкому випуску якісного програмного забезпечення.

Ansible – це популярний інструмент для автоматизації управління конфігураціями, розгортання програмного забезпечення та оркестрації інфраструктури [18]. Він вирізняється на фоні інших своєю простотою

використання, масштабованістю та потужними можливостями, що дозволяє ефективно управляти великою кількістю серверів та інших компонентів інфраструктури.

Неперервна інтеграція та безперервне розгортання (CI/CD) є основними складовими DevOps методології, що дають змогу організаціям значно прискорити постачання програмного забезпечення, підвищуючи його якість та ефективність. Налаштування CI/CD процесу дає змогу автоматизувати інтеграцію нового коду в основне сховище, тестування та впровадження змін у коді, забезпечуючи оновлення без перебоїв. Впровадженню CI/CD подарувало командам розробників та операційним команди більшу злагожденість, що призвело до оптимізації процесу розробки та поставки програмних продуктів.

1.5. DevOps моніторинг

Від впровадження DevOps практик очікується пришвидшення розробки, регулярне тестування та можливість частіше оновлювати програмне забезпечення, при цьому покращуючи якість і скорочуючи витрати на всі процеси. Для досягнення цього існують інструменти моніторингу, які забезпечують автоматизацію, розширені вимірювання та видимість протягом усього життєвого циклу розробки: починаючи з планування та закінчуючи розгортанням.

Сучасний життєвий цикл розробки програмного забезпечення швидший, ніж будь-коли раніше. Оскільки часті зміни коду стали звичним явищем, командам розробників потрібен DevOps моніторинг, який надає комплексне уявлення про виробниче середовище в реальному часі.

Моніторинг DevOps передбачає нагляд за всіма процесами розробки: планування, розробки, інтеграції, тестування та розгортання [19]. Він надає

повноцінне відстеження у режимі реального часу стану програм, служб та інфраструктури у виробничому середовищі.

До критично важливих компонентів моніторингових DevOps застосунків можна віднести:

- Система оповіщень;
- Системи збору даних;
- Управління продуктивністю застосунків (APM);
- Автоматизоване відновлення;
- Потокова передача в реальному часі;
- Відтворення історії;
- Візуалізація.

DevOps моніторинг дозволяє розробникам швидко й в автоматичному режимі реагувати на будь-які погіршення взаємодії з клієнтами. Ще більш важливішим є те, що моніторинг дозволяє командам розробників виконати «зсув вліво» до попередніх етапів розробки та сфокусуватись на мінімізації негативного ефекту від неякісних чи помилкових змін.

Необхідність використання DevOps моніторингу можна сформулювати такими тезисами:

- Регулярні зміни коду потребують прозорості;
Постійні зміни в коді, спричинені безперервною інтеграцією та розгортанням, пришвидшили темпи змін і зробили виробниче середовище дедалі складнішим. З мікросервісами та мікроінтерфейсами, що входять у сучасне хмарне середовище, у виробництві працюють сотні (інколи тисячі) різних процесів, кожен з яких має унікальні вимоги щодо масштабу, затримки, резервування ресурсів та безпеки.

- Проведення експериментів;

Потреба в оптимізації продуктів відповідно до потреб клієнтів призводить до постійних експериментів. Виробничі процес може запускати сотні експериментів паралельно, через що системам моніторингу важко повідомляти про конкретну причину погіршення роботи.

- Управління змінами ;

Оскільки більшість збоїв у виробництві спричинені змінами, то вміння керувати ними є вкрай важливим. Необхідно визначити ризики, пов'язані зі змінами, і автоматизувати кроки на основі ризику зміни. Особливо добре стає помітно це в критичних сферах, наприклад, медична, військова або фінансова

- Моніторинг залежної системи.

Тепер командам потрібно не лише контролювати системи, які вони створюють, а й контролювати та керувати продуктивністю та доступністю всіх залежних систем.

До найпопулярніших моніторингових DevOps інструментів можна віднести:

- Prometheus: популярний інструмент моніторингу системи з відкритим вихідним кодом; він створений спеціально для моніторингу в DevOps процесах; потужна наскрізна система моніторингу та має гарний менеджер сповіщень.
- Grafana: аналітична та інтерактивна платформа візуалізації з відкритим вихідним кодом, яка підтримує методи представлення даних за допомогою архітектури підключених панелей;
- New Relic: потужна хмарна платформа моніторингу, яка забезпечує повний набір технологій для спостереження.

1.5. Використання Docker в DevOps процесах

Docker – це платформа, розроблена, щоб допомогти розробникам створювати, спільно використовувати та запускати контейнерні (ізолювані) програми [20].

Контейнер (Container) у Docker — це легка, автономна та ізолювана програма, яка включає в себе все необхідне для запуску: код, середовище виконання, потрібні бібліотеки та всі залежності [20]. Контейнери дають змогу розробникам запаковувати застосунки з усіма необхідними компонентами.

До головних переваг та особливостей використання Docker можна віднести такі моменти [21]:

- Портативність;
Docker дозволяє упаковувати додатки та їхні залежності в легкі портативні контейнери, які можуть узгоджено працювати в різних середовищах, включаючи розробку, тестування і т.д.
- Послідовність;
Docker забезпечує послідовну та надійну роботу програм у різних середовищах, усуваючи проблеми універсальності та покращує якість програмного забезпечення.
- Ізоляція;
Контейнери Docker забезпечують ізоляцію між програмами та їхніми залежностями, запобігаючи конфліктам і гарантуючи, що зміни в одному контейнері не впливають на інші контейнери чи host-систему.
- Масштабованість;
Docker дозволяє легко масштабувати застосунок забезпечуючи ефективне використання ресурсів і балансування навантаження.
- Швидке розгортання;

Docker забезпечує швидке й автоматичне розгортання додатків, скорочуючи час і зусилля на розгортання.

- Багаторазове використання;

Образами Docker можна ділитися та повторно користуватись, що дозволяє швидше розробляти, тестувати та розгортати програми.

- Інтеграція в DevOps процеси;

Docker широко використовується в робочих процесах DevOps, що забезпечує бездоганну інтеграцію з іншими інструментами та практиками DevOps, наприклад, конвеєри неперервної інтеграції та розгортання (CI/CD), які були описані в позапопередньому розділі.

- Безпека.

Docker надає вбудовані безпековий функціонал: ізоляція контейнерів; обмеження ресурсів; чітке керування доступами.

Загальний принцип використання Docker можна побачити на рис. 3.

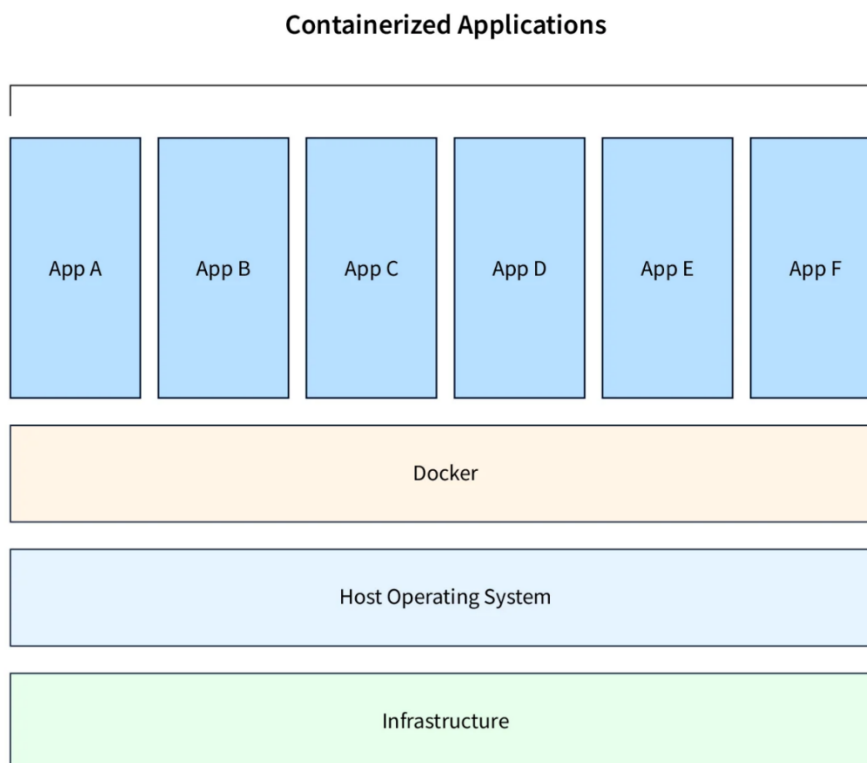


Рис.3.:Застосунок в Docker.

Docker у DevOps стосується інтеграції технології контейнеризації Docker у практики та робочі процеси DevOps. У контексті DevOps Docker використовується, як інструмент для оптимізації та вдосконалення розробки, тестування, розгортання та керування додатками, що дозволяє організаціям та компаніям досягати швидшої та ефективнішої розробки із постачанням програмного забезпечення за мінімальні терміни.

Docker для DevOps процесу може запропонувати кілька переваг [21]:

- Забезпечення узгоджених і відтворюваних збірок;
- Покращення переносимості додатків у різних середовищах;
- Спрощення процесів розгортання та повернення до попередньої версії;
- Покращення співпраці між командами розробки та операцій;
- Полегшення впровадження сучасних практик DevOps:
 - Неперервна інтеграція та розгортання (CI/CD);
 - Автоматизоване тестування.

Окрім традиційного розгортання та контейнеризованого варто згадати, що є і за допомогою віртуальної машини. Необхідно зробити порівняння цих методів, щоб краще зрозуміти, чому Docker стає беззаперечним переможцем у цьому змаганні [21].

Принцип традиційного розгортання програмного забезпечення (рис. 4):

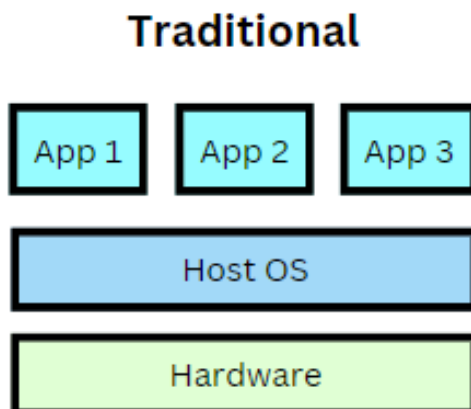


Рис. 4: Традиційний принцип розгортки.

Повна відсутність переваг порівняно з двома іншими способами.

До головних недоліків традиційного способу можна віднести наступне:

1. Відсутність ізоляції ресурсів;
2. Проблематика масштабування;
3. Висока вартість;
4. Відсутність гнучкості дій в разі помилки системи.

Принцип розгортання програмного забезпечення на VM (рис. 5):

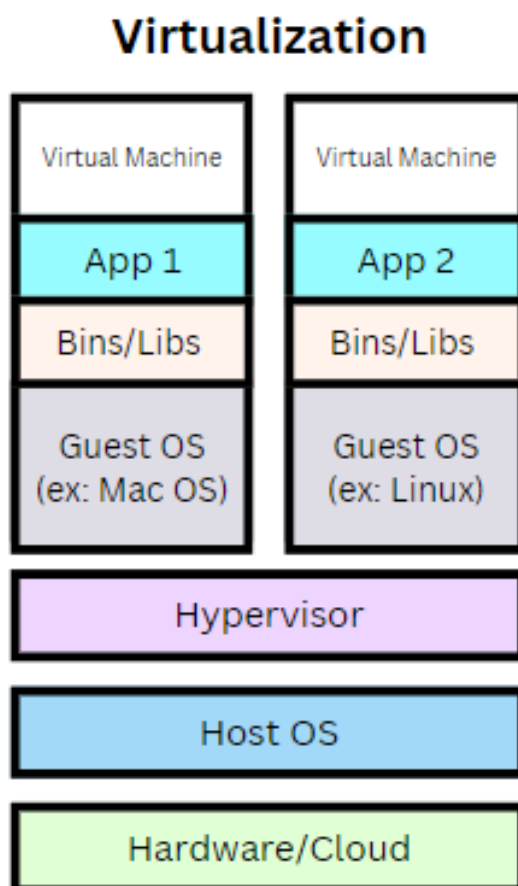


Рис. 5: Принцип розгортка через віртуальну машину.

До головних переваг способу на основі VM можна віднести наступне:

1. Краще використання ресурсів, аніж в традиційному способі;
2. Застосунки стають ізольованими.

До головних недоліків способу на основі VM можна віднести наступне:

1. Розмір .iso файлів;

2. Відсутність портативності;
3. Відсутність здатності до масштабування;
4. Вартість може неочікувано зрости.

Принцип розгортання програмного забезпечення в контейнері (рис. 6):

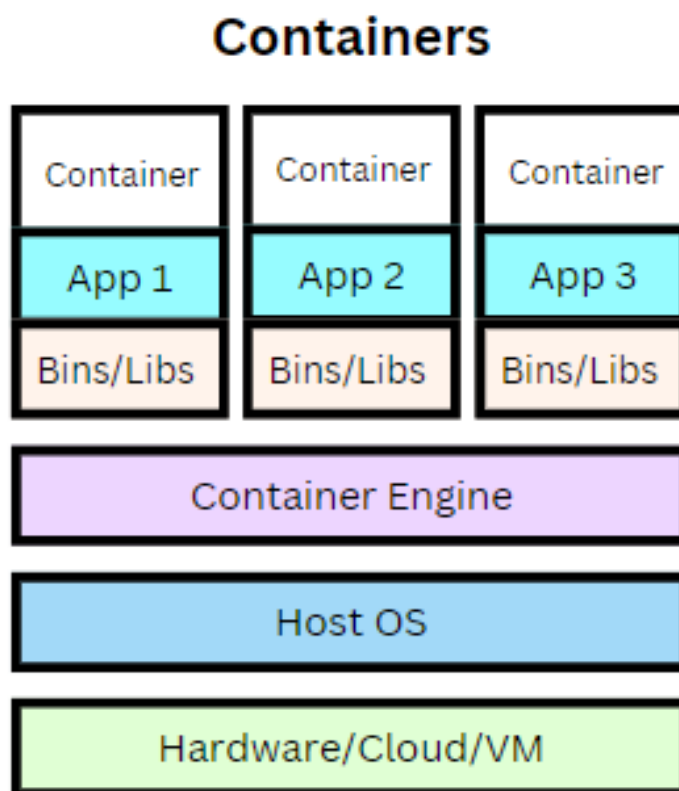


Рис. 6: Принцип розгортки з використанням Docker.

До головних переваг способу із використанням контейнеру можна віднести наступне:

1. Надзвичайно маленький образ та висока швидкість його запуску;
2. Надзвичайно висока портативність контейнерів;
3. Низька вартість;
4. Високий рівень масштабованості за потреби.

До головних недоліків способу із використанням контейнеру можна віднести наступне:

1. Обмеження можливостей деяких застосунків;
2. Можливі проблеми із безпекою не дивлячись на ізолюваність.

1.7. Використання Kubernetes в DevOps процесах

Kubernetes – портативна, масштабована платформа з відкритим вихідним кодом для керування контейнерними, робочими навантаженнями та службами [22]. Вона полегшує і декларативне налаштування, і автоматизацію.

Перелік головних причин, чому Kubernetes став незамінним інструментом під час налаштування DevOps “конвеєру”:

- Виявлення служб і балансування навантаження;
Якщо трафік на контейнер високий, то Kubernetes здатний збалансувати навантаження і розподілити мережевий трафік так, щоб розгортання було стабільним та без провисань.
- Оркестрування сховища;
Kubernetes дозволяє автоматично монтувати систему зберігання даних на ваш вибір (локальні сховища, публічні хмарні сервіси, тощо).
- Автоматизовані розгортання та відкати до попередніх версій;
Є можливість описати бажаний стан для розгорнутих контейнерів за допомогою Kubernetes, і він може змінювати фактичний стан на бажаний з контрольованою швидкістю.
- Автоматичне пакування контейнерів;
Можна чітко задавати Kubernetes скільки процесорної та оперативної пам'яті потрібно кожному контейнеру.
- Самовідновлення;
Kubernetes перезапускає контейнери, які вийшли з ладу, замінює

контейнери, знищує контейнери, які не реагують на визначену користувачем перевірку стану, і не просуває їх клієнтам, поки вони не будуть знову готові до роботи.

- Керування конфігураціями;
Kubernetes дозволяє зберігати та керувати конфіденційною інформацією, зокрема паролями, токенами OAuth та ключами SSH.
- Горизонтальне масштабування;
Можна змінювати масштаб свого застосунку вгору або вниз використовуючи графічний інтерфейс або на основі використання процесора.

У DevOps використання Kubernetes дозволяє автоматизувати процеси розгортання, масштабування та управління додатками, забезпечуючи при цьому високу доступність, стабільність та найкраще використання ресурсів системи.

1.8. Використання DevOps практик в хмарних середовищах та порівняння їх провайдерів

Хмарні обчислення (Cloud Computing) — це доступ за вимогою до обчислювальних ресурсів — фізичних серверів або віртуальних серверів, сховищ даних, мережеских можливостей, інструментів розробки застосунків, програмного забезпечення, аналітичних інструментів та інше [23].

Різновиди хмарних обчислень зазвичай ділять на три великі категорії:

1. Infrastructure as a Service (IaaS);

IaaS містить основні структурні блоки для хмарних технологій. Зазвичай IaaS надає доступ до мережеских функцій, комп'ютерів і місця для зберігання даних. IaaS надає вам найвищий рівень гнучкості та контролю над Вашими ресурсами.

2. Platform as a Service (PaaS);

PaaS позбавляє вас від необхідності керувати апаратним забезпеченням та операційними системами і дозволяє зосередитися на розгортанні та управлінні вашими програмами.

3. Software as a Service (SaaS).

SaaS надає Вам повний продукт, яким керує постачальник послуг. У більшості випадків люди, що мають на увазі SaaS, мають на увазі програми для кінцевих користувачів.

IaaS становить складову PaaS, тоді як сам PaaS становить підмножину SaaS. Використання основних структурних блоків кожного різновиду хмарного сховища можна побачити на рис. 7.

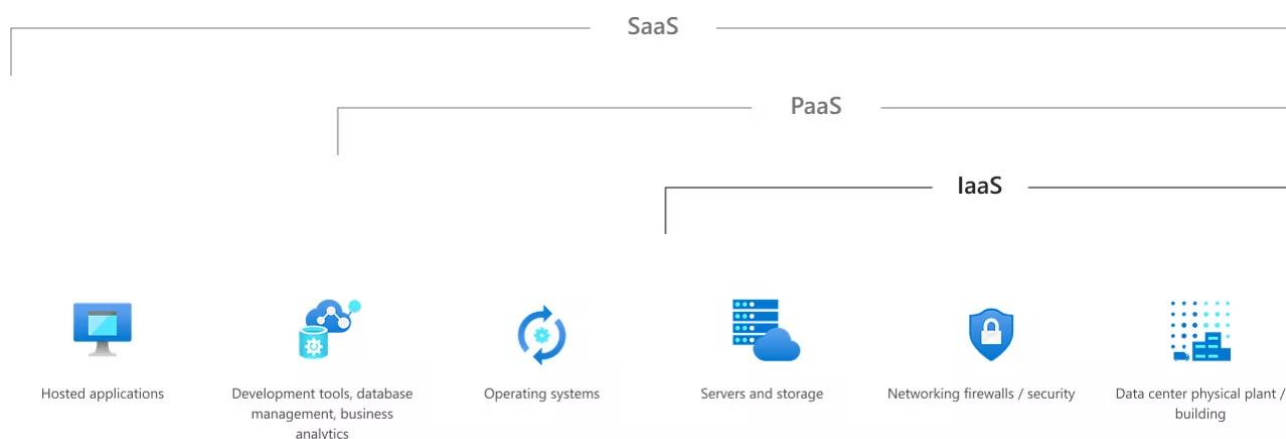


Рис. 7: Складові SaaS, PaaS та IaaS.

Розглянемо основний метод використання DevOps практик в хмарному середовищі. Цей метод стосується організацій які керують і автоматизують свою інфраструктуру за допомогою технології Cloud Computing на основі DevOps практик. Це надає їм гнучкий підхід до різних виробничих процесів. Вважається, що процеси DevOps є вкрай гнучкими, якщо їх правильно впроваджувати, але інколи вони можуть стикатися з обмеженнями локального середовища.

У подібних випадках хмарна інфраструктура є важливим рушієм для DevOps. Це мінімізує затримку та забезпечує централізоване керування через уніфіковану платформу для розгортання, тестування, інтеграції та випуску програмного забезпечення [24]. Хмарні середовища часто також пропонують

інструменти CI/CD для автоматизації процесів DevOps. Це дає змогу командам DevOps співпрацювати між собою в розподілених корпоративних середовищах, а також адаптуватися до можливих змін.

Рішення впровадження Cloud DevOps часто вважаються економічно ефективнішим, ніж локальні рішення. Вони полегшують керування, об'єднуючи різні середовища та зменшуючи навантаження на розробників. Ця хмарна автоматизація DevOps високого рівня допомагає мінімізувати людські помилки та оптимізувати рутинні процеси та завдання.

До провідних хмарних постачальників в світі в першу чергу варто віднести Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform [25]. Вони мають цілий перелік інструментів для DevOps інженера. Всі ці платформи вкрай схожі за своїми можливостями та характеристиками, але варто визнати, що AWS є найбільш популярним. Кожен із них забезпечує наскрізне середовище для команд DevOps, що дозволяє їм позбутися необхідності завантажувати, вивчати та інтегрувати багатопшарові рішення.

У даній магістерській дисертації реалізація проекту буде проведена саме в Google Cloud Platform. Хоча це і не найпопулярніший хмарний провайдер, але треба розуміти, що якість послуг та їх різноманіття приблизно однакові.

Третина всього хмарного середовища зайнята AWS (рис. 8) в основу за рахунок гарного маркетингу та часу, коли сервіс з'явився, а він був розробленим одним з першим. Компанії, особливо великі, не готові переходити з одного хмарного сховища на інше, якщо відмінності і кількості переваг вимірюється на рівні статистичної похибки.

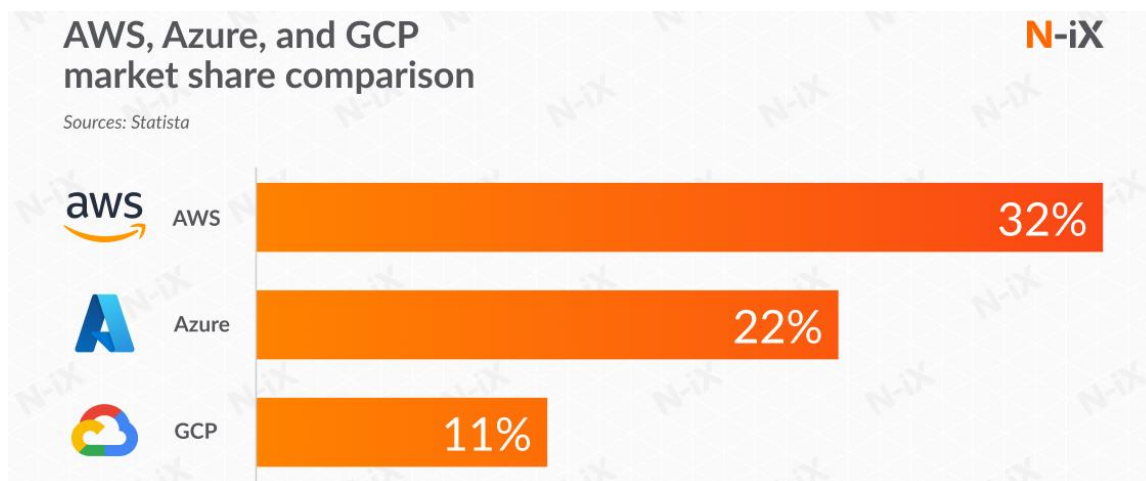


Рис. 8: Статистика використання AWS, Azure та GCP.

1.9. Недоліки, які можна зустріти при використанні методології DevOps

Недоліків використання DevOps методології не так багато, але вони є. Серед основних можна вказати такі [26]:

1. Проблеми впровадження самої методології у виробничий процес:
 1. Важкість зміни культури розробки до якого команда вже звикла;
 1. Складність використання нових інструментів для впровадження та використання DevOps практик.
2. Вартість впровадження нової методології:
 2. Вартість подібних спеціалістів може бути зависокою для певних компаній і не є доцільним;
 2. Зростання поточних витрат на технічне обслуговування.
3. Моніторинг:
 3. Потреба великою кількості ресурсів, оскільки на це йде багато зусиль;
 3. Повна залежність від автоматизації може бути шкідливою.

4. Складність набору інструментів:

- 4. Вкрай велика кількість знань, якими має володіти справжній DevOps інженер;
- 4. Проблеми і простій системи під час початкової інтеграції нових інструментів у виробничий процес.

Однак, не так все погано, як може здатися на перший погляд. Є ціла низка стратегій для подолання потенційних проблем, а саме:

- 1. Потрібен централізований підхід до зміни культури розробки на DevOps зі сторони керівництва;
- 2. Знизити проблему безпеки за рахунок регулярних аудитів для глибокої перевірки;
- 3. Витратами можна керувати шляхом оптимізації використання ресурсів. Автоматизація рутинних завдань може з часом значно зменшити вартість виробничого процесу;
- 4. Оскільки ручні завдання будуть автоматизовані, тому безперервний моніторинг і технічне обслуговування стануть легко керованими.

Висновки до першого розділу

Методологія DevOps, що виникла на початку 2000-х років як природня реакція на необхідність кращої співпраці між командами розробників та операцій, стали невід’ємними у сучасному світі розробки програмного забезпечення. Їх історія почалася з усвідомлення того, що традиційні підходи до розробки і впровадження програмного забезпечення не можуть забезпечити достатню швидкість і якість у сучасному світі, де швидкість виробничого процесу тільки збільшується. Завдяки цій методології, розробка програмного забезпечення стала більш ефективною, гнучкою та швидкою.

Основні принципи DevOps включають в себе такі аспекти:

1. Автоматизацію процесів;
2. Неперервну інтеграцію та розгортання (CI/CD);
3. Спільну відповідальність між командами;
4. Управління інфраструктурою, як кодом (IaC).

Ці принципи дозволяють забезпечити високий рівень співпраці, прискорення розробки та підвищення якості продуктів.

Для DevOps спеціаліста важливими є знання у галузях системного адміністрування, програмування, мережевої безпеки, автоматизації, а також навички роботи з інструментами для управління інфраструктурою та оркестрації контейнерів. DevOps інженери повинні мати розуміння про процеси CI/CD, моніторинг систем, а також знання про хмарні платформи та сервіси, які вони надають в користування.

Впровадження CI/CD у DevOps вимагає використання інструментів, що підтримують автоматизацію процесів інтеграції та розгортання, таких як Jenkins, GitLab CI/CD, CircleCI. Ці інструменти дають змогу безперервно інтегрувати зміни в коді та на автоматичному рівні розгортати їх у різних середовищах, що підвищує швидкість розробки та знижує ризик помилок.

Моніторинг у DevOps є ключовим елементом, що дозволяє забезпечити стабільність і продуктивність систем. Використовуючи такі інструменти, як Prometheus та Grafana, є змога збирати, аналізувати та візуалізувати дані про стан систем, швидко виявляти і реагувати на проблеми.

Docker став невід'ємною складовою DevOps практик, бо він забезпечує стандартизацію середовищ розробки та розгортання. Використання контейнерів дозволяє створювати ізольовані середовища, що спрощує процеси тестування та розгортання додатків.

Kubernetes (оркестратор контейнерів) дає можливість автоматизувати управління контейнеризованими застосунками, забезпечуючи їх масштабованість, високу доступність та можливість саомстійно відновлюватись.

Він дозволяє ефективно керувати складними системами, автоматизувати розгортання і оновлення застосунків.

Впровадження DevOps практик у хмарних середовищах значно розширює можливості для автоматизації, масштабованості та гнучкості. Наразі це один з останніх етапів еволюції DevOps культури. Хмарні платформи (Amazon Web Services, Google Cloud Platform та Azure) надають потужні інструменти для управління інфраструктурою, автоматизації процесів конвеєру CI/CD, моніторингу та забезпечення належного рівня безпеки. Це дозволяє компаніям швидко адаптуватися до умов ринку, ефективно використовувати ресурси та знижувати витрати на виробничий процес.

Отже, DevOps та хмарні технології (Cloud Computing) стали вже невід'ємними компонентами сучасного підходу до розробки та розгортання програмного забезпечення (особливо у великих компаніях). Вони дозволяють компаніям досягати високої швидкості розробки, підвищувати якість продуктів та ефективно управляти інфраструктурою. Використання цих практик та інструментів забезпечує конкурентні переваги.

2. ПІДГОТОВКА ДО ВПРОВАДЖЕННЯ АВТОМАТИЗАЦІЇ РУТИННИХ ЗАВДАНЬ НА ПРИКЛАДІ ВЕБ-ЗАСТОСУНКУ

Для реалізації веб-застосунку “Блог” було створено простий проект в якому можна зробити обліковий запис, робити дописи та додати допис і обране. На цьому застосунку буде реалізовано базову інтеграцію DevOps практик. Буде виконана реалізація основних аспектів DevOps методології, які дадуть змогу поліпшити якість розробки та швидкість оновлень такого веб-застосунку.

Впровадження DevOps практик допоможуть веб-застосунку поліпшити якість розробки та підвищити швидкість оновлень веб-застосунку.

Реалізацію веб-застосунку “Блог” було зроблено на основі фреймворку Django. Для короткої демонстрації роботи застосунку буде використаний Postman.

Через особисті вподобання для системи контролю версій був обраний GitHub на якому і буде розміщений веб-застосунок.

Для налаштування неперервної інтеграції та поставки (CI/CD) буде використано GitHub Actions, оскільки це інструмент, інтегрований з GitHub, який забезпечує гнучкі та потужні можливості для автоматизації процесів розробки. Він є оптимальним рішенням для проектів малої та середньої складності.

Очевидно, що необхідно використати Docker для розробки локально, бо це один зі стовпів методології DevOps. За допомогою Docker буде створено ізольований контейнер (container), що в подальшому дасть змогу спростити налаштування та розгортання вже в хмарному середовищі.

Як постачальника хмарних послуг буде обрано Google Cloud Platform, бо він входить в трійку найбільших провайдерів та має безкоштовний пробний період, якого буде достатньо для мого проекту. GCP забезпечує високу надійність, масштабованість та безпеку, що робить його ідеальним вибором для хмарного розгортання веб-застосунку.

2.1. Використання GitHub Actions для впровадження CI/CD

Впровадження практик безперервної інтеграції (CI) та безперервної доставки (CD) стає стандартом для сучасної розробки програмного забезпечення.

GitHub Actions надає потужні можливості для автоматизації всіх цих процесів безпосередньо в межах самої платформи GitHub. В цьому розділі нам необхідно детально розглянути, як саме потрібно використовувати GitHub Actions для впровадження CI/CD, включаючи такі аспекти, як:

3. Налаштування;

4. Створення робочих процесів;
5. Інтеграцію з іншими інструментами та практичні приклади.

GitHub Actions — це інструмент автоматизації, вбудований безпосередньо в GitHub, який дозволяє створювати робочі процеси під власні потреби для автоматизації різноманітних завдань, таких як тестування коду, збірка проекту та його розгортання, тощо. Для налаштування GitHub Actions використовують звичайні YAML файли в якому розробники визначають послідовність дій (actions), що мають виконуватись у відповідь на певні події (events). До подій можна віднести такі дії, як push (передача локального коду у віддалений репозиторій), pull request (запит на внесення змін з однієї гілки репозиторія в іншу), issue (відслідковування помилок та завдань) та інші.

Основні складові GitHub Actions, які ми будемо використовувати в подальшому для налаштування неперервної інтеграції:

- Job (робота);
Певна діяльність у сховищі, яка ініціює запуск робочого процесу.
- Workflow (робочий процес);
Автоматизований процес, який можна налаштувати для виконання одного або кількох завдань (job). Робочі процеси визначаються у файлі типу YAML і запускатимуться, коли їх ініціює подія (event).
- Runners (виконувач);
Сервер, який запускає ваші робочі процеси, коли вони спровоковані.
- Action (дії).
Спеціальна програма, яка виконує повторюване завдання.

Схематичне зображення принципу роботи робочого процесу (workflow) в GitHub Actions можна побачити на рис. 9.

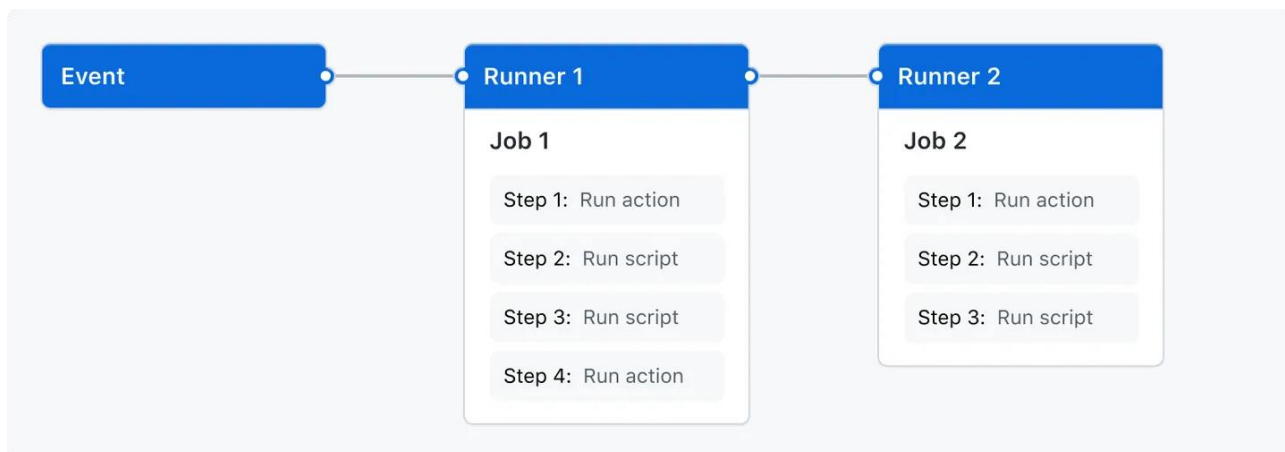


Рис. 9: Схеми робочого процесу (workflow) в GCP.

Для створення та налаштування workflow (робочого процесу) необхідно заповнити YAML файл відповідно до своїх потреб.

Базовий шаблон (template) для YAML файлу:

```

name: librevlas-learn-github-actions
run-name: ${github.actor} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm install -g bats
      - run: bats -v
  
```

Заповнений файл необхідно завантажити (зробити commit Ваших змін) у Ваш репозиторій GitHub за таким шляхом: `.github/workflows/`.

Необхідно детально розглянути (табл. 1) кожен з кроків (рядків), що використовуються у файлі налаштувань для подальшого розуміння виконуваних кроків.

Розбір команд, які використовуються у файлі YAML базової конфігурації налаштування workflow (робочого процесу) в GitHub Actions.	
<code>name: librevlas-learn-github-actions</code>	Назва робочого процесу (workflow).
<code>run-name: \${{ github.actor }} is learning GitHub Actions</code>	Назва для виконувача (runner) у Вашому робочому процесі (workflow).
<code>on: [push]</code>	Тригер, який для цього робочого процесу (workflow). В даному випадку тригером є саме команда push.
<code>jobs:</code>	Групує всі завдання, які будуть виконуватись в нашому робочому процесі (workflow).
<code>check-bats-version:</code>	Визначає роботу (job) із назвою check-bats-version.
<code>runs-on: ubuntu-latest</code>	Конфігурація системи для запуску.
<code>steps:</code>	Групує всі кроки, яку буде запускати робота (job) check-bats-version.
<code>- uses: actions/checkout@v4</code>	Слово uses вказує, що на цьому кроці виконуватиметься дія v4.
<code>- uses: actions/setup-node@v4 with: node-version: '20'</code>	Встановлення потрібної версії. В даному випадку саме Node.js.
<code>- run: npm install -g bats</code>	Виконання певної команди виконувачем (runner).
<code>- run: bats -v</code>	Виведення поточної версії програмного забезпечення.

Табл. 1: YAML файл для GitHub Actions.

Переглянути активність (runs) робочого процесу (workflow) можна у розділі “Actions” (рис. 10) в GitHub.



Рис. 10: Розділ Actions в GitHub.

В наступному розділі цієї магістерської дисертації буде подано налаштування відповідного YAML файлу для вибудови робочого процесу (workflow) в сервісі GitHub Actions.

2.2. Можливості та способи використання Docker та Kubernetes для ізольованого розташування веб-застосунку

Перед ознайомленням із конкретним використанням Docker необхідно чітко визначити потребу його використання при впровадженні DevOps практик для автоматизації рутинних завдань.

Docker надає можливість створювати, так звані, контейнери (container) для розгортки програмного забезпечення, які мають значну кількість переваг порівняння з традиційними методом та із використанням віртуальної машини (VM). Детально про переваги та недоліку було описано в розділі 1.6 цієї магістерської дисертації.

Контейнер (container) — це процес ізольованого програмного середовища, який виконується на host-машині, ізольовано від усіх інших процесів, запущених на цій host-машині. Ця ізоляція використовує kernel namespace (функція Linux Kernel) та cgroups (Control Groups, функція Linux Kernel). Docker робить ці можливості доступними та простими у використанні.

На сьогоднішній день Docker є універсальним інструментом, що працює на багатьох типах інфраструктури (рис. 11), для зручного тестування та розробки програмного забезпечення.

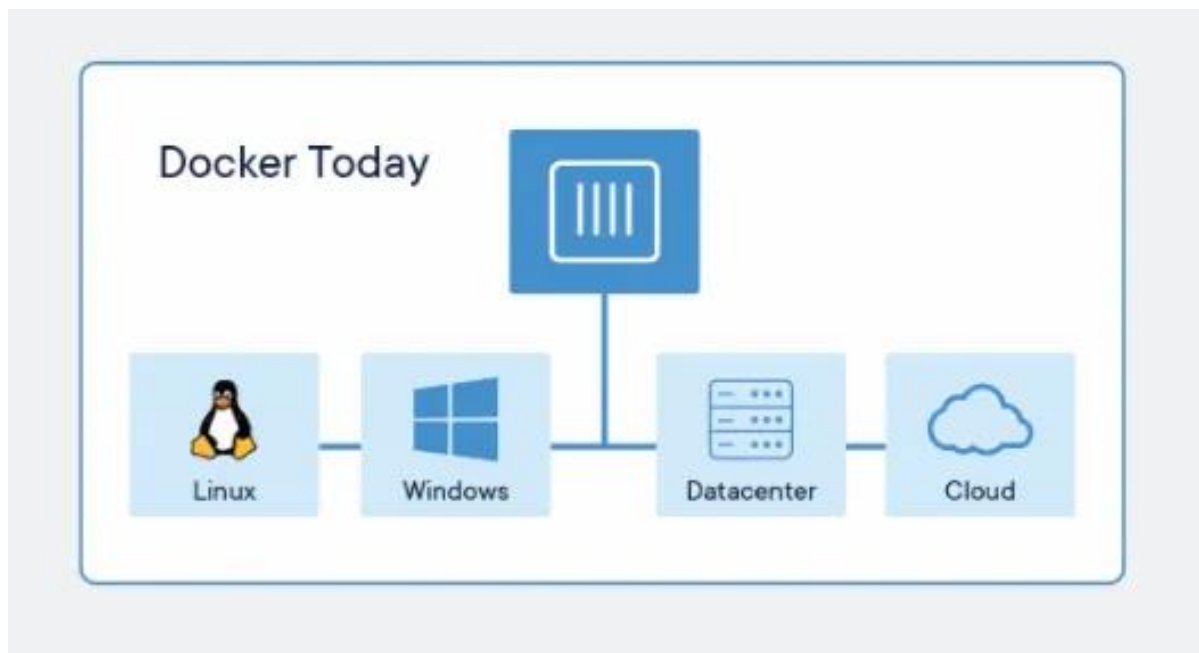


Рис. 11: Платформи, які підтримуються Docker.

Необхідно визначитись і з тим, що собою представляє образ (image) при роботі з Docker. Запущений контейнер використовує ізольовану файлову систему. Ця ізольована файлова система надається за допомогою образу (image), і образ (image) має містити все необхідне для запуску програми – усі залежності, конфігурації, сценарії, двійкові файли і т.д. Образ також містить інші конфігурації для контейнера, такі як змінні середовища, команди за замовчуванням для запуску та інші метадані.

Для роботи із Docker необхідно виконати такі кроки:

1. Встановити Docker на свій комп'ютер. Найкраще для використання буде операційна система із сімейства Linux (Debian, Ubuntu, Fedora та інші.), хоча можна використовувати MacOS чи Windows із WSL (Windows Subsystem Linux);
2. Мати встановлений та налаштований клієнт Git;
3. Мати текстовий редактор на Ваш вибір. На мою думку, краще використовувати Vim, GNU Nano, GNU Emacs або Codium.

Щоб створити образ, необхідно скористатися Dockerfile. Dockerfile — це просто текстовий файл без розширення, який містить перелік інструкцій для виконання. Docker буде використовувати цей скрипт для створення образу контейнера (container).

Для створення Dockerfile необхідно використати таку команду:

- `touch Dockerfile` (в Git Bash не залежно від ОС);
- `New-Item -Path . -Name Dockerfile -ItemType File` (PowerShell);
- `type nul > Dockerfile` (Command Prompt).

Після чого варто додати інформацію в новостворений Dockerfile:

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Для того, щоб зібрати образ необхідно виконати такі команди:

```
cd /path/to/your/directory
docker build -t librevlasfile .
```

Команда збірки `docker` використовує Dockerfile для створення нового образу. Можна помітити, що Docker завантажив багато «шарів». Це виконується через те, що ми зробили запит розробнику, що хочемо почати із зображення `node:18-alpine`. На момент початку цього не було на нашому пристрої, тому Docker потрібну було завантажити зображення з самого початку.

Після того, як Dockerfile був створений та контейнер (container) був зібраний необхідно ще й запустити його. Для запуску застосунку в ізольованому просторі необхідно виконати команду: `docker run -dp 127.0.0.1:3000:3000 librevlasfile`.

Опис використання основних інструкцій при запуску контейнеру (container) в Docker:

- `-d` або `--detach`;

Команда запускає виконання контейнеру (container) у фоновому режимі.

- `-p` або `--publish`.

Команда, яка створює зіставлення портів (port) між host і контейнером (container).

Далі буде наведено короткий огляд використання Docker в тандемі з Kubernetes. В практичній частині магістерської дисертації використання Kubernetes опускається.

Після того, як ми можемо організувати роботу окремих компонентів програми, що працюють в автономних контейнерах, настав час розібратись як можна організувати керування всіма ними за допомогою такого оркестратора, як Kubernetes (K8s). Kubernetes надає багато інструментів для масштабування, роботи в мережі, захисту та підтримки контейнерних програм, які значно перевищують можливості самих контейнерів.

Ми розглянемо базовий опис YAML файлу, які так само, як і для Docker, використовуються для опису дій в Kubernetes для оркестрації контейнерами (container).

Усі контейнери (container) в Kubernetes сприймаються, як модулі, які згруповані, які мають однакове розташування та використовують спільні ресурси. Більш того, у справжній програмі Вам майже ніколи не знадобиться створення окремих модулів. Натомість, більшість ваших робочих навантажень (workloads) заплановані на розгортання, які є масштабованими групами модулів (pod), які автоматично обслуговуються за допомогою Kubernetes.

Нарешті, усі об'єкти Kubernetes можна і потрібно описувати в, так званих, маніфестах, які називаються Kubernetes YAML файлами. Ці файли YAML описують усі компоненти та конфігурації створеної програми Kubernetes і

можуть використовуватися для створення та знищення програми в будь-якому з середовищ Kubernetes.

Перед використанням та першим налаштуванням Kubernetes варто виконати такі передумови:

1. Має бути встановлений Docker;
2. Ви маєте вже вміти користуватись контейнерами: вміти їх створювати та запускати;
3. Необхідно увімкнути можливість використання Kubernetes в застосунку Docker.

Базовий шаблон (template) YAML файлу для налаштування Kubernetes:

```
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
  - name: testpod
    image: alpine:latest
    command: ["ping", "8.8.8.8"]
```

Цей YAML файл буде описувати створення групи з контейнером, який ізольований лише для ping 8.8.8.8.

Після цього необхідно в терміналі перейти до файлу pod.yaml. Варто тепер виконати запуск вашої групи (pod) за допомогою такої команди:

```
kubectl apply -f pod.yaml
```

Тепер ми дійшли до того, що Kubernetes має працювати і варто перевірити чи він запущений. Це можна зробити такою командою:

```
kubectl get pods
```

В терміналі маємо отримати приблизно такий результат:

NAME	READY	STATUS	RESTARTS	AGE
demo	1/1	Running	0	4s

Якщо не спрацювало, то варто перевірити чи точно був увімкнений Kubernetes в налаштуваннях Docker.

Надалі можна перевірити всі виконані події командою:

```
kubectl logs demo
```

Результат має бути приблизно таким:

```
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=37 time=21.393 ms
64 bytes from 8.8.8.8: seq=1 ttl=37 time=15.320 ms
64 bytes from 8.8.8.8: seq=2 ttl=37 time=11.111 ms
...
```

Після закінчення роботи можна знищити нашу групу:

```
kubectl delete -f pod.yaml
```

Параметр `-f` означаєш шлях до файлу конфігурації Kubernetes з якого треба видалити виділені ресурси.

2.3. Використання хмарного середовища на прикладі Google Cloud Platform (GCP)

Для повноцінного використання хмарного середовища Google Cloud Platform (GCP) необхідно знати і розуміти, як загалом все влаштовано в ньому.

Ресурси, які містить Google Cloud Platform.

Google Cloud складається з набору фізичних об'єктів (комп'ютери та жорсткі диски), і віртуальних ресурсів (віртуальні машини (VM), які розташовані в центрах обробки даних (data center) по всьому світу). Кожен регіон являє собою сукупність зон, які ізольовані між собою в межах одного регіону.

Ресурси Google Cloud Platform можна поділити на такі категорії:

1. Глобальні;
2. Регіональні;
3. Зональні.

Деякі ресурси можуть бути доступні з будь-якого іншого ресурсу в різних регіонів і зон. Ці глобальні ресурси (global resources) включають попередньо налаштовані образи дисків, зображення дисків та мережі. Доступ до деяких ресурсів можуть мати лише ресурси, розташовані в тому самому регіоні. Ці регіональні ресурси (regional resources) містять статичні зовнішні IP-адреси. Доступ до інших ресурсів можуть мати тільки ресурси, які знаходяться в тій же зоні. Ці зональні ресурси включають, наприклад, екземпляри віртуальних машин (VM), їх типи та диски.

Обсяг операції залежить від типу ресурсів, з якими ви працюєте. Наприклад, створення мережі є глобальною операцією, оскільки мережа є глобальним ресурсом, тоді як резервування IP-адреси є регіональною операцією, оскільки адреса є лише регіональним ресурсом.

На діаграмі (рис. 12) показано зв'язок між глобальними, регіональними та зональними ресурсами:

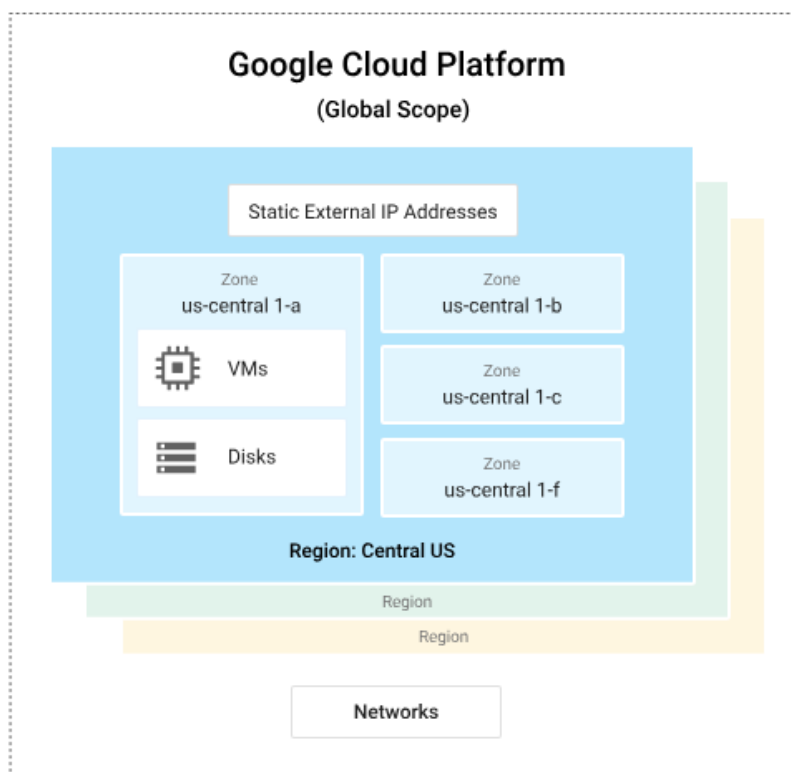


Рис. 12: Різновиди ресурсів GCP.

Проекти на Google Cloud Platform.

Усі ресурси Google Cloud, які будуть виділені та використані, мають належати до певного проекту. Проект складається з налаштувань, дозволів та інших метаданих, які описують принцип роботи програмного забезпечення розташованого в хмарному середовищі. Ресурси в рамках одного проекту можуть легко працювати разом, наприклад, спілкуючись через внутрішню мережу відповідно до правил регіонів і зон.

Кожен проект створений на Google Cloud Platform має мати:

1. Назва проекту, яку створює користувач;
2. Ідентифікатор проекту, який ви можете надати або Google Cloud може надати вам;
3. Номер проекту, який надає Google Cloud.

В нашому випадку проект має такі параметри опису (рис. 13):

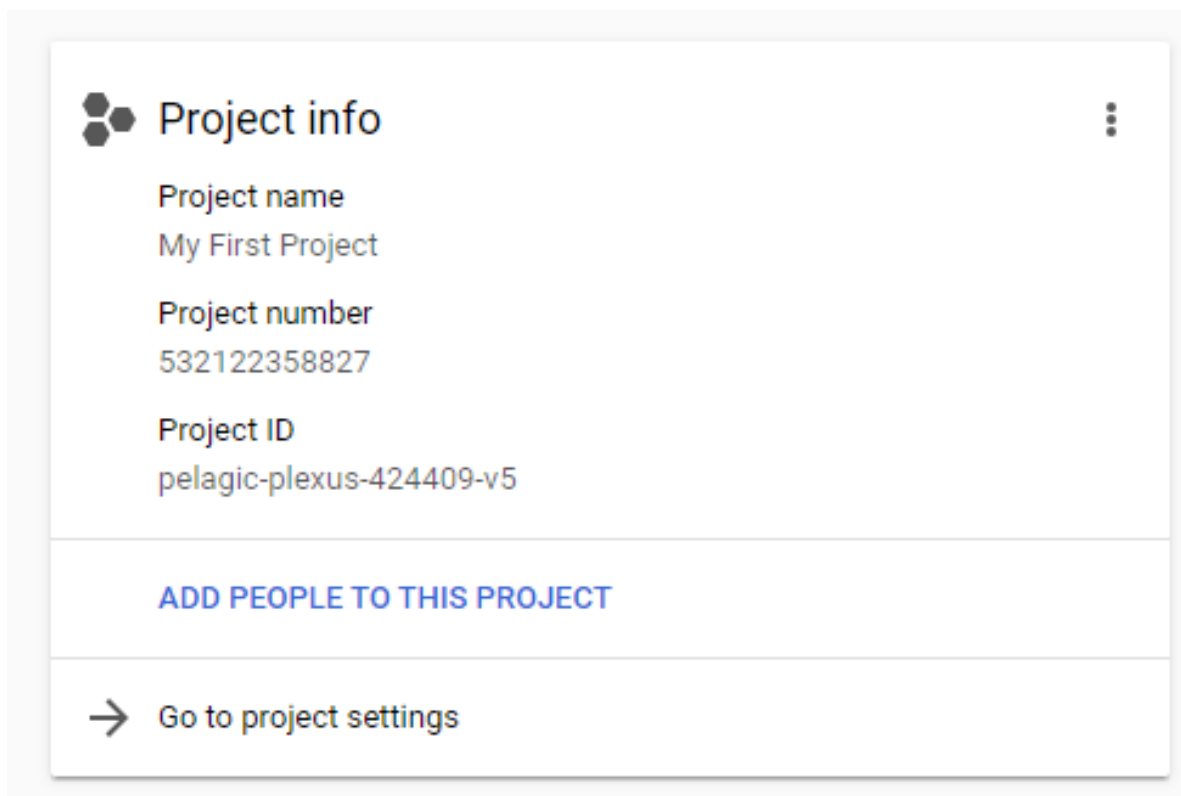


Рис. 13: Опис параметрів проекту в GCP.

Кожен ідентифікатор проекту є унікальним в Google Cloud Platform. Після того, як проект створено, то є можливість видалити проект, але його ідентифікатор більше ніколи не буде використовуватися.

Ви можете створити кілька проектів і використовувати їх, щоб розділити свою роботу будь-яким доцільним для вас способом. Наприклад, у вас може бути один проект, доступ до якого мають усі члени команди, і окремий проект, до якого можуть отримати доступ лише певні члени команди.

Взаємодіяти з Google Cloud Platform можна трьома способами:

1. Графічний інтерфейс (GUI);
2. Термінальний інтерфейс взаємодії (CLI);
3. Клієнтські бібліотеки.

Ми будемо використовувати саме перший варіант взаємодії, бо його можливостей більш, ніж достатньо для наших задач.

Google Cloud Platform Dashbord (рис. 14) надає можливості перегляду основних показників на спеціальній панелі, яку можна налаштовувати під свої потреби.

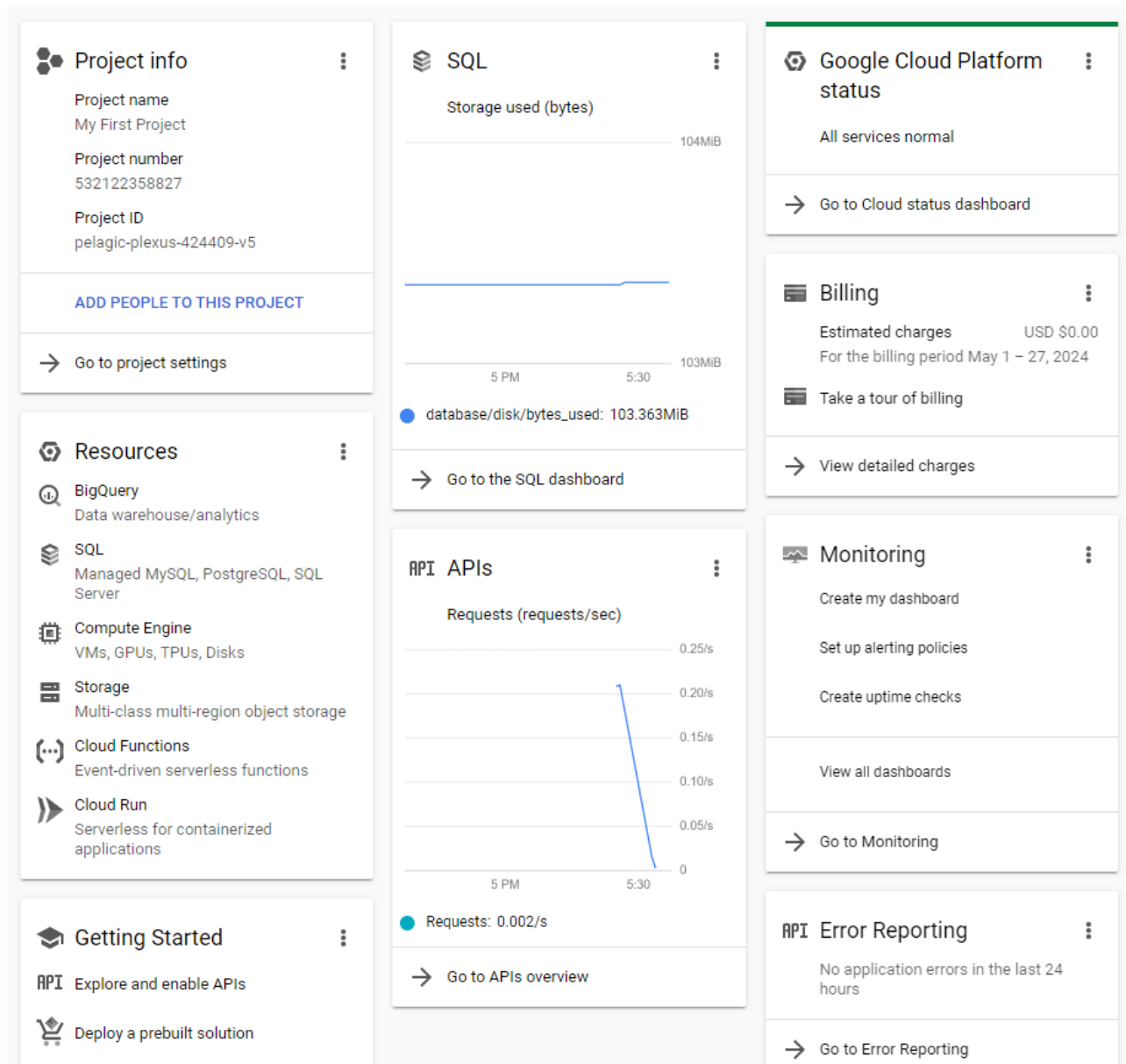


Рис. 14: Панель стану проекту в GCP.

Всі інструменти, які пропонує Google Cloud Platform можна поділити на такі категорії:

- Обчислення (Compute);
- Дані та їх збереження в БД (Storage та Databases);
- Мережевий зв'язок (Networking);

- Аналітика (Analytics);
- Машинне навчання (Machine Learning);
- Операції (Operations);
- Безпека (Security);
- API (Application Programming Interface).

Розглянемо основні інструменти, які Google Cloud Platforma пропонує для використання:

- Cloud Build;
Це служба, яка виконує збірку на Google Cloud Platform.
- Google Cloud Deploy;
Це служба, яка автоматизує доставку програм у цілий перелік цільових середовищ у визначеній розробником послідовності.
- Cloud Storage;
Це керований сервіс для зберігання неструктурованих даних. Зберігайте будь-яку кількість даних і відновлюйте їх так часто, як забажаєте.
- Google Kubernetes Engine (GKE);
Це найбільш масштабований і повністю автоматизований сервіс використання Kubernetes.
- Cloud Run;
Це сервіс, який надає можливість запускати frontend та backend сервіси, пакетні завдання, розгортати веб-сайти та програми та обробляти робочі навантаження в черзі без необхідності керувати інфраструктурою.
- Cloud SQL.
Повністю керована, економічно ефективна служба реляційної бази даних (БД) для PostgreSQL, MySQL і SQL Server.

Детальний огляд останнього інструменти буде виконано в наступному розділі, бо він буде безпосередньо використовуватись в практичній частині цієї магістерської дисертації.

2.4. Можливості та потреба використання сервісу Google Cloud SQL

В цій магістерській дисертації буде використано Google Cloud SQL для зберігання інформації про користувачів.

Google Cloud SQL надає хмарну альтернативу локальним базам даних MySQL, PostgreSQL і SQL Server. Необхідно використовувати Cloud SQL, якщо є потреба витрачати менше часу на керування базою даних (database) і більше часу на її використання.

Багато програм, що працюють на Compute Engine, App Engine та інших службах Google Cloud, використовують Cloud SQL для зберігання бази даних.

Google Cloud SQL має свою панель стану (рис. 15), яка аналогічна до Google Cloud Platform Dashboard. Основні показники можна побачити саме на цій панелі.

Cloud SQL може запропонувати багато послуг, тож немає необхідності створювати та підтримувати їх власноруч. Можна зосередитися на своїх даних і дозволити Cloud SQL обробляти такі операції, як:

- Резервне копіювання;
- Висока доступність;
- Відновлення після відмови;
- Мережеве з'єднання;
- Імпортування та експортування даних;
- Технічне обслуговування;
- Моніторинг;
- Журнали звітів (logging).

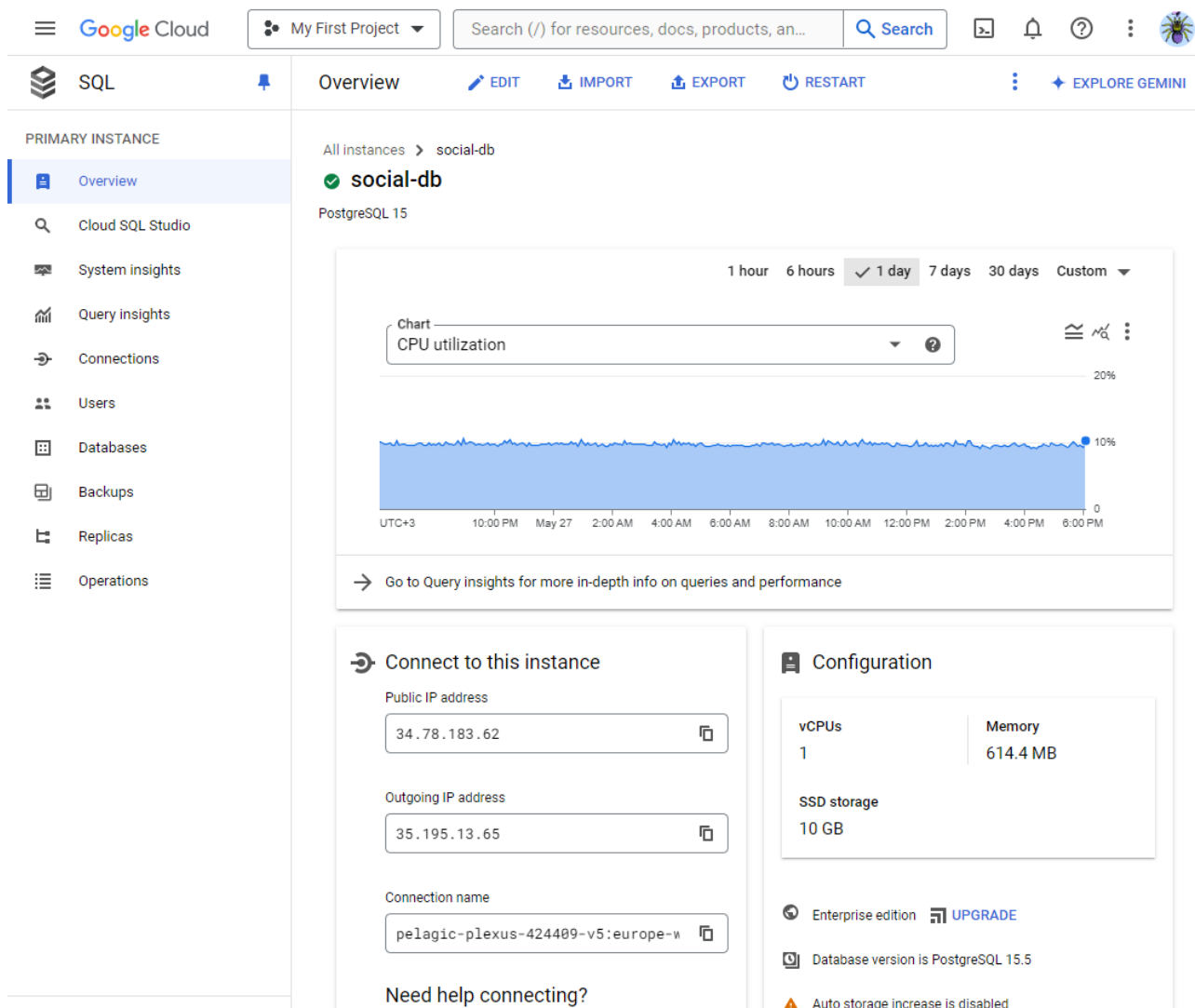


Рис. 15: Панель стану Google Cloud SQL.

Кожен екземпляр (instance) Cloud SQL підтримується віртуальною машиною (VM), яка працює на host-сервері Google Cloud. Кожна віртуальна машина керує програмою бази даних (database), такою як MySQL Server, PostgreSQL або SQL Server, і сервісними агентами (agent), які надають допоміжні служби, такі як журналювання (logging) та моніторинг (monitoring). Параметр високої доступності також забезпечує резервну віртуальну машину в іншій зоні з конфігурацією, ідентичною основній віртуальній машині.

Cloud SQL дозволяє створювати та видаляти бази даних (БД) і користувачів баз даних, але це не повноцінний інструмент адміністрування бази даних.

Існує багато інструментів адміністрування бази даних (БД), з яких можна обрати потрібну, залежно від механізму вашої бази даних, зокрема такі:

1. phpMyAdmin для MySQL;
2. MySQL Workbench для MySQL;
3. Toad Edge для MySQL та PostgreSQL;
4. pgAdmin.org для PostgreSQL;
5. SQL Server Management Studio для SQL Server;
6. Visual Studio Code для SQL Server.

Принцип роботи екземпляра (instance) в Cloud SQL зображено на рис. 16.

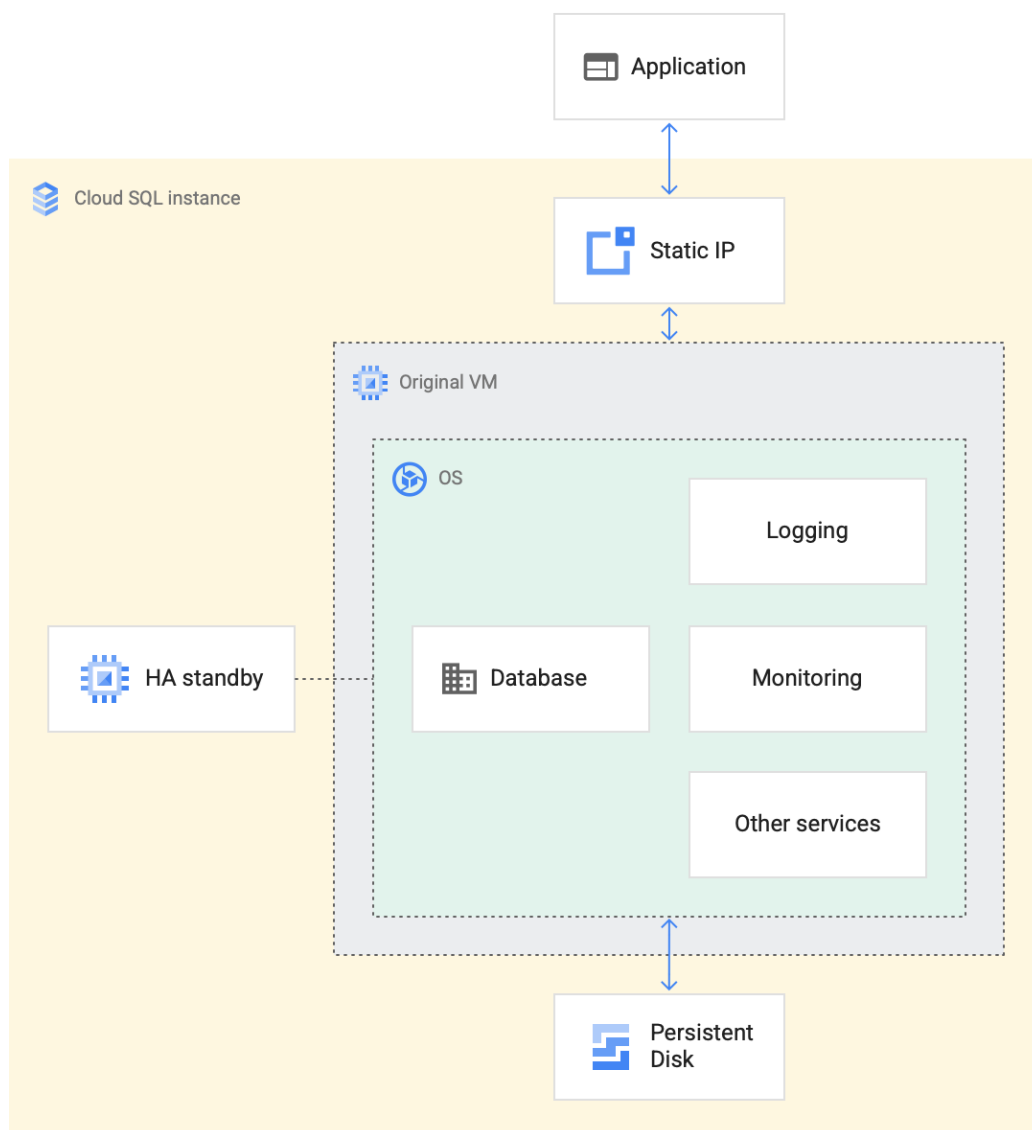


Рис. 16: Схема роботи екземпляра в Cloud SQL.

2.5. Моніторинг системи на Google Cloud Platform

Ця категорія (моніторингова) служб може допомогти зрозуміти поведінку, стан і продуктивність програм та інших служб Google Cloud під час роботи. Cloud Monitoring автоматично збирає та зберігає інформацію про продуктивність переважної кількості служб Google Cloud. Є змога обирати показники та індикатори Prometheus за допомогою Google Cloud Managed Service для Prometheus.

Хмарний моніторинг надає різні служби, які можна використовувати, щоб оцінити працездатність і продуктивність програм, а також більшості частини інших служб Google Cloud, які будуть в користуванні.

Сповіщення систем моніторингу.

Щоб отримувати сповіщення, коли значення показники ефективності відповідають потрібним критеріям, які визначаєні Вами, необхідно створити створити правила надходження сповіщень (alerts). Цей перелік правил сповіщень має містити набір людей або груп, які мають отримувати сповіщення. Моніторинг підтримує загальні канали сповіщень, зокрема електронну пошту, хмарний мобільний додаток і такі застосунки, як Slack. До прикладу, можна створити правила попередження, щоб Вам надходило сповіщення, коли використання віртуального центрального процесора (vCPU) машини перевищує 80%.

Кожне сповіщення містить відповідну інформацію про збій і містить посилання на інцидент. Інцидент (incident) — це постійний нотування, яке зберігає всю інформацію, яку можна згодом використовувати для повного усунення несправності. Як правило, до така запису входить перелік статусів інциденту, посилання на журнали, діаграму записаних метричних даних, мітки та тривалість подій.

Служба оповіщення (alert service) інтегрована в більшість служб Google Cloud Platform (GCP). Якщо ці інтеграції існують, то є змога побачити панель зі списком рекомендованих сповіщень або кнопку на діаграмі, яка дозволяє створити правила для сповіщень. Незалежно від способу, правила надходження сповіщень попередньо мають бути налаштовані.

Можна створювати правила надходження сповіщень і керуванням ними за допомогою консолі Google Cloud, Cloud Monitoring API, Google Cloud CLI або Terraform.

Різновиди інформації, яку збирає та зберігає моніторинговими система Cloud Monitoring:

- Системні показники, створені службами Google Cloud;
Ці показники надають інформацію про те, як працює служба.
- Показники системи та додатків, які збирає Ops Agent про системні ресурси та програми, запущені в екземплярах Compute Engine;
- Визначені користувачем показники;
Такі показники створюються користувачем за допомогою API хмарного моніторингу або за допомогою бібліотеки (OpenTelemetry).
- Зовнішні показники, визначені деякими бібліотеками з відкритим вихідним кодом або сторонніми постачальниками;
- Показники Prometheus, які збираються керованою службою Google Cloud для Prometheus або за допомогою агента операцій і приймача Prometheus.
- Метрики на основі журналів подій.

Такі метрики записують числову інформацію про журнали в Cloud Logging. Такі журнали включають в себе кількість помилок, які виявляє сервіс, і загальну кількість записів журналу, отриманих проектом Google Cloud.

Щоб візуалізувати моніторингові дані, побачити тенденції, виявити проблеми або переглянути інші деталі, можна використовувати інформаційну панель (dashboard) і службу створення діаграм (charting service).

- Google Cloud автоматично створює панель Google Cloud-managed Dashboard, коли додаєте новий ресурс до проекту Google Cloud;
- Сервіс для діаграм, Metrics Explorer, дозволяє швидко візуалізувати та досліджувати дані по часу. Налаштувавши діаграми є змога порівнювати поточні дані з попередніми, відображати проблеми, а також відображати кілька показників одночасно.

Приклад роботи моніторингової системи середнього проекту можна побачити на рис. 17.

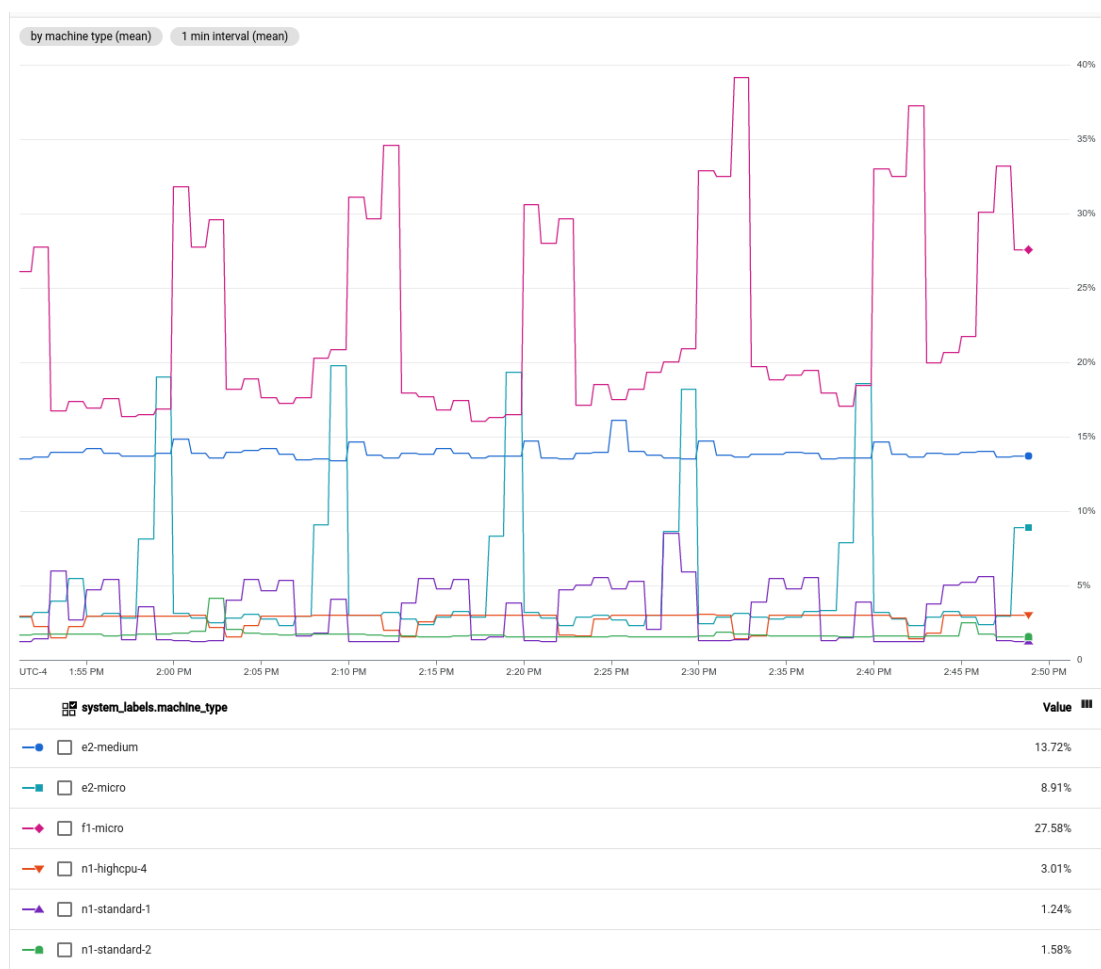


Рис. 17: Моніторинг стану системи в GCP.

Висновки до другого розділу

Використання сучасних інструментів для автоматизації рутинних завдань та хмарних технологій дозволяє відчутно покращити процес розробки, тестування та розгортки програмного забезпечення.

Одним з ключових інструментів такого процесу є GitHub Actions, який забезпечує неперервну інтеграцію та безперервне розгортання (CI/CD). Цей інструмент дозволяє автоматизувати різноманітні завдання, такі як збірка коду, запуск тестів та випуск оновлень. Завдяки GitHub Actions розробники тепер можуть швидко та ефективно проводити перевірку своїх змін, гарантуючи високу стабільність та якість продукту на кожному етапі розробки програмного забезпечення.

Паралельно з GitHub Actions, Docker займає вкрай важливу роль у створенні та управлінні ізольованих контейнерами (container), що значно спрощує процеси розгортки програмного забезпечення та масштабування додатків за потреби. Docker надає можливість упакувати застосунок з усіма його залежностями в один контейнер (container), який можна запускати в будь-якому середовищі з повною впевненістю на його однаковість роботи незалежно від середовища. Це особливо зручним стає для розробників, які можуть з впевненістю переносити свої застосунки між локальними машинами, серверами для тестування та хмарними платформами.

Хмарний провайдер Google Cloud Platform (GCP) надає широкий спектр хмарних сервісів, які доповнюють вже обрані GitHub Actions та Docker. GCP може забезпечити високу надійність, потрібну масштабованість та безпеку для застосунків всіх рівнів складності. Використання GCP дозволяє на практиці командам зосередитися саме на розробці програмного забезпечення, а не на управлінні всією інфраструктурою. Вкрай важливим для практичного проекту впровадження DevOps практик у веб-застосунок блогу є компонент Google Cloud

SQL, який надає можливість керувати базами даних для застосунків. Cloud SQL забезпечує доволі високу продуктивність, автоматичне резервне копіювання та можливість до масштабування баз даних, що дозволяє зосередитися на інших процесах замість управління базами даних.

Моніторингова система Google Cloud також займає значну роль у підтримці стабільності та продуктивності застосунків. Ця система забезпечує гарний інструментарій для моніторингу стану системи, збору показників (метрик), логів (logs) та налаштування сповіщень про стан системи. Це дозволяє оперативно реагувати на будь-які проблеми та забезпечувати високу доступність сервісів. Завдяки інтеграції з іншими сервісами GCP, моніторингова система дозволяє автоматизувати процеси швидкого відновлення та оптимізації, що підвищує ефективність управління застосунками.

Отже, використання GitHub Actions, Docker, Google Cloud Platform та її сервісів, таких як Google Cloud SQL та моніторингової системи, створює зручну та доволі потужну систему для сучасної розробки програмного забезпечення. Ці інструменти забезпечать неперервну інтеграцію та розгортання, надійне та ефективне управління додатками, а також їх моніторинг і та здаість до масштабування за потреби. Як результат, команда розробників отримає можливість швидше та більш якісно поставляти продуктової оновлення користувачам, із зниженням витрат на інфраструктуру та підвищуючи загальну продуктивність всього процесу розробки.

3. БЕЗПОСЕРЕДНЯ РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ТА ВПРОВАДЖЕННЯ АВТОМАТИЗАЦІЇ РУТИННИХ ЗАВДАНЬ DEVOPS

В сучасному світі розробка програмного забезпечення хоча б якогось рівня складності вимагає не тільки створення для користувача зручного продукту наповненого різноманітним функціоналом, але й забезпечення надійної та ефективної інфраструктури для його підтримки та подальших оновлень. Реалізація блогу, як веб-застосунку передбачає інтеграцію компонентів, які були детально розглянуто в перших двох розділах цієї магістерської дисертації. Однак, для забезпечення стабільної роботи та швидкого розгортання програмного забезпечення, необхідно також впроваджувати автоматизацію рутинних завдань DevOps. В практичній частині магістерської дисертації буде виконано імплементацію неперервного інтеграції та безперервного розгортання (CI/CD) на основні сервісу GitHub Actions, контейнеризацію застосунку, підключення хмарного сховища даних та моніторинг роботи системи.

Завдяки таким підходам можна буде значно підвищити ефективність процесу розробки, забезпечити швидке виявлення та виправлення помилок, а також мінімізувати ризики, пов'язані з ручним втручанням в процес розгортки застосунку та управління всією інфраструктурою. Впровадження таких DevOps практик є критичним кроком для команд, які прагнуть створювати сучасні, надійні веб-застосунки здатні до масштабування за потреби.

3.1. Створення веб-застосунку та опис принципу його роботи

Реалізація автоматизації рутинних завдань DevOps методології буде виконана на застосунку, що по свої суті являє простий backend серів з API для блогу.

Перелік та огляд технології, які були використанні для створенні веб-застосунку:

1. Python3;

Мова програмування, яка добре інтегрована з Django фреймворком.

2. Django;

Високорівневий веб-фреймворк мовою програмування Python, який надає можливість швидко створювати backend веб-застосунків з можливістю потрібного масштабування.

3. Django REST Framework;

Потужний та доволі гнучкий інструмент для створення веб-API, який заснований на фреймворці Django.

4. Djoser;

Бібліотека для фреймворку Django, яка забезпечує можливість зручного інтерфейсу для роботи з аутентифікацією користувачів через RESTful API.

5. Simple JWT.

Бібліотека для Django, яка забезпечує підтримку аутентифікацію за допомогою JSON Web Tokens (JWT). Ця бібліотека побудована на основі Django REST Framework.

Функціонал реалізований в цьому застосунку:

1. Створення користувача (реєстрація користувача);
2. Обліковий запис користувача (логін користувача);
3. Створення публікацій;
4. Залишити реакцію “Подобається” під дописом;
5. Аналітика кількості реакцій під дописом;

Необхідно використовувати параметри `date_from` та `date_to` для фільтрації за часовим показником.

6. Endpoint з інформацією про активність користувача.

Інформація є про останній вхід і час останнього відвідування.

В таблиці №2 можна знайти перелік всіх використаних endpoint:

Action	URL	Method
Create user	/api/v1/auth/users/	POST
User activity	/api/v1/auth/activity/	GET
Obtain token	/api/v1/auth/token/	POST
Refresh token	/api/v1/auth/refresh/	POST
Create post	/api/v1/posts/	POST
List of posts	/api/v1/posts/	GET
Get post	/api/v1/posts/int:pk	GET
Like post	/api/v1/posts/int:pk/like/	POST
Unlike post	/api/v1/posts/int:pk/unlike/	POST
Like analytics	/api/v1/analytics/	GET

Табл. 2: Перелік всіх endpoint.

На зображенні (рис. 18) продемонстровано інтерфейс входу в панель адміністратора цього застосунку:

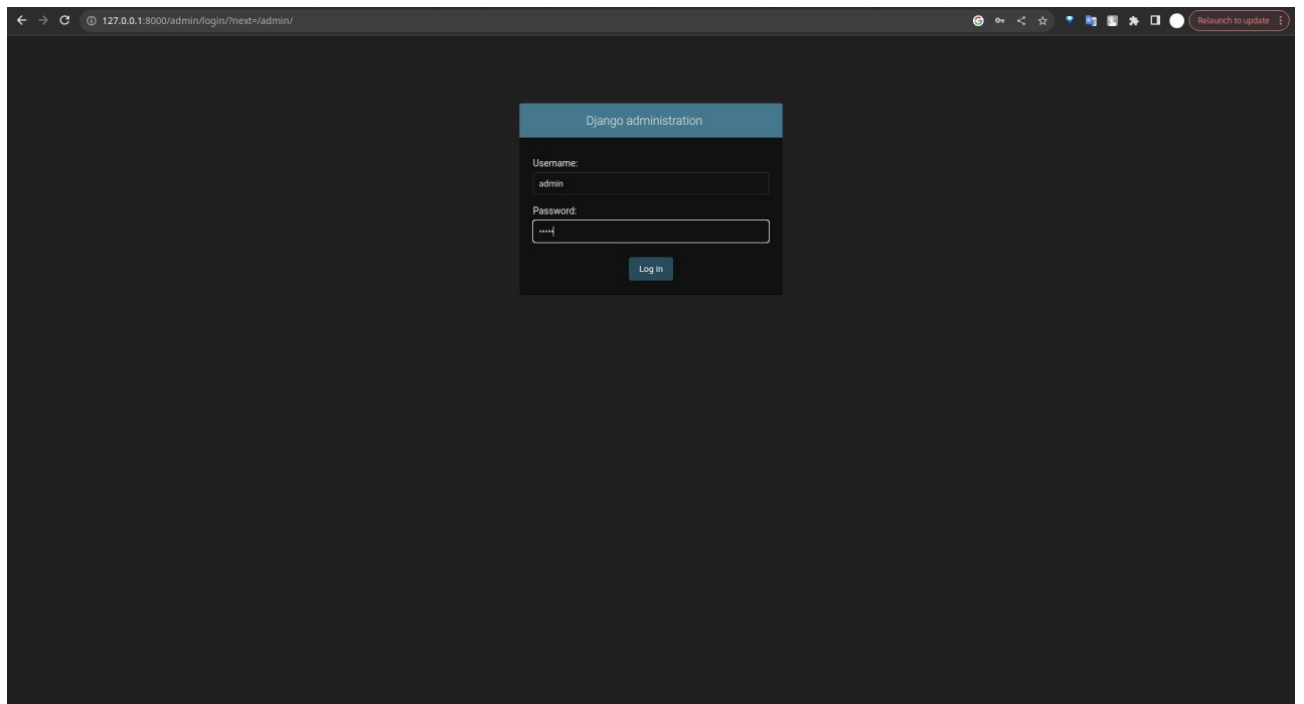


Рис. 18.

Зображення (рис. 19) демонструє адміністративну панель із переліком всіх сутностей:

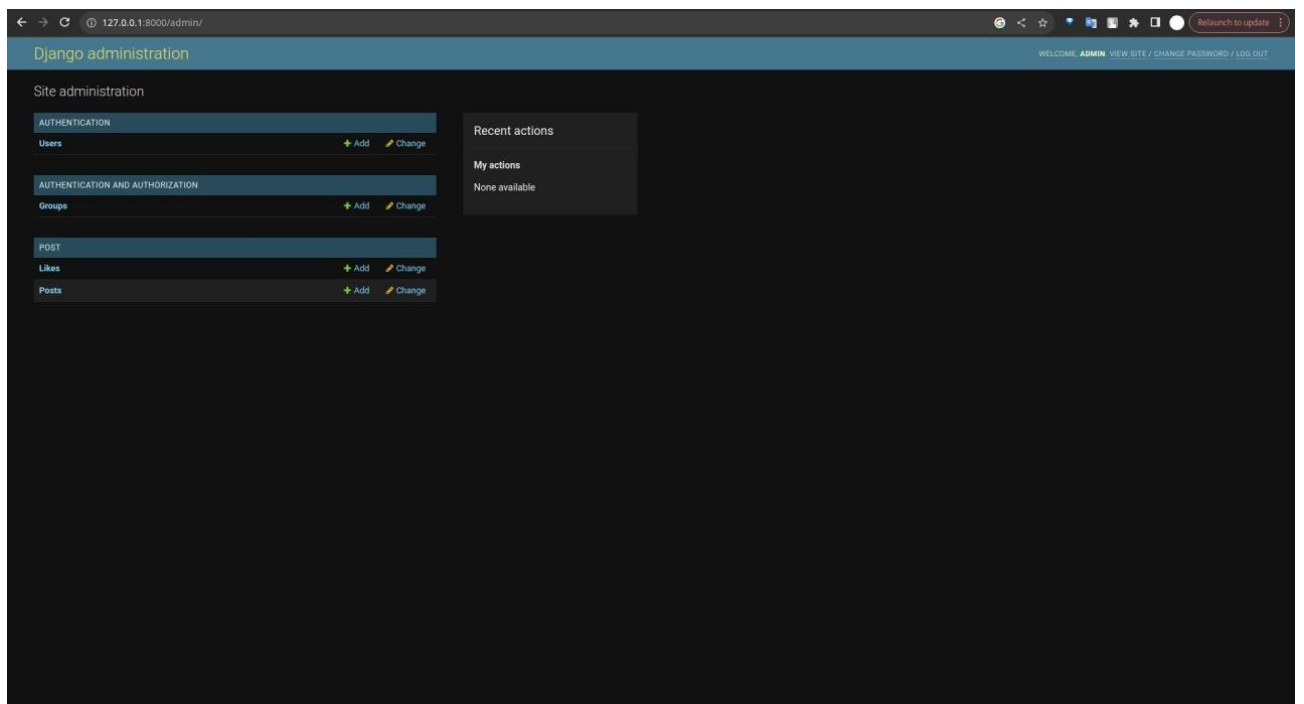


Рис. 19.

Зображення (рис. 20) демонструє список створених статей в сервісі та демонструє інтерфейс для керування ними:

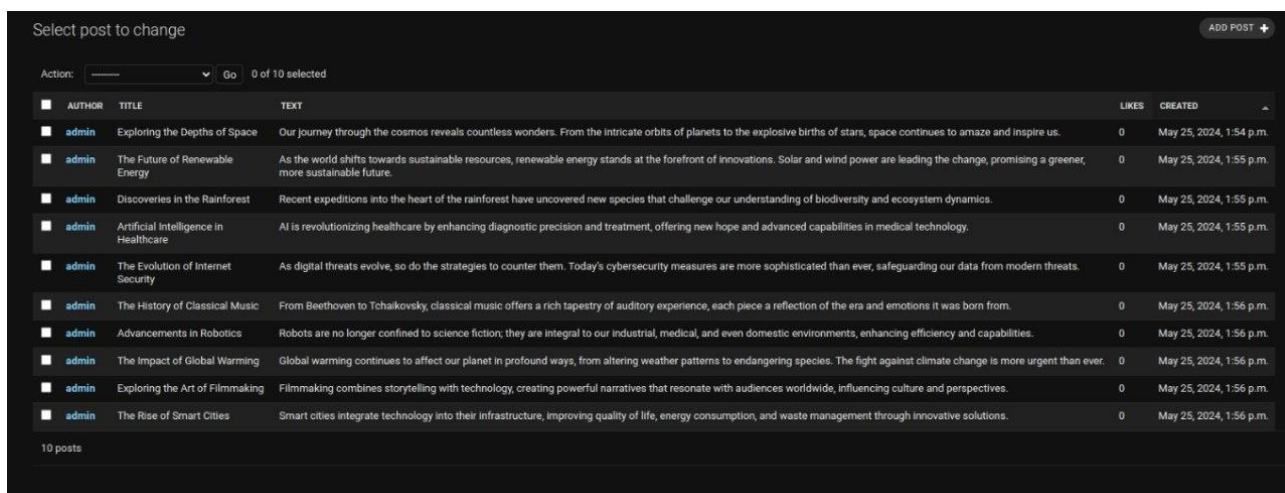


Рис. 20.

Для авторизації користувачів використовується одержання JWT tokenів використовуючи нашу API (авторизація), що зображено на рис. 21.

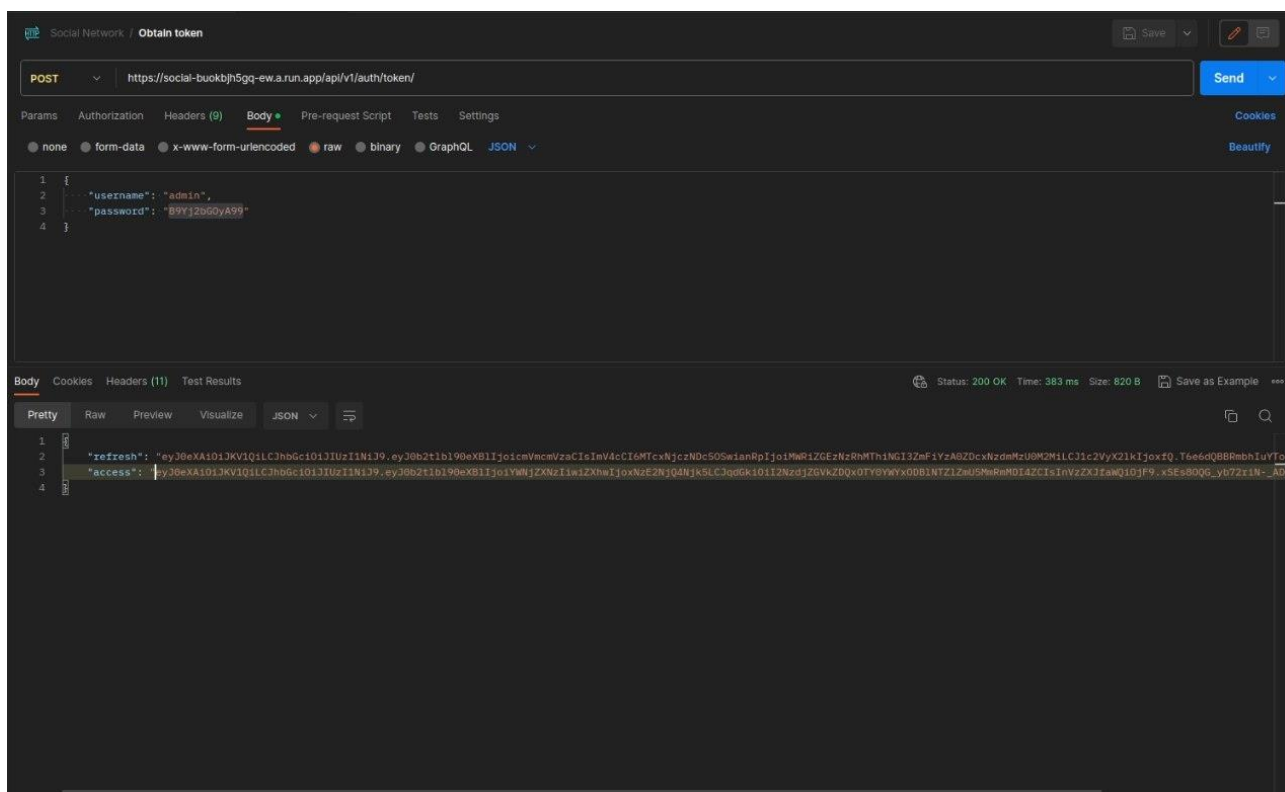


Рис. 21.

Перелік публікацій, що присутні в базі даних можна побачити на рис. 22.

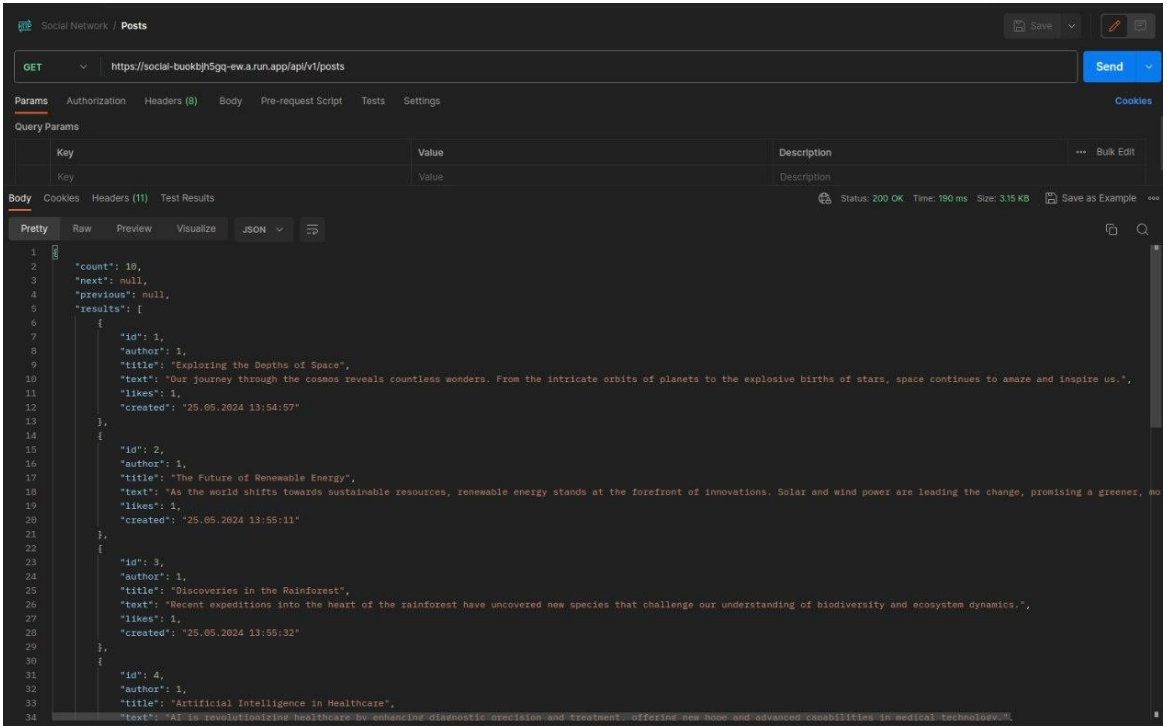


Рис. 22.

Кожна публікація собою представляє набір значень (id, author, title, text, likes, created), що продемонстровано на рис. 23.

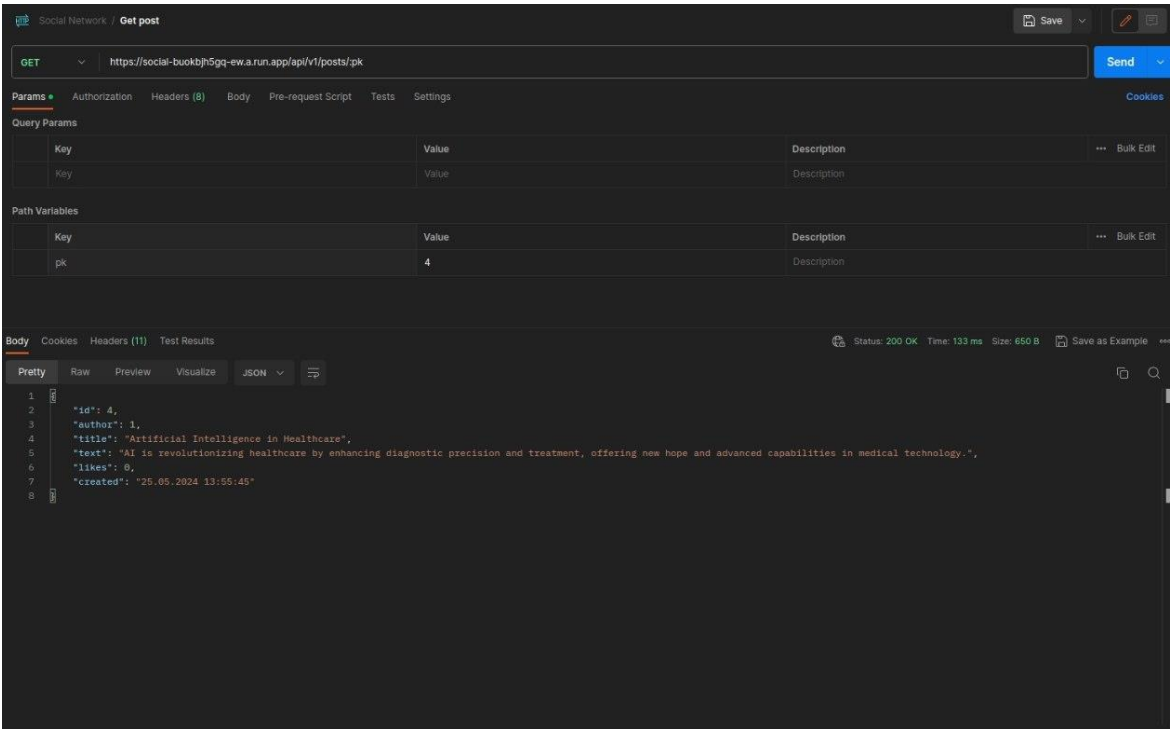


Рис. 23.

Аналітика вподобань публікацій реалізована наступним чином (рис. 24).

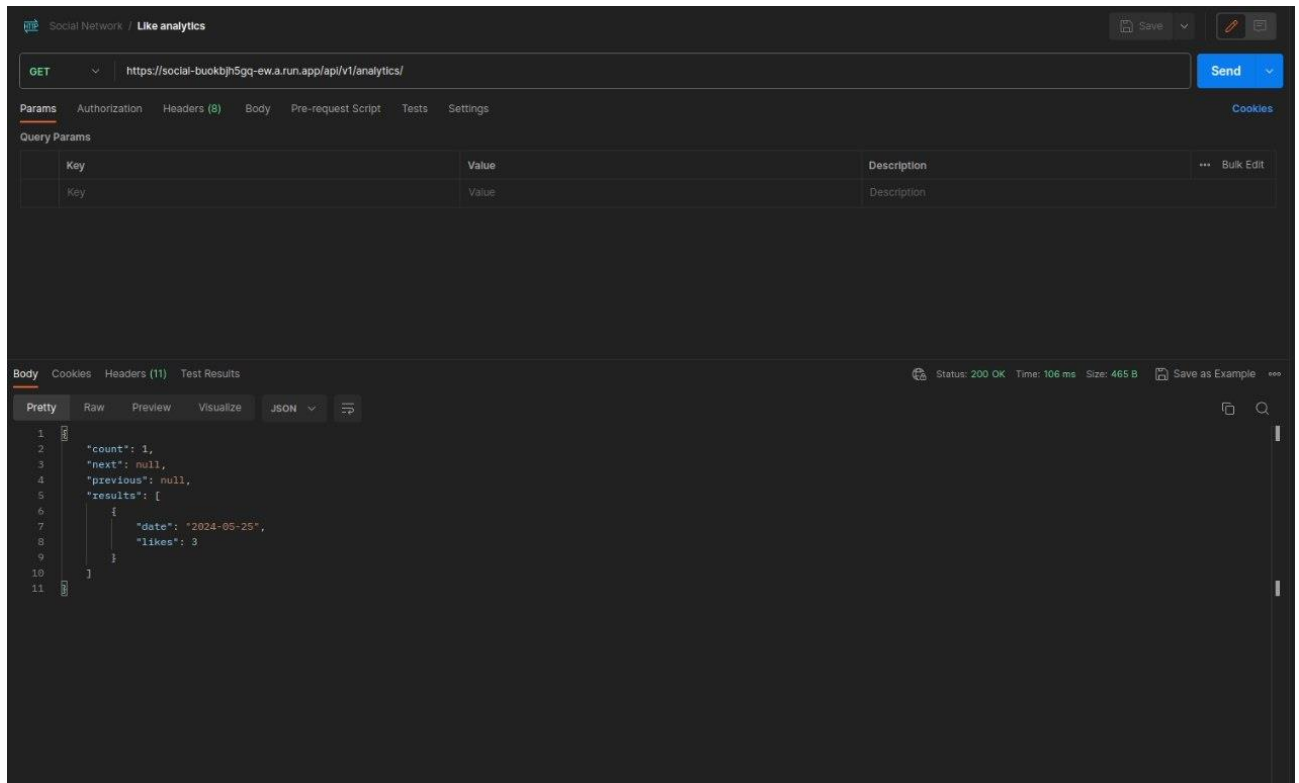


Рис. 24.

3.2. Використання Docker для розробки веб-застосунку

Тепер необхідно реалізувати контейнеризацію веб-застосунку за допомогою Docker для того, щоб пришвидшити розгортку застосунку та покращити рівень його безпеки.

Для того, щоб застосунок правильно та оптимально був запакований в контейнер створимо Dockerfile з відповідним змістом (рис. 25).

```

1  FROM python:3.8-slim-buster
2
3  ENV PYTHONDONTWRITEBYTECODE 1
4  ENV PYTHONUNBUFFERED 1
5
6  WORKDIR /app
7
8  RUN apt-get update && apt-get install -y --no-install-recommends gcc \
9      && apt-get clean \
10     && rm -rf /var/lib/apt/lists/*
11
12  COPY requirements.txt /app/
13  RUN pip install --upgrade pip && \
14     pip install -r requirements.txt
15
16  COPY . /app/
17
18  ENTRYPOINT ["sh", "entrypoint.sh"]

```

Рис. 25: Зміст Dockerfile.

Далі буде наведено пояснення кожного кроку в Dockerfile (табл. 3) для розуміння того, як саме наш проект зібраний до купи.

Опис виконання кожного кроку в Dockerfile.		
FROM	python:3.8-slim-buster	Вказівка, що наш образ побудований на образі Python3.8 в мінімальній конфігурації.
ENV	PYTHONDONTWRITEBYTECODE 1 ENV PYTHONUNBUFFERED 1	Це інструкції для встановлення змінних оточення, які змушують миттєва видавати логи системи.
WORKDIR	/app	Робоча директорія всередині самого контейнеру.

Табл. 3.

<pre> RUN apt-get update && apt- get install -y --no-install- recommends gcc \ && apt-get clean \ && rm -rf /var/lib/apt/lists/* </pre>	Команди для оновлення компонентів, встановлення компілятора GCC та очищення системи від сміття.
<pre> COPY requirements.txt /app/ </pre>	Копіювання файлу необхідних бібліотек для роботи.
<pre> RUN pip install --upgrade pip && \ pip install -r requirements.txt </pre>	Встановлення всіх необхідних залежностей.
<pre> COPY . /app/ </pre>	Копіювання всього вихідного коду програму в контейнер.
<pre> ENTRYPOINT ["sh", "entrypoint.sh"] </pre>	Команда, яка не буде виконуватись під час зборки контейнеру. Вона спрацює лише після запуску зібраного контейнеру.

Табл. 3.

Під час запуску контейнеру інструкція Dockerfile буде виконувати bash скрипт (рис. 26) з файлу `entrypoint.sh`, який виконає міграцію бази даних.

```

1  #!/bin/sh
2  python manage.py migrate --no-input
3  python manage.py collectstatic --no-input \
4  && python manage.py runserver 0.0.0.0:8000

```

Рис. 26.

3.3. Створення та налаштування Google Cloud SQL та GCP Artifact Registry

Наступні декілька зображень будуть демонструвати процес створення бази даних з використанням інструменту Google Cloud SQL.

Процес створення бази даних із назвою «social-db» (рис. 27).

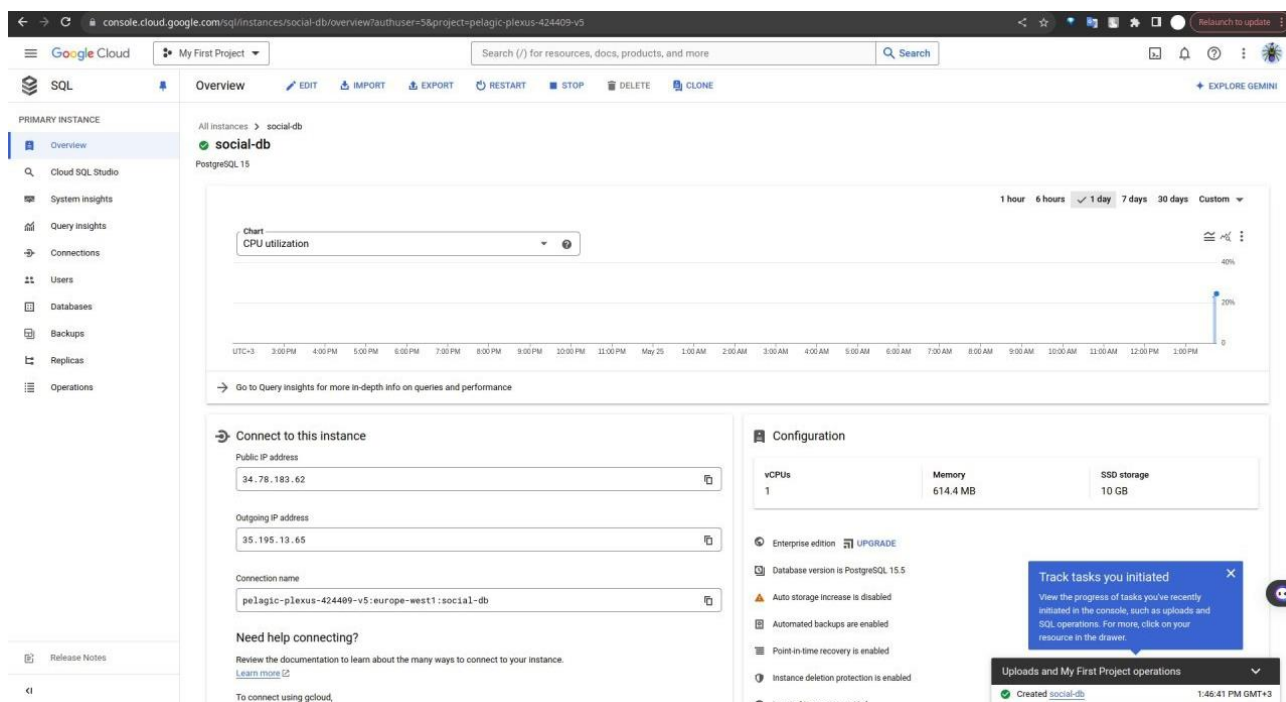


Рис. 27: Створення бази даних.

Демонстрація процесу налаштування умов використання створеної бази даних з попереднього кроку (рис. 28-29).

console.cloud.google.com/sql/instances/create?engine=PostgreSQL&authuser=5&project=pelagic-plexus-424409-v5

Google Cloud My First Project Search (/) for resources, docs, products, and more Search

Create a PostgreSQL instance

Region
europe-west1 (Belgium)

Zonal availability
☒ **Single zone**
 In case of outage, no failover. Not recommended for production.
☐ **Multiple zones (highly available)**
 Automatic failover to another zone within your selected region. Recommended for production instances. Increases cost.

Primary zone
Any

[HIDE ZONES](#)

Customize your instance

You can also customize instance configurations later.

Machine configuration

Machine shapes
Choose from the most commonly used shapes or choose a custom shape. Make sure your selection has enough memory to hold your largest table.

Shared core

☒ 1 vCPU, 0.614 GB
☐ 1 vCPU, 1.7 GB

Data cache
Exclusive to Enterprise Plus edition. Adds a local SSD for read caching to reduce read latency and improve performance at additional cost. [Learn more](#)

☐ Enable data cache

Storage
Storage type is SSD. Storage size is 250 GB, and will automatically scale as needed. Google-managed key enabled (most common).

Pricing estimate

\$0.07 per hour (estimated, without discounts)
That's about \$1.65 per day.
Feature usage and traffic costs aren't included in estimate.

[SHOW COST BREAKDOWN](#)

Summary

Cloud SQL Edition	Enterprise
Region	europe-west1 (Belgium)
DB Version	PostgreSQL 15
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	250 GB
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled
Network throughput (MB/s)	125 of 125
Disk throughput (MB/s)	Read: 120.0 of 125.0 Write: 107.8 of 107.8
IOPS	Read: 7,500 of 12,000 Write: 7,500 of 10,000

Puc. 28.

console.cloud.google.com/sql/instances/create?engine=PostgreSQL&authuser=5&project=pelagic-plexus-424409-v5

Google Cloud My First Project Search (/) for resources, docs, products, and more Search

Create a PostgreSQL instance

Region
europe-west1 (Belgium)

Zonal availability
☒ **Single zone**
 In case of outage, no failover. Not recommended for production.
☐ **Multiple zones (highly available)**
 Automatic failover to another zone within your selected region. Recommended for production instances. Increases cost.

Primary zone
Any

[HIDE ZONES](#)

Customize your instance

You can also customize instance configurations later.

Machine configuration

Machine shapes
Choose from the most commonly used shapes or choose a custom shape. Make sure your selection has enough memory to hold your largest table.

Shared core

☒ 1 vCPU, 0.614 GB
☐ 1 vCPU, 1.7 GB

Data cache
Exclusive to Enterprise Plus edition. Adds a local SSD for read caching to reduce read latency and improve performance at additional cost. [Learn more](#)

☐ Enable data cache

Storage
Storage type is SSD. Storage size is 250 GB, and will automatically scale as needed. Google-managed key enabled (most common).

Pricing estimate

\$0.07 per hour (estimated, without discounts)
That's about \$1.65 per day.
Feature usage and traffic costs aren't included in estimate.

[SHOW COST BREAKDOWN](#)

Summary

Cloud SQL Edition	Enterprise
Region	europe-west1 (Belgium)
DB Version	PostgreSQL 15
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	250 GB
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled
Network throughput (MB/s)	125 of 125
Disk throughput (MB/s)	Read: 120.0 of 125.0 Write: 107.8 of 107.8
IOPS	Read: 7,500 of 12,000 Write: 7,500 of 10,000

Puc. 29.

Вибір потрібного сервісу (PostgreSQL) для зберігання даних зображень на рис. 33.

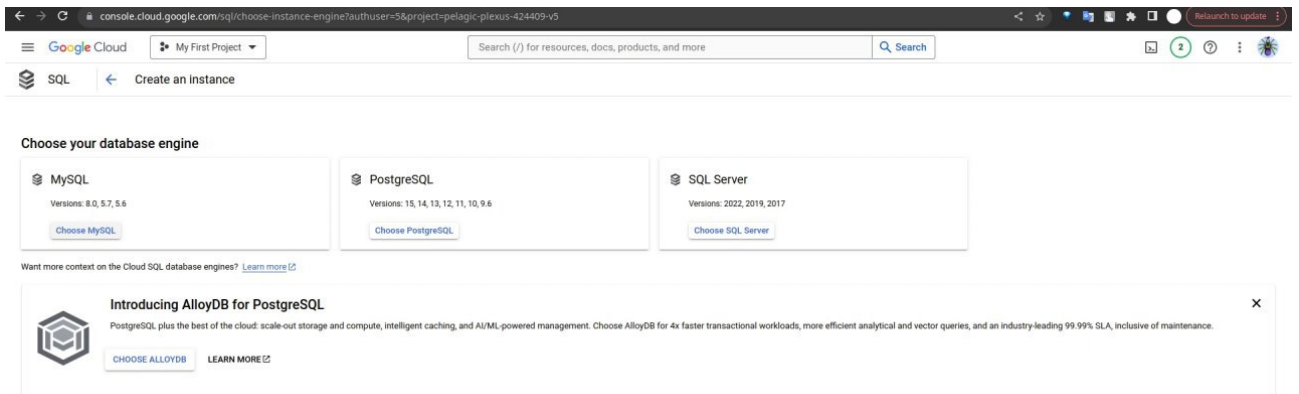


Рис. 30.

Подальше наповнення бази даних публікаціями можна побачити на рис.

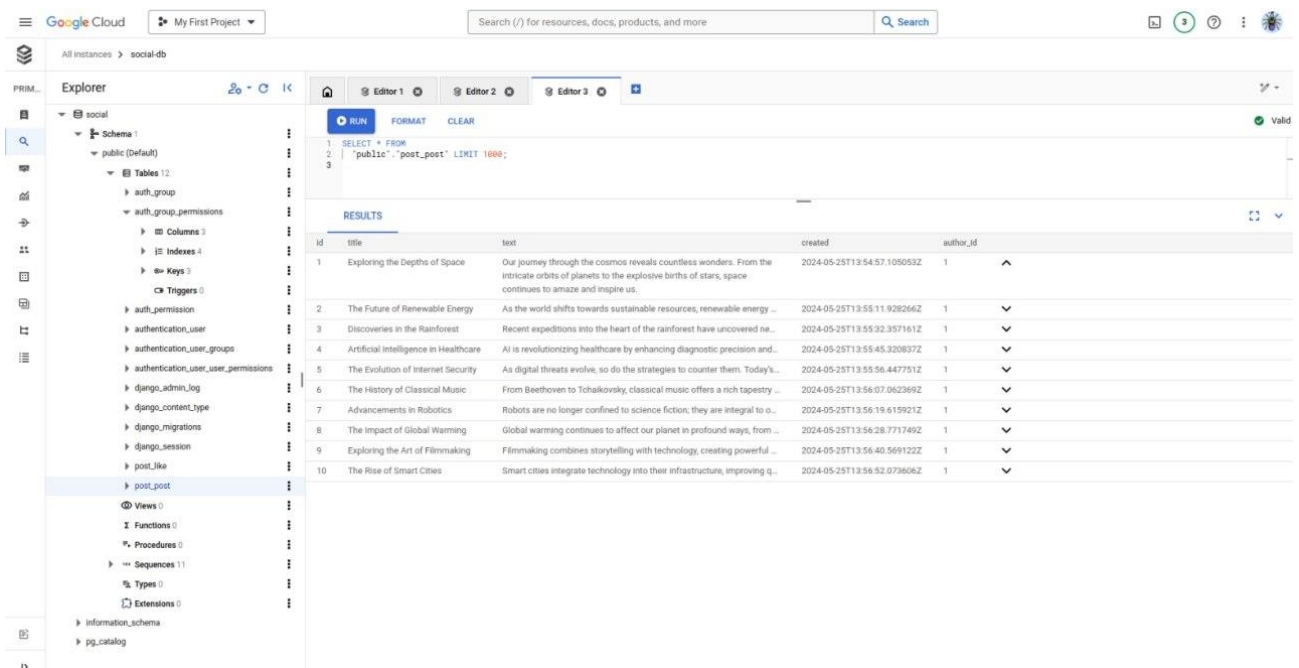


Рис. 31.

На рис. 32-34 зображено процес створення репозиторію для Docker зображень використовуючи GCP Artifact Registry.

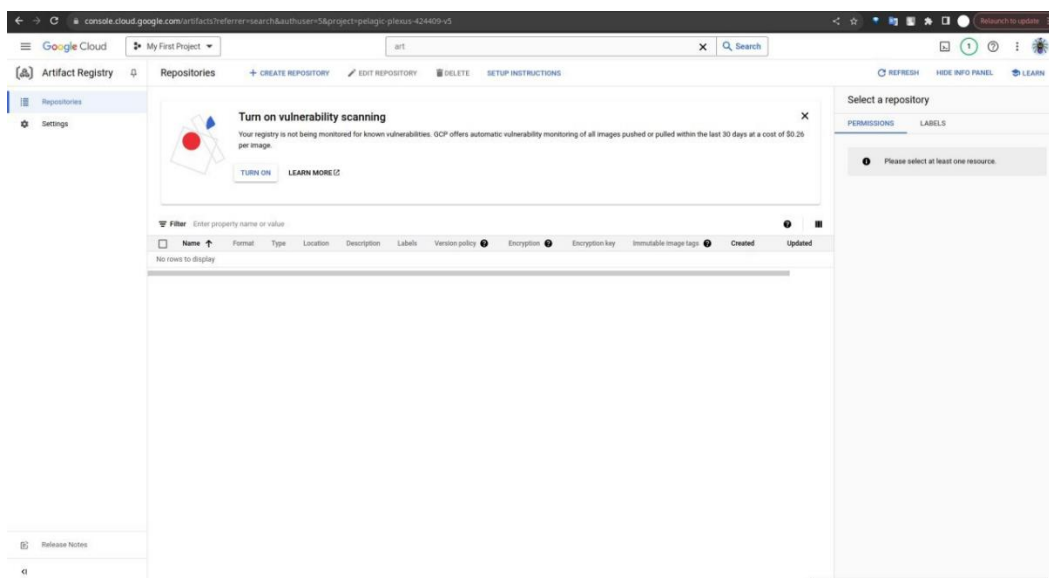


Рис. 32.

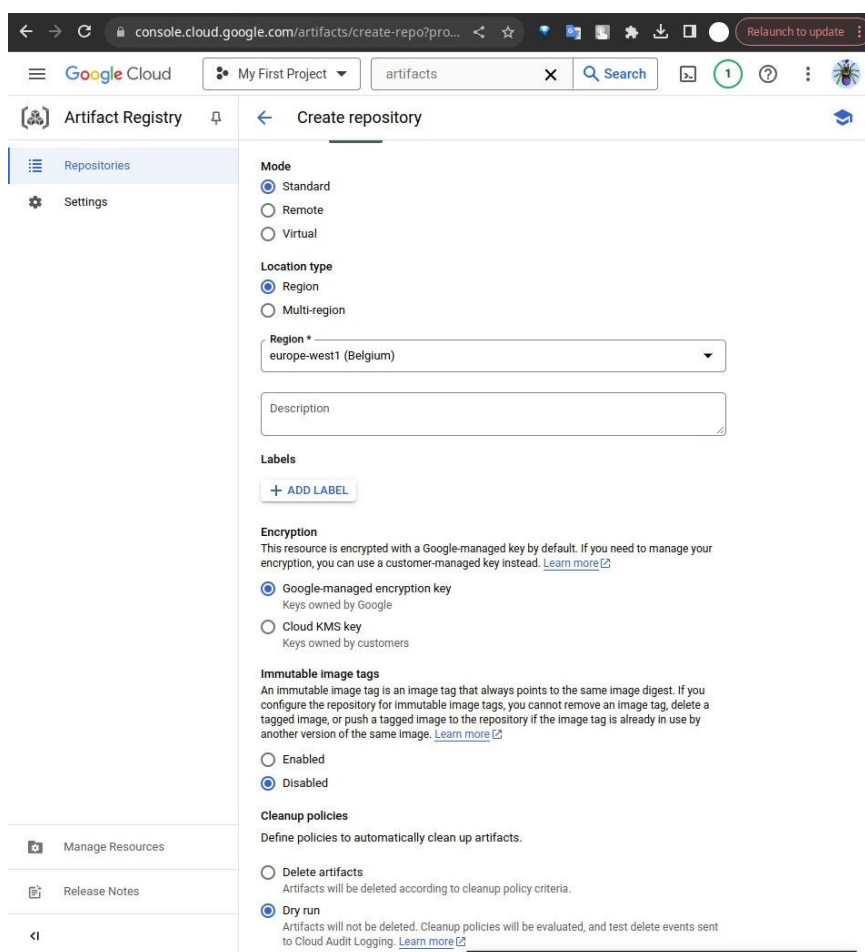


Рис. 33.

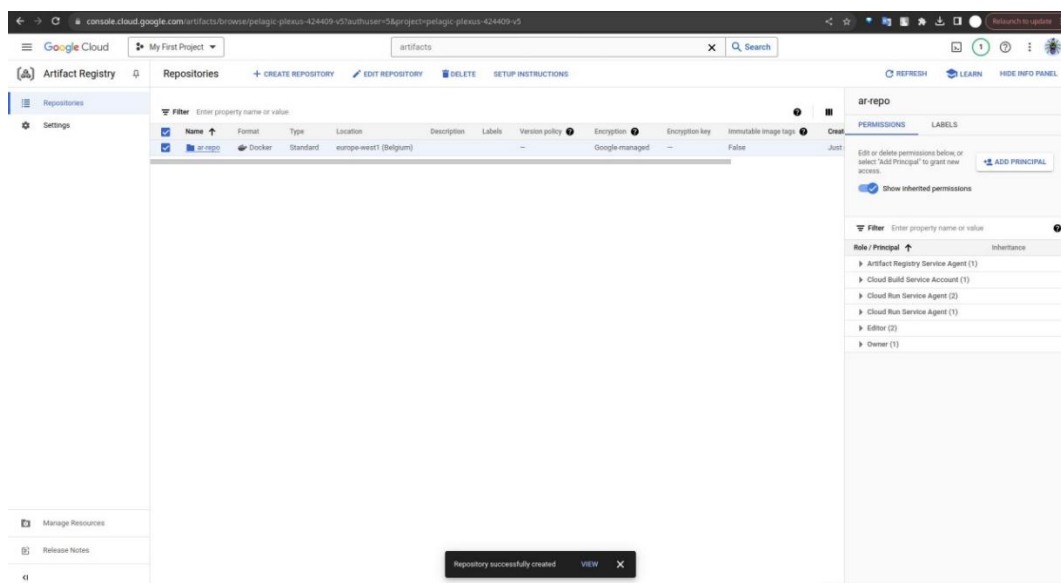


Рис. 34.

3.4. Імплементация практики CI/CD на основі GitHub Actions в хмарному середовищі Google Cloud Platform

Першим кроком (рис. 35-36) необхідно створити сервісний обліковий запис в Google Cloud Platform для подальшого використання в GitHub Actions.

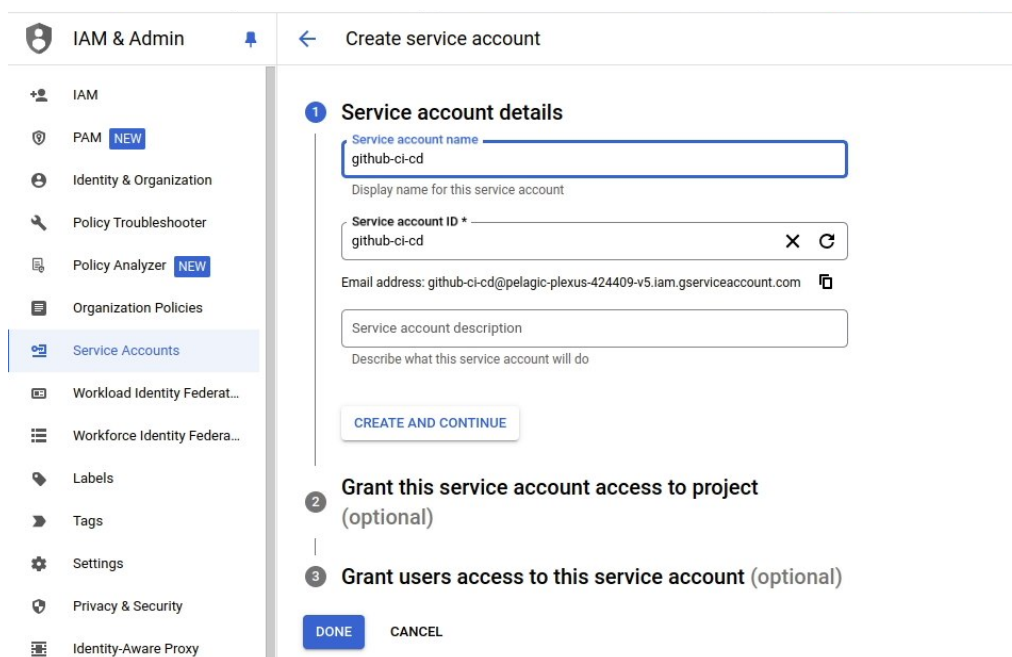


Рис. 35.

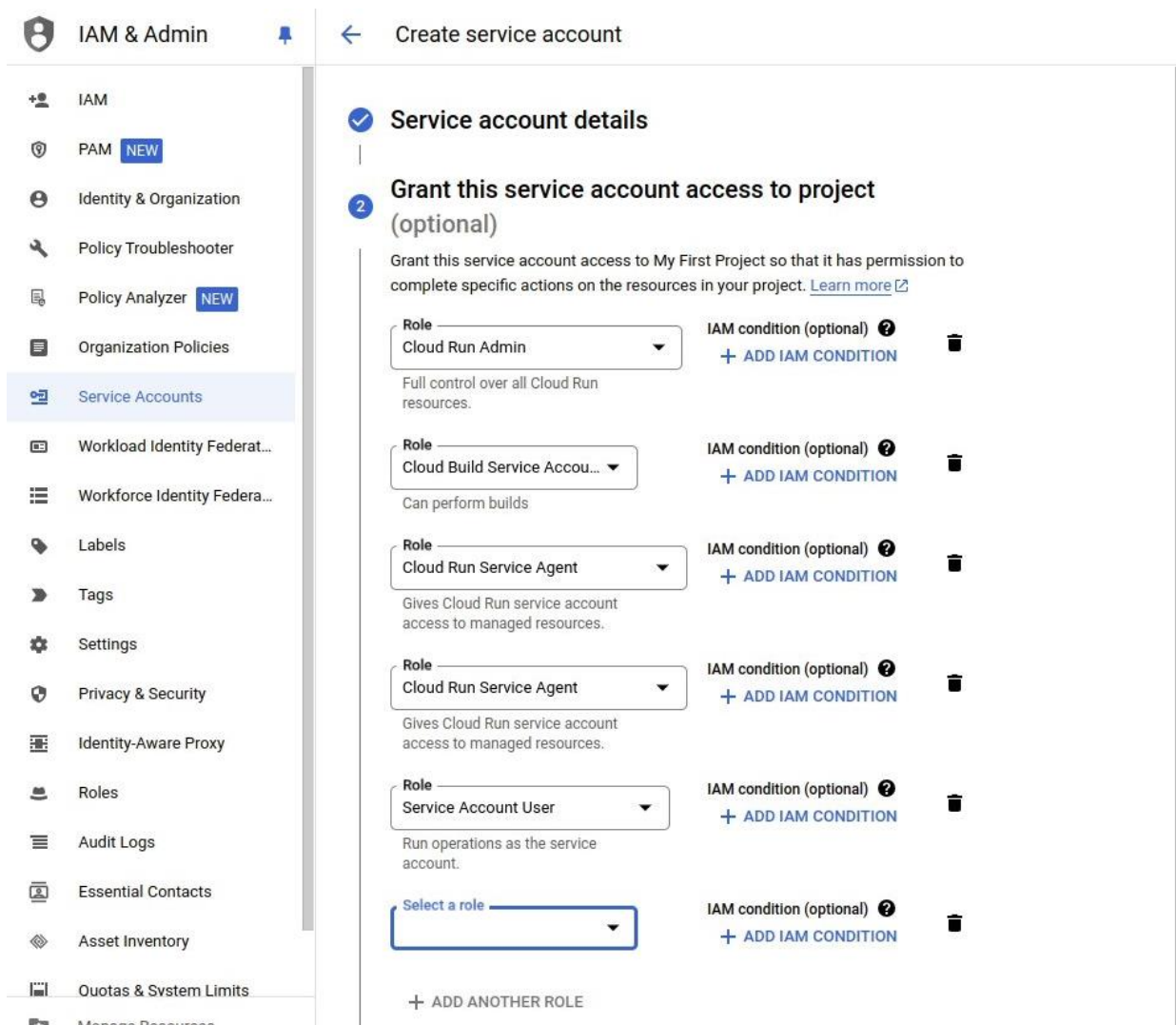


Рис. 36.

Другим кроком (рис. 37-38) необхідно створити приватний ключ для сервісного облікового запису.

Starting June 16, 2024, Google will automatically disable service account keys detected in public repositories. You can customize this behavior by using the 'iam.serviceAccountKeyExposureResponse' organization policy. [Learn more](#)

Service accounts for project "My First Project"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. [Learn more about service accounts](#)

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. [Learn more about service account organization policies](#)

Filter Enter property name or value

☐	Email	Status	Name ↑	Description	Actions
☐	532122358827-compute@developer.gserviceaccount.com	Enabled	Default compute service account		⋮
☐	github-ci-cd@pelagic-plexus-424409-v5.iam.gserviceaccount.com	Enabled	github-ci-cd		⋮

- Manage details
- Manage permissions
- Manage keys
- View metrics
- View logs
- Disable
- Delete

Puc. 37.

Keys

Service account keys could pose a security risk if compromised. We recommend you avoid downloading service account keys and instead use the [Workload Identity Federation](#). You can learn more about the best way to authenticate service accounts on Google Cloud [here](#).

Starting June 16, 2024, Google will automatically disable service account keys detected in public repositories. You can customize this behavior by using the 'iam.serviceAccountKeyExposureResponse' organization policy. [Learn more](#)

Create private key for "github-ci-cd"

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

Key type

☒ JSON
Recommended

☐ P12
For backward compatibility with code using the P12 format.

CANCEL CREATE

Puc. 38.

Третім кроком (рис. 39), необхідно додати секретні змінні для репозиторію на GitHub, яку в подальшому будуть використані в скрипті налаштування Github Actions.

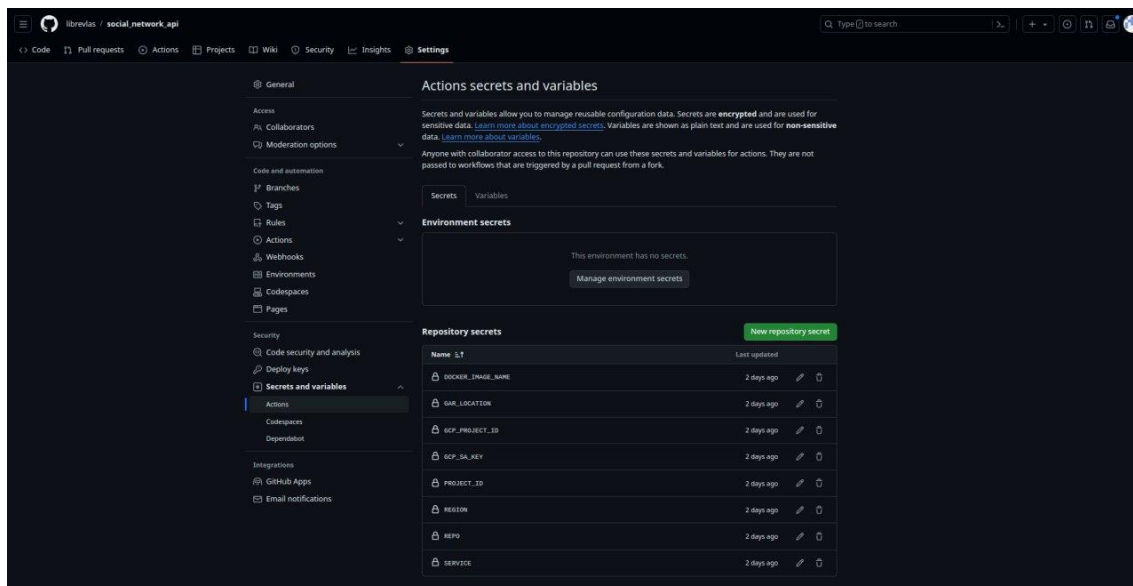


Рис. 39.

Четвертим кроком (рис. 40-41) варто вже створити розгортання backend сервісу власноруч використовуючи в Google Cloud Platform інструмент Cloud Run.

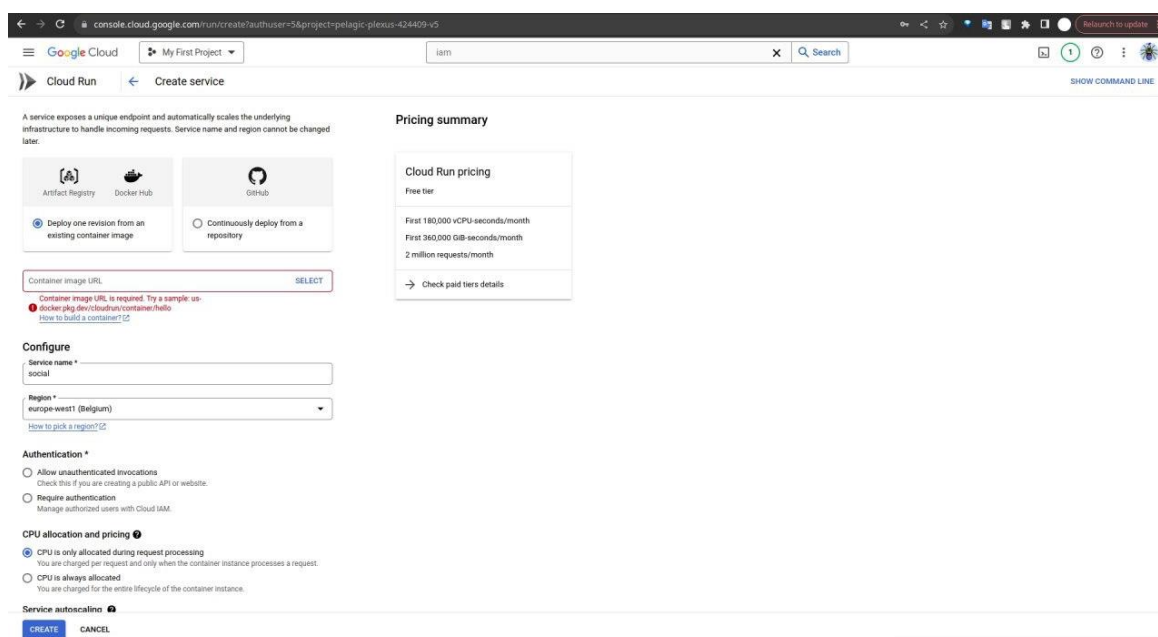


Рис. 40.

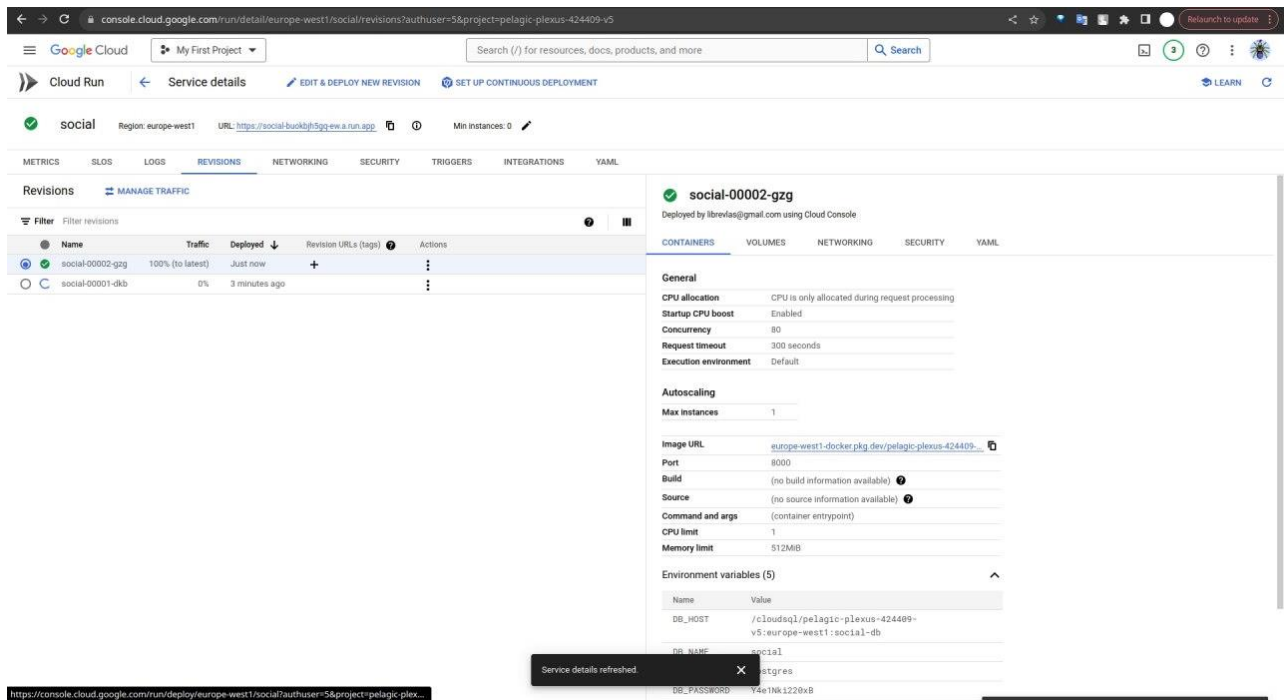


Рис. 41.

На зображенні (рис 42) продемонстрована панель з основними метриками продуктивності системи, такими як:

1. Request count (кількість запитів);
2. Request latencies (кількість затриманих запитів);
3. Container instance count (кількість контейнерів);
4. Billable container instance time (кількість оплачуваного часу);
5. Container CPU utilization (завантаження центрального процесора контейнерами);
6. Container memory utilization (використання пам'яті контейнера).

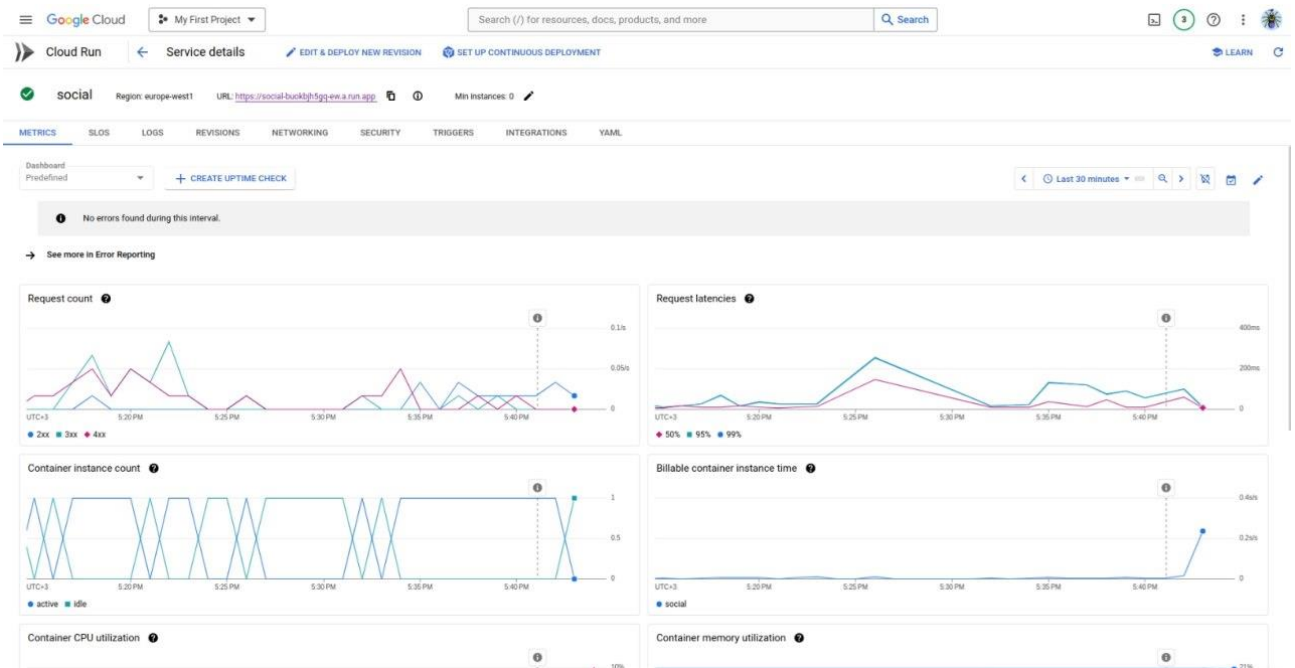


Рис. 42.

Тепер необхідно розібратись з файлом `google-cloudrun-docker.yml` (рис. 43) в якому описано все для налаштування дій GitHub Actions, що виконується під час появи нового commit в основну гілку (master).

```

1  name: Build and Deploy to Cloud Run
2
3  on:
4    push:
5      branches: [ "master" ]
6
7  jobs:
8    deploy:
9      permissions:
10       contents: 'read'
11       id-token: 'write'
12
13     runs-on: ubuntu-latest
14     steps:
15       - name: Checkout
16         uses: actions/checkout@v2
17
18       - name: Google Auth
19         id: auth
20         uses: 'google-github-actions/auth@v0'
21         with:
22           credentials_json: '${{ secrets.GCP_SA_KEY }}'
23
24       - name: Docker Auth
25         id: docker-auth
26         uses: 'docker/login-action@v1'
27         with:
28           username: '$_json_key'
29           password: '${{ secrets.GCP_SA_KEY }}'
30           registry: '${{ secrets.GAR_LOCATION }}-docker.pkg.dev'
31
32       - name: Build and Push Container
33         run: |
34           docker build -t "${{ secrets.GAR_LOCATION }}-docker.pkg.dev/${{ secrets.PROJECT_ID }}/${{ secrets.REPO }}/${{ secrets.SERVICE }}:${{ github.sha }}" ./social_network
35           docker push "${{ secrets.GAR_LOCATION }}-docker.pkg.dev/${{ secrets.PROJECT_ID }}/${{ secrets.REPO }}/${{ secrets.SERVICE }}:${{ github.sha }}"
36
37       - name: Deploy to Cloud Run
38         id: deploy
39         uses: google-github-actions/deploy-cloudrun@v0
40         with:
41           service: '${{ secrets.SERVICE }}'
42           region: '${{ secrets.REGION }}'
43           image: '${{ secrets.GAR_LOCATION }}-docker.pkg.dev/${{ secrets.PROJECT_ID }}/${{ secrets.REPO }}/${{ secrets.SERVICE }}:${{ github.sha }}'
44
45       - name: Show Output
46         run: echo ${{ steps.deploy.outputs.url }}

```

Рис. 43.

Порядковий опис YAML файлу для налаштування GitHub Actions в тандемі з Google Cloud Run.	
<code>name: Build and Deploy to Cloud Run</code>	Назва робочого процесу (workflow).
<code>on: push: branches: ["master"]</code>	Вказує умову яка має запускати workflow. Буде запуск при кожному push в головну гілку "master".
<code>jobs: deploy:</code>	Опис набору задач, які будуть виконані в даному workflow. В нас буде виконувати лише deploy.
<code>permissions: contents: 'read' id-token: 'write'</code>	Цей крок визначає рівень дозволів. В нас є дозвіл на зчитування вмісту репозиторію та дозвіл записувати токени.
<code>runs-on: ubuntu-latest</code>	Налаштування середовища в якому все буде працювати.
<code>steps:</code>	Описує послідовність кроків, які будуть виконані в рамках цього завдання.
<code>- name: Checkout uses: actions/checkout@v2</code>	Використовує попередньо визначену дію actions/checkout@v2 для отримання копії репозиторію.
<code>- name: Google Auth id: auth uses: 'google-github-actions/auth@v0' with: credentials_json: '\${{ secrets.GCP_SA_KEY }}'</code>	Використовує дію google-github-actions/auth@v0 для аутентифікації в Google Cloud за допомогою заданого ключа сервісного облікового запису (\${{ secrets.GCP_SA_KEY }}).
<code>- name: Docker Auth id: docker-auth uses: 'docker/login-action@v1' with: username: '_json_key' password: '\${{ secrets.GCP_SA_KEY }}' registry: '\${{ secrets.GAR_LOCATION }}-docker.pkg.dev'</code>	Використовується дія docker/login-action@v1 для входу в реєстр Docker, використовуючи ключ сервісного облікового запису та налаштування реєстру Docker.
<code>- name: Build and Push Container run: docker build -t "\${{ secrets.GAR_LOCATION }}- docker.pkg.dev/\${{ secrets.PROJECT</code>	Виконує команди для побудови контейнера в Docker та його завантаження в Google Artifact

<pre>_ID }}/\${{ secrets.REPO }}/\${{ secrets.SERVICE }}:\${{ github.sha }}" ./social_network docker push "\${{ secrets.GAR_LOCATION }}- docker.pkg.dev/\${{ secrets.PROJECT _ID }}/\${{ secrets.REPO }}/\${{ secrets.SERVICE }}:\${{ github.sha }}"</pre>	Registry. Образ позначається hash commit, як (\${{ github.sha }}).
<pre>- name: Deploy to Cloud Run id: deploy uses: google-github- actions/deploy-cloudrun@v0 with: service: \${{ secrets.SERVICE }} region: \${{ secrets.REGION }} image: \${{ secrets.GAR_LOCATION }}- docker.pkg.dev/\${{ secrets.PROJECT _ID }}/\${{ secrets.REPO }}/\${{ secrets.SERVICE }}:\${{ github.sha }}</pre>	Використання дії google-github-actions/deploy-cloudrun@v0 для розгортки контейнера в Google Cloud Run, використовуючи вказані параметри сервісу, регіону та образу контейнера.
<pre>- name: Show Output run: \${{ steps.deploy.outputs.url }} echo</pre>	Вивід URL вже розгорнутого сервісу, використовуючи вихідні дані з попереднього кроку розгортання.

Табл. 4: Покроковий опис YAML файлу для налаштування Google Cloud Run в тандемі з GitHub Actions.

На рис. 44 можна побачити статус виконання job після кожного виконання push в головну гілку проекту (master).

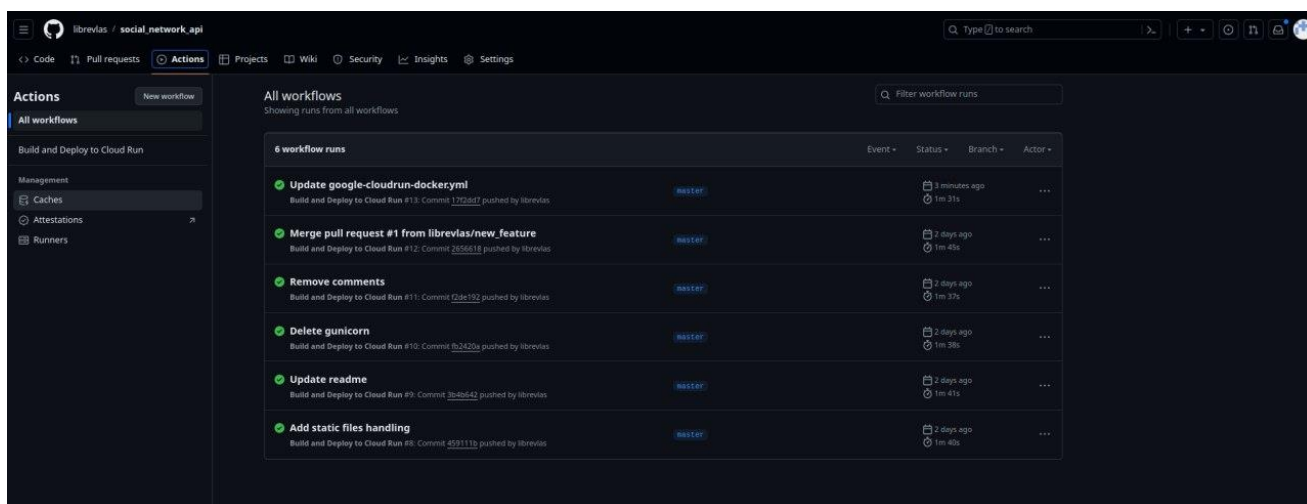


Рис. 44.

Демонстрація одного запуску продемонстрована на рис. 45.

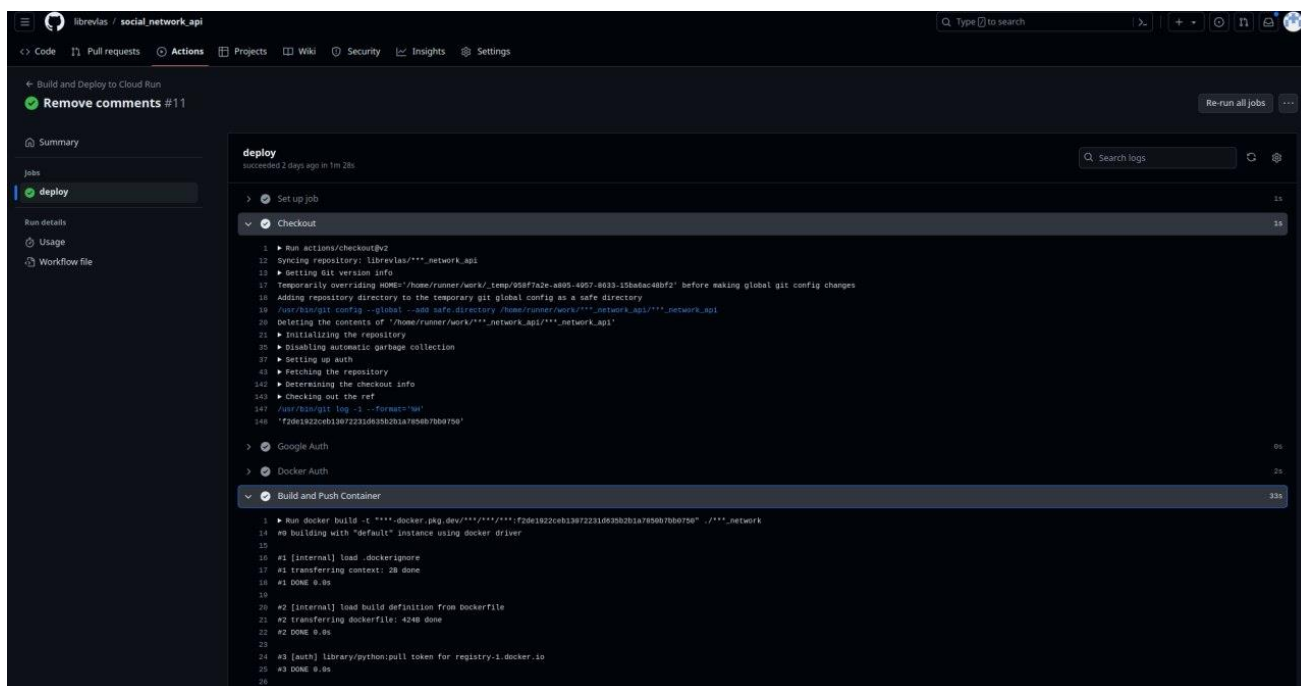


Рис. 45.

Демонстрація результату роботи (рис. 46) всього процесу після повного налаштування.

Run and billable time		
Learn about OS pricing on GitHub Actions		
Job	Run time	Billable time
deploy	1m 28s	0s
	1m 28s	0s

Рис. 46.

На виконання всього процесу в нас пішло лише 1 хв. та 28 с. при використанні GitHub Actions. Якщо робити це все вручну кожного разу, то буде витрачатися 10-15 хвилин часу. Наразі всі процеси автоматизовані. Вони всі поступово виконуються один за одним. В ручному процесі довелося б кожного разу збирати образ проекту, потім завантажувати Artifact Registry на Google Cloud Platform і, на завершення, виконувати перерозгортку на Cloud Run.

Висновки до третього розділу

У третьому розділі ми розглянули ключові аспекти створення, розгортання та автоматизації веб-застосунку, зосереджені на сучасних практиках DevOps. Проведені дослідження та практичні реалізації підтвердили ефективність використання сучасних інструментів та платформ для забезпечення неперервної інтеграції та поставки програмного забезпечення.

Створення веб-застосунку.

Перший підрозділ був присвячений розробці API веб-застосунку. В цьому розділі детально описано принцип його роботи, а саме:

1. Архітектуру веб-застосунку;
2. Технології, що були використані для створення API;
3. Розробка основних функціональних модулів, що забезпечують взаємодію з користувачем та обробку даних.

В результаті, створений API веб-застосунку демонструє просто програму, яку буде взято за основу для імплементації DevOps практик.

Використання Docker.

Другий підрозділ присвячено необхідності використання Docker для контейнеризації застосунків. Основні виконані кроки:

1. Створення Docker-образу: процес створення потрібного Dockerfile для веб-застосунку, що містить всі необхідні залежності та конфігурації.
2. Переваги: виділено переваги контейнеризації (ізоляція середовища, портативність та спрощене розгортання).

Цей підхід значно спрощує управління залежностями та забезпечує узгодженість всього робочого середовища на різних етапах розробки програмного забезпечення.

Google Cloud SQL та GCP Artifact Registry.

Третій підрозділ описує налаштування хмарної платформи Google Cloud Platform:

- Google Cloud SQL: створюємо та налаштовуємо хмарну СУБД, включаючи створення екземпляру (instance). Це забезпечує надійне та масштабоване управління даними на майбутнє.
- GCP Artifact Registry: продемонстровано процес створення та управління артефактами, такими як Docker-образи для API веб-застосунку. Описано налаштування репозиторію та інтеграцію з процесами розробки через GitHub Actions.

Ці сервіси надають ефективне управління даними та артефактами, підвищуючи загальну надійність та безпеку веб-застосунку.

CI/CD з GitHub Actions та Google Cloud Platform.

Четвертий підрозділ присвячений безпосередньому впровадженню неперервної інтеграції та доставки (CI/CD) з використанням GitHub Actions та Google Cloud Platform:

- Автоматизація: опис створення сценаріїв GitHub Actions для автоматичної збірки та розгортання веб-застосунку при кожній зміні в репозиторії.
- Переваги: виділено переваги автоматизації CI/CD, такі як зменшення кількості помилок, швидкість доставки оновлень та покращення загальної ефективності розробки.

Цей підхід забезпечує безперервний цикл розробки, що значно підвищує якість та швидкість виходу нових версій продукту.

ВИСНОВКИ

У магістерській дисертації на тему "Метод автоматизації рутинних завдань DevOps" розглянуто різноманітні аспекти створення, розгортання та автоматизації розробки API веб-застосунку з використанням сучасних інструментів та практик DevOps. Дослідження підтвердили ефективність впровадження таких технологій, як Docker, Google Cloud Platform та GitHub Actions для підвищення продуктивності та надійності процесів розробки програмного забезпечення.

Перший етап роботи був зосереджений на визначенні основних принципів DevOps культури та необхідності її впровадження у виробничий процес підприємства. Проведено глибокий історичний огляд для кращого розуміння причин, що призвели до появи та розвитку DevOps культури. Детально було розглянуто інструментарій для автоматизації рутинних завдань: процесу розробки програмного забезпечення; тестування; оновлення; моніторинг стану всіх процесів. Також, проведений детальний огляд на всі переваги та недоліки використання методології DevOps.

Другий етап магістерської дисертації полягає в огляді того, як саме необхідно користуватись інструментами, що будуть використані безпосередньо в третьому розділі цієї роботи. Також проведений невеликий порівняльний аналіз для визначення оптимальних інструментів для кожного етапу у виробничому процесі, що налаштований по DevOps методології.

Третій етап розробки був присвячений створенню ізольованої програми через Docker та подальшого її використання для впровадження неперервної інтеграції та поставки (CI/CD) з використанням GitHub Actions в хмарному середовищі Google Cloud Platform. Створено повноцінний сценарій GitHub Actions для автоматичної збірки та розгортання веб-застосунку при кожній зміні в репозиторії. Інтеграція з Google Cloud Platform дозволяє автоматично розгорнути контейнери, забезпечуючи безперервну поставку нових версій

застосування. Цей підхід значно підвищив продуктивність та надійність процесу розробки, зменшуючи час на розгортання та підвищуючи якість коду.

Загалом, дисертація підтвердила, що використання методології DevOps для автоматизації рутинних завдань у процесі розробки програмного забезпечення значно підвищує ефективність, надійність та безпеку системи. Інтеграція Docker, Google Cloud Platform та GitHub Actions дозволила створити гнучку та надійну інфраструктуру, що забезпечує зменшення ручної праці, підвищення швидкості поставки оновлень та загальну можливість до масштабованості системи.

Ця робота може служити корисним ресурсом для команд розробників та системних адміністраторів, які прагнуть впровадити сучасні DevOps практики у свої проекти, забезпечивши достатню якість та ефективність робочого процесу розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win» by Gene Kim, Kevin Behr, George Spafford;
2. «A Brief History of DevOps and the Link to Cloud Development Environments» by Laurent Balmelli;
3. «Enterprise DevOps: Leverage ESM data to improve speed and agility» by Jeff Miller;
4. «DevOps & Cloud: The Fuel for Digital Transformation and Innovation» by Gert Jan van Halem;
5. «History of DevOps» by Aniket Gupta;
6. «What is DevOps?»;
<https://www.atlassian.com/devops>
7. «The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations» by Gene Kim, Jez Humble, Patrick Debois, John Willis, Nicole Forsgren;
8. «Complete DevOps Roadmap»;
<https://www.geeksforgeeks.org/devops-roadmap>
9. «DevOps Roadmap»;
<https://roadmap.sh/devops>
10. «What is CI/CD? (Differences, Benefits, Tools, Fundamentals)» by Hamid Akhtar;
11. «What is CI/CD?»;
<https://www.redhat.com/en/topics/devops/what-is-ci-cd>
12. «What is CI/CD security?»;
<https://www.redhat.com/en/topics/security/what-is-cicd-security>
13. «Shift right to test in production»;
<https://learn.microsoft.com/en-us/devops/deliver/shift-right-test-production>

14. «Shift testing left with unit tests»;
<https://learn.microsoft.com/en-us/devops/develop/shift-left-make-testing-fast-reliable>
15. «What is Jenkins?» by Milica Dancuk;
16. «GitHub Actions documentation»;
<https://docs.github.com/en/actions>
17. «Get started with GitLab CI/CD»;
<https://docs.gitlab.com/ee/ci/>
18. «Ansible Documentation»;
<https://docs.ansible.com/ansible/latest/index.html>
19. «How DevOps Monitoring Works: Concepts, Types & Best Practices» by Chrissy Kidd;
20. «Pros and Cons of Docker» by Mark Labrecque;
21. «The Evolution of Virtualization, Servers and Containers» by Ron Flax;
22. Kubernetes Official Documentation;
<https://kubernetes.io/>
23. «What is cloud computing?»
<https://aws.amazon.com/what-is-cloud-computing/>
24. «What is DevOps: Understanding the Role of Cloud in DevOps» by Anirudh V. K.;
25. «Top cloud providers: AWS, Microsoft Azure, and Google Cloud, hybrid, SaaS players» by Larry Dignan;
26. «Disadvantages of Using DevOps»;
<https://www.3pillarglobal.com/insights/disadvantages-of-using-devops>